UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Semántica dinámica de cálculos de sustituciones explícitas a distancia

Tesis presentada para optar al título de
Doctor de la Universidad de Buenos Aires
en el área Ciencias de la Computación

Pablo Barenbaum

Directores de tesis: Eduardo Bonelli
Delia Kesner
Consejero de estudios: Alejandro Ríos

Buenos Aires, 2020

Semántica dinámica de cálculos de sustituciones explícitas a distancia

Los cálculos de sustituciones explícitas son variantes del cálculo-λ en los que la operación de sustitución no se define a nivel del metalenguaje, sino con reglas de reescritura que la implementan. Nuestro principal objeto de estudio es un cálculo de sustituciones explícitas particular, el Linear Substitution Calculus (LSC), definido por Accattoli y Kesner en 2010. Se caracteriza por el hecho de que las reglas de reescritura operan no localmente (*a distancia*). En esta tesis, en primer lugar, definimos máquinas abstractas que implementan estrategias de evaluación en el LSC: call-by-name para evaluación débil y fuerte, call-by-value y call-by-need. Demostramos que dichas máquinas son correctas y preservan la complejidad temporal. En segundo lugar, definimos una extensión de la estrategia de evaluación call-by-need en el LSC para evaluación fuerte. Demostramos que la estrategia es completa con respecto a call-by-name, usando un sistema de tipos intersección no idempotente, y mostramos cómo extenderla para lidiar con *pattern matching* y recursión. Por último, estudiamos la teoría de residuos y familias de radicales en el LSC. Para ello definimos una variante del LSC con etiquetas de Lévy, lo que nos permite demostrar que cumple con la propiedad de *Finite Family Developments*. Aplicamos esta propiedad para obtener resultados de optimalidad, estandarización y normalización de estrategias en el LSC, y generalizamos algunos de estos resultados al marco axiomático de *Deterministic Family Structures*.

Palabras clave: semántica de lenguajes de programación, cálculo-λ, sustituciones explícitas, estrategias de evaluación, evaluación lazy, máquinas abstractas, sistemas de tipos, teoría de residuos.

Dynamic Semantics of Calculi with Explicit Substitutions at a Distance

Explicit substitution calculi are variants of the λ-calculus in which the operation of substitution is not defined at the metalanguage level, but rather implemented by means of rewriting rules. Our main object of study is a particular explicit substitution calculus, the Linear Substitution Calculus (LSC), introduced by Accattoli and Kesner in 2010. Its distinguishing feature is that rewriting rules operate non-locally (*at a distance*). In this thesis, first, we define abstract machines to implement evaluation strategies in the LSC: call-by-name for weak and strong evaluation, call-by-value, and call-by-need. We prove that these machines are correct and that they preserve computational time complexity. Second, we define an extension of the call-by-need evaluation strategy in the LSC for strong reduction. We show that the strong call-by-need strategy is complete with respect to call-by-name, using a non-idempotent intersection type system, and we show how to extend the strategy to deal with pattern matching and recursion. Finally, we study the theory of residuals and redex families in the LSC. To this aim, we define a variant of the LSC endowed with Lévy labels, which allows us to prove that it enjoys the *Finite Family Developments* property. We apply this property to obtain results on optimality, standardization, and normalization for the LSC, and we generalize some of this results to the axiomatic framework of Deterministic Family Structures.

Keywords: programming language semantics, λ-calculus, explicit substitutions, evaluation strategies, lazy evaluation, abstract machines, type systems, residual theory.

Contents

Chapter 1

Introduction

1.1 Computation and λ-Calculi

Computation is about solving problems by mechanically manipulating abstract representations of reality. We are exposed to computation since very early on in our daily lives. To count objects, for example, we may use our fingers as a model of reality: one finger stands for one object. This representation is *abstract* in that it discards all the irrelevant features of the objects, such as size or color, and it keeps only the relevant ones: in this case, the abstract quality we call *quantity*.

Computation is not inseparably tied to modern digital computers. Computers are of course invaluable tools for implementing computational processes, but the study of computation deals primarily with the underlying principles reigning the mechanical manipulation of abstract representations, regardless of their potential implementation. The Babylonians, for example, developed algorithms for solving equations nearly 4000 years before the advent of digital computers [94].

Far from being a purely theoretical endeavor, computation has profound practical consequences. In our times, software is ubiquituous. It governs most aspects of our societies, impacting not only in seemingly personal matters such as entertainment and communication, but also in public affairs such as news broadcasting, monetary transactions, weather forecasting; in safety-critical systems such as medical equipment, nuclear reactors, and avionics; and in sensitive emerging technologies such as self-driving cars and cryptocurrencies.

Traditionally, computation has been associated to the view of processes as mere input–output relations, that is, in the final result that they yield when given particular input data. Contrary to this traditional point of view, computational processes exhibit complex behaviors, and usually many properties besides their output are of significant practical relevance. For instance, how can one be sure that a computational process will not take too long to arrive to the expected answer? How can one be sure that a third-party cannot tamper with a system so that it behaves maliciously? How can one design programming languages so that programs resemble declarative specifications—rather than machine code listings—but in such a way that execution is still efficient?

Developing methods to answer these questions satisfactorily, and to aid the development

of correct programs, is of utmost importance, considering the already mentioned critical nature of software. In the last decades, a vast repertoire of formal methods has been developed, including formal specification languages, automatic theorem provers, program analyzers and synthetizers, and verification techniques such as data-flow analysis, model-checking and abstract interpretation, among others. In this thesis, we are specially interested in the theoretical foundations that support the correct and efficient implementation of *programming languages* and *proof assistants*. These theoretical foundations encompass a broad range of topics in rewriting theory, type theory, and formal semantics.

One of our main concerns is related to the observation that, in general, there may be many different ways to carry out a particular computation. For example, in the expression $2^{1234} * 0$, we may perform the exponentiation first, or we may realize that the result will be 0 regardless of the value of 2^{1234}. The *ways* in which calculations may be carried out are known as *evaluation strategies*. This thesis is about evaluation strategies in a very specific setting: a variant of the λ-calculus called the Linear Substitution Calculus.

In the following subsections, we intend to give a bird's-eye overview of various topics that we will touch on in this thesis.

The λ-calculus
The Static View of the λ-Calculus: as a Logic
The Dynamic View of the λ-Calculus: as a Programming Language
Evaluation strategies
Abstract Machines and Reasonable Cost Models
Weak vs. Strong Reduction Strategies
Normalization
Residuals and Developments
Sharing and Optimality
Explicit substitutions
The Linear Substitution Calculus

1.1.1 The λ-Calculus

In this thesis, the main object of study is a programming language proposed by Beniamino Accattoli and Delia Kesner in 2010 [9], the Linear Substitution Calculus (LSC). The LSC is a descendant of the λ-calculus, and it owes its existence to a long tradition, starting around the beginning of the XX century, when logicians sought to lay out formal foundations for mathematics. In that context, Alonzo Church developed the λ-calculus [36], as a means to formally define the notion of *effectively computable method*, corresponding to the modern notions of *algorithm* or *program*. Other formalisms to define computation were independently developed at about the same time, such as the renowned Turing machines [136].

The λ-calculus is itself an abstract model of computation, in which expressions represent mathematical functions, and execution proceeds by repeatedly transforming or *rewriting* those expressions. Expressions in the λ-calculus are formal syntactical objects called λ-*terms*

(or *terms* for short). An expression of the form $\lambda x.t$ represents a function that maps the variable x to the term t, where t is in turn an expression that might contain occurrences of the variable x. An expression of the form fa denotes the application of the function f to the argument a.

For example, the expression $\lambda x.x$ represents the function that receives a parameter x and returns x, that is, the identity function. Occurrences of the variable x inside an expression of the form $\lambda x.t$ are said to be *bound*. Occurrences of variables that are not bound are said to be *free*. The set of variables that occur free in a term t is usually denoted by $\mathsf{fv}(t)$—for example $\mathsf{fv}(x(\lambda y.yz)) = \{x, z\}$. If a variable is bound, its scope is *local*, so its name is irrelevant to outside observers, and it may be renamed as desired. For example, $\lambda y.y$ is another way of writing the identity function: the terms $\lambda x.x$ and $\lambda y.y$ are formally identified. For this reason, and strictly speaking, λ-terms are not merely expressions, but actually equivalence classes of expressions, modulo renaming of bound variables. The equivalence relation that identifies terms up to the names of its bound variables is called *α-equivalence*. We refer the reader to standard bibliography, for example [95, Ch. 1, Sec. 2], for the precise definition of α-equivalence, which requires some care. Throughout this work we always freely rename bound variables, using *Barendregt's variable convention* [22, 2.1.13]: during definitions and proofs, we may assume that bound variables have been chosen so that their names are apart from free variables and from each other.

In the λ-calculus there is only one possible kind of transformation, known as the *β-reduction* rule:

$$(\lambda x.t)\, s \quad \to \quad t\{x := s\}$$

The β-reduction rule means to reflect one of the most common mathematical practices: it expresses the fact that in order to apply a function ($\lambda x.t$) to an argument (s) one should replace all the occurrences of the formal parameter (x) in the body of the function (t) by the actual argument (s). The expression $t\{x := s\}$ represents the operation of *substitution* of all the free occurrences of the variable x in the term t by the term s. An example computation is given by the following sequence of rewrite steps[1]. Sequences of rewrite steps are sometimes called *derivations* or *reductions*:

$$(\lambda f.f\, 2)(\lambda x.x + x) \quad \to \quad (\lambda x.x + x)\, 2 \quad \to \quad 2 + 2$$

Even though the definition of β-reduction reflects common mathematical practice, the λ-calculus was novel, at the time it was conceived, in that substitution was an explicitly defined operation. Explicit definition allows one to reason rigorously about its behavior.

In this thesis we aim, in fact, to reason rigorously about the behavior of programs. For example, we may want to prove that a certain way of executing a program always reaches a final answer, *i.e.* that it cannot "hang". These properties can be formally stated using the λ-calculus and related calculi, which are based crucially on the notions of substitution and β-reduction. Even though these notions are simple from an intuitive standpoint, defining them precisely is not without pitfalls, and the resulting systems turn out to be surprisingly complex.

[1] The pure λ-calculus does not have a built-in notion of numbers or addition; we use them in this example for clarity of exposition.

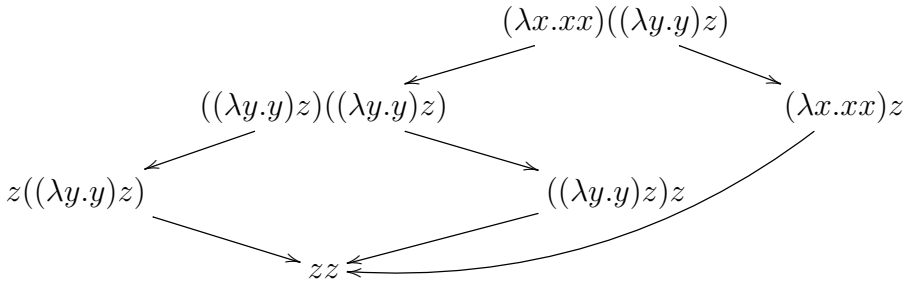
A simple example of the richness of the λ-calculus is that, even though functions formally take only one argument, it still allows to simulate many-argument functions. This can be achieved by resorting to the well-known technique of *currying*, attributed to Moses Schönfinkel [133]. A function $f(x, y)$ of two arguments can be *curried* by regarding it as a single-argument function g such that $g(x)$ is again a single-argument function, and in turn $g(x)(y) = f(x, y)$. More in general, the λ-calculus is *Turing-complete*, which means that it is expressive enough to define all partial computable functions $f : \mathbb{N}^k \to \mathbb{N}$, via a suitable encoding. See for example [22, Section 6.3] for a proof.

Another illustration of the complexity of the phenomena that arise in the λ-calculus is that sequences of rewrite steps can be infinitely long. The term $\Omega := (\lambda x.x\,x)\,(\lambda x.x\,x)$, for instance, may be rewritten to itself in a single β-reduction step: $\Omega \to \Omega$, which leads to the infinite sequence:

$$\Omega \to \Omega \to \Omega \to \dots$$

Ensuring that computations do terminate, under appropriate conditions, is a kind of problem that we encounter very often.

For a further significant example of the kinds of structures that we are interested in, consider the term $(\lambda x.xx)((\lambda y.y)z)$, and note that there are many ways to rewrite it, depending on the order in which computation steps are performed. This gives rise to a *reduction graph*:



Ensuring that computations always reach the same final result, even in the presence of such "forks", is another kind of problem that we often confront. We are also sometimes interested in the question of whether two different computations are *equivalent* in some sense. For example, all the three reduction sequences leading from the term $(\lambda x.xx)((\lambda y.y)z)$ to the final result zz seem to perform the same computational work, albeit in different order. There is indeed a precise notion of equivalence, called *permutation equivalence*, which allows one to identify these three sequences.

One of the most important theorems that we should mention about the λ-calculus is the *Church–Rosser property*, also known by the name of an equivalent property, *confluence*. We use the following standard notation: if R is a binary relation, then R^* stands for its reflexive and transitive closure, and R^{-1} stands for its inverse. One may understand the λ-calculus as an *equational theory* by defining β-equivalence as the least equivalence relation containing β-reduction. More precisely, two terms are said to be β-equivalent, written $t =_\beta s$, whenever $t\ (\to \cup \to^{-1})^*\ s$. The following theorem is due to Alonzo Church and J. Barkley Rosser:

Theorem 1.1 (Confluence). *If $t =_\beta s$ then t and s have a common reduct, that is, there exists a term u such that $t \to^* u$ and $s \to^* u$.*

Proof. See [22, Theorem 11.1.10]. □

A corollary of this theorem is that the λ-calculus is *consistent* as a logic, in the sense that not every β-equality holds—observe for example that x and y do not have a common reduct, so the equality $x =_\beta y$ cannot hold.

During the 1940s and 1950s, the group of Alonzo Church and his collaborators, including Stephen Kleene, J. Barkley Rosser, Haskell Curry, Leon Henkin, and Alan Turing, developed the core metatheory of the λ-calculus. They studied not only the original λ-calculus, but also several of its variants, such as extensions and restrictions of the system, variants endowed with *type systems*, and other related formalisms, such as combinatory logics. See [33] for an excellent overview of the history of the λ-calculus. Standard references for the theory of the λ-calculus itself are the books by Henk Barendregt ([22]) and J. Roger Hindley and Jonathan P. Seldin ([74]).

Interest in the λ-calculus was originally motivated by specific theoretical concerns; in particular, in formally characterizing the notion of effectively computable method. From this point of view, the λ-calculus attains some kind of local optimum: it is simultaneously concise, reasonably readable, and Turing-complete.

Over the years, the λ-calculus has been the object of continued interest from both the logical and the computer science communities, and it is still an active area of research. The wide range of concerns addressed by these communities has shifted the motivations to study the λ-calculus since its conception in the 1930s.

The Static View of the λ-Calculus: as a Logic

In this thesis we are mostly concerned with the ways in which programs may be represented and executed in run-time, so our point of view of the λ-calculus is chiefly a *dynamic* one. Nevertheless, there is a complementary, *static* view of λ-calculi, in which the logical structure of programs is the primary interest. The interplay between the static and the dynamic views plays a major role in Chapters 4 and 5.

The static view of programs has its roots on mathematical logic. More specifically, variants of the λ-calculus provided with *type systems*, are interesting from the logical point of view, because terms can be interpreted as encodings of deductions in formal logical systems. Generally speaking, a *type* is a category (in the ordinary sense of the word) that serves to classify terms according to their inherent structure or their observable behavior.

The notion of type can be traced back to the effort by Bertrand Russell to mend the contradictions found in Gottlob Frege's set theory [128, Appendix B]. The purpose of types in this setting is to stratify the universe of discourse, differentiating between *objects* and *predicates about objects*. This prevents contradictions that spring from diagonalization arguments, such as Russell's celebrated paradox: in the absence of such stratification, one may consider the set of sets $A := \{X \mid X \notin X\}$ and obtain a contradiction by noting that $A \in A$ holds if and only if $A \notin A$ holds.

As part of the attempt to obtain consistent foundations for mathematics, Church introduced his *simple theory of types* in 1940 [37]. In this system, terms of the λ-calculus are

assigned types by means of a formal deductive system. One may assume that there is a set of *basic types* (α, β, γ, etc.) and that for any two types A and B there is an *arrow type* $A \to B$, reserved for functions mapping elements of type A to elements of type B. Variables are intrinsically decorated with their type, writing x^A for an occurrence of a variable of type A. Formal parameters of functions are also decorated with their corresponding type, writing $\lambda x^A.\, t$ for a function that maps a parameter x of type A to the term t. For example, the identity function $\lambda x^A.\, x^A$ is of type $A \to A$, and an operator of function composition might be defined as the term

$$\lambda g^{B\to C}.\, \lambda f^{A\to B}.\, \lambda x^A.\, g^{B\to C}\,(f^{A\to B}\, x^A)$$

whose type is $(B \to C) \to ((A \to B) \to (A \to C))$.

The assignment of types to terms is defined by means of a *typing judgment* of the form $\vdash t : A$, representing the knowledge that the term t is of type A. A term of type A is said to be an *inhabitant* of A. Valid judgments are defined by the following inductive *typing rules*:

$$\frac{}{\vdash x^A : A} \qquad \frac{\vdash t : B}{\vdash \lambda x^A.t : A \to B} \qquad \frac{\vdash t : A \to B \quad \vdash s : A}{\vdash t\, s : B}$$

For example, the third rule states that if we know that t is a term of type $A \to B$ and s is a term of type A, then we may conclude that the application ts is a term of type B.

A remarkable feature of these rules is that types in the simply typed λ-calculus of Church can be thought as *formulas* of propositional logic[2], and terms of type A can be thought as witnesses that the formula A is provable, that is, a *proof* of A. For example, the term $\lambda x^\alpha.\, \lambda y^\beta.\, x^\alpha$ is of type $\alpha \to (\beta \to \alpha)$, and it can be interpreted as a proof of the propositional formula $\alpha \to (\beta \to \alpha)$. A term $\lambda x^\alpha.\, t$ of type $\alpha \to \beta$ corresponds to a proof of the implication $\alpha \to \beta$ that proceeds by assuming a proof x of α as a hypothesis, and then providing a proof t of β, possibly depending on this hypothesis. On the other hand, if t is a proof of the implication $\alpha \to \beta$, and s is a proof of the antecedent α, then ts denotes the proof of β obtained by *modus ponens*.

The realization that types correspond to formulas and terms correspond to proofs is sometimes known as the *Curry–Howard isomorphism* or the *propositions as types correspondence*. The correspondence has far-reaching consequences. It can be extended to other logical constructs besides implication, including conjunction, disjunction, universal and existential quantification, and classical reasoning. It can also be extended to relate other logical/computational phenomena: for instance, reducing a term, *i.e.* performing a computation step $t \to t'$, corresponds to *normalizing* a proof. Achieving this realization took a number of decades, and it is the result of the work of many logicians, mathematicians, and computer scientists, including Haskell Curry [45], Robert Feys [46], Nicolas Goveert de Bruijn [50], and William Howard [76]

Since the 1970s, logical systems have been systematically studied from the type-theoretical point of view suggested by the *propositions as types* paradigm. For instance, second order intuitionistic logic corresponds to a variant of the λ-calculus with parametric polymorphism, known as System F. It was independently discovered by Jean-Yves Girard [59] and John Reynolds [126].

[2]More precisely, formulas in minimal implicational logic.

In 1971, Per Martin-Löf proposed the system of intuitionistic type theory [114] as a foundation for constructive mathematics, which has become a field of research on its own right. The correspondence between propositions and types has been extended to other logical systems, such as classical logic (in various works, remarkably [68], [124], [44]) and linear logic, an influential system defined originally by Jean-Yves Girard in [60]. It has also inspired other systems such as the Calculus of Constructions [41] and Homotopy Type Theory [138].

The study of type systems as encodings of logical deductions laid the basis for the development of proof assistants. Proof assistants are computer programs that allow the user to write (using a formal language) mathematical definitions, statements of theorems, and proofs of those theorems, and verify that the proofs are correct. Many modern proof assistants such as Coq [54], Isabelle [121], and Agda [122] are based on type theory. The efficient implementation of proof assistants poses some of the questions that motivate our work.

Most of the type systems that were mentioned above are examples of type systems *à la* Church. This means that types are an *intrinsic* part of the language. For example, in the simply typed λ-calculus of Church, types are an inseparable aspect of the syntax of terms, hence $\lambda x^{\alpha}.x^{\alpha}$ is the identity function of type $\alpha \to \alpha$, and $\lambda x^{\alpha\to\beta}.x^{\alpha\to\beta}$ is the identity function of type $(\alpha \to \beta) \to (\alpha \to \beta)$. Note that the two variants of the identity function are syntactically different terms. In this system, it does not even make sense to speak of a term without types such as $\lambda x.x$.

There is another very common kind of type system in which types are not an intrinsic part of the syntax of terms. These are known in the literature as type systems *à la* Curry. In these systems, types are rather *extrinsic* annotations, representing properties that terms might or might not have. For example, in the simply typed λ-calculus of Curry, the expression $\lambda x.x$ is a well-formed term representing the identity function. The identity $\lambda x.x$ may be assigned many types: in particular it has type $\alpha \to \alpha$, and it also has type $(\alpha \to \beta) \to (\alpha \to \beta)$, while it does not have type $\alpha \to (\beta \to \alpha)$. In type systems à la Curry typing judgments are typically *hypothetical judgments* of the form $\Gamma \vdash t : A$, where Γ is a *typing context* representing the *hypotheses* required to conclude that the term t has type A. Formally, contexts are lists of pairs of the form $x : A$ giving types to the free variables in t. Typing rules are slightly adapted for the simply typed λ-calculus à la Curry:

$$\frac{(x : A) \text{ is a hypothesis in } \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash t\, s : B}$$

Note that an abstraction $\lambda x.t$ has type $A \to B$ in a given context Γ whenever the body t has type B in the extended context $\Gamma, x : A$.

Intersection types. In most of this thesis, we will not work with type systems, since we are interested in the dynamic view of λ-calculi, and *untyped* λ-calculi are a better fit for this purpose. However, in Chapters 4 and 5, we will make use of *intersection types* to study the dynamic behavior of programs. Intersection types systems are, usually, type systems à la Curry, originally introduced by Mario Coppo and Mariangiola Dezani-Ciancaglini [38] to study termination. These systems are characterized by the presence of a *type constructor*

representing *intersection*: for any two types A and B there is a type $A \cap B$ whose inhabitants are the terms that simultaneously have type A and type B. Intersection type systems also are accompanied with a relation of *inclusion* between types, $A \subseteq B$. Intersection and inclusion respect all the laws that one would expect from their suggestive notation; for example $A \cap B \subseteq A$. Besides the usual three rules of the simply typed λ-calculus, the two following typing rules are added:

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B} \qquad \frac{\Gamma \vdash t : A \quad A \subseteq B}{\Gamma \vdash t : B}$$

The first rule states that a term has type $A \cap B$ if it simultaneously has types A and B. The second one states that a term of type A can be regarded as a term of type B whenever A is a "subset" of B. The remarkable feature of this type system is that typability characterizes exactly the seemingly unrelated property of *normalization*. A term t is said to be *normalizing* if there exists a finite reduction sequence $t \to t_1 \to t_2 \to \ldots \to t_n$ such that t_n is a normal form, *i.e.* there is no reduction step $t_n \to t_{n+1}$. The surprising result is that a term t is normalizing if and only if there exist Γ and A such that $\Gamma \vdash t : A$. See [23, Part 3] for a complete presentation and a survey of many related intersection type systems and their properties.

In typical intersection type systems, the type constructor $(\cap)$ is *idempotent*, that is, the relations $A \subseteq A \cap A$ and $A \cap A \subseteq A$ are both declared to hold. In contrast, in *non-idempotent* intersection type systems, the type constructor $(\cap)$ is not declared to be idempotent. Non-idempotent intersection type systems were originally formulated by Philippa Gardner [58]. Pioneering works on this topic are also by Assaf Kfoury et al. [93, 92] and Daniel de Carvalho [34]. The interest of non-idempotent intersection type systems is that, just as their idempotent counterparts, they allow to characterize normalization properties but, as opposed to their idempotent counterparts, the characterization of normalization provides an explicit decreasing measure. This makes non-idempotent intersection types suitable for analyzing the dynamic properties of reduction in a *quantitative* fashion.

Intersection types play an important role in Chapters 4 and 5 of this thesis, in which we use a system based on non-idempotent intersection types as a technical tool to ensure termination, adapting a technique by Delia Kesner [86].

The Dynamic View of the λ-Calculus: as a Programming Language

As we mentioned before, most of this thesis will be concerned with the *dynamic* view of the λ-calculus and related formalisms. The dynamic view of λ-calculi is interesting from the computational point of view, because it allows us to understand calculi as execution models for programming languages in general, and for *functional* programming languages in particular.

Historically, the understanding of programming language semantics has evolved from a concrete, machine-oriented perspective to an abstract, machine-independent perspective. When the first programming languages were conceived, in the mid-1950s, they were seen as auxiliary tools to mechanically translate mathematical expressions into machine code. A programming language in this sense is a convenient way of abbreviating machine instructions, relying on the remarkable observation that a mathematical expression implicitly encodes a computation mechanism: the post-order traversal of the abstract syntax tree of an arithmetic

expression corresponds closely to a sequence of instructions that calculate its value on a stack-based machine. This was the approach taken in languages like Fortran [18], developed at IBM by John Backus and his team. Although Fortran was a revolutionary breakthrough at the time, it lacked an intrinsic notion of semantics, and it was meant to be understood through the semantics of the target machine.

On the other hand, programming languages can be defined much more abstractly, as formal objects provided with rigorous notions of syntax and semantics. This perspective started to emerge in the 1960s with languages like John McCarthy's Lisp [116], based on the manipulation of symbolic expressions in a way independent of the underlying machine; APL [81], a mathematical notation devised by Kenneth Iverson to describe computational manipulations on arrays; and Algol [112], an international effort to standardize the then incipient algorithmic notation.

In the 1960s, Peter J. Landin published a series of influential papers [99, 100], proposing a family of programming languages, or rather a *framework* for understanding programming language semantics, named Iswim. It was based on the λ-calculus, and it attempted to capture the common mechanism of abstraction underlying all existing programming languages. Landin provided a formal semantics for Iswim by defining an abstract machine to execute Iswim programs, the SECD machine. He also discussed how the semantics of languages like Lisp and Algol could be understood by encoding Lisp and Algol programs in Iswim.

This abstract take on programming language semantics crystallized during the 1970s with the advent of *functional* programming languages. These languages are characterized by being built upon a λ-calculus substratum. Some notable examples are Scheme [134], which introduced some novel ideas like *lexical closures* and *control operators* (in particular, `call/cc`); and the language ML [65], which was the first one to implement a *Hindley–Milner type system*. It is remarkable that, still today, Scheme and ML are among the few relatively mainstream programming languages whose standard specifications include a formally defined semantics (see [131] and [119]). Another influential work from this era was John Backus' 1977 Turing award lecture, *Can Programming be Liberated from the von Neumann style?*, in which he envisioned a style of programming based on the *functional composition* of smaller subprograms. Thereby, programs respect well-defined algebraic laws and they are subject to being reasoned about mathematically.

Since the 1970s, the community surrounding functional programming languages has been growing steadily, which has put forth the λ-calculus as the quintessential programming language. Most aspects of programming language theory, such as extensions with new features and compilation techniques, have been routinely studied by considering suitable adaptations of the λ-calculus. Features that were first conceived and studied in theoretical settings, such as parametric polymorphism, type-inference, inductive data-types, and pattern matching, have been successfully exported to mainstream programming languages.

Let us end this subsection by mentioning a series of influential functional programming languages that were designed in the 1980s and 1990s, namely Hope [32], Miranda [137], and Haskell [83], in which all functions are required to be *pure*. This means that they behave like actual mathematical functions: they are not allowed to perform side-effects like mutation or

external input/output. This restriction brings the execution model of these languages even closer to the pure λ-calculus. Some of these languages are also characterized by the fact that they use *lazy evaluation*, that is, the evaluation of expressions is delayed until their value is actually needed. The evaluation strategy known as *call-by-need* (introduced below in Sec. 1.1.2), closely related with lazy evaluation, is a recurring topic throughout this thesis.

1.1.2 Evaluation Strategies

In most of this thesis we will be interested in studying different "ways" of executing programs. The way of executing a program depends on the execution model of the underlying programming language, whose design space has many dimensions. For example, one dimension that drew the attention of early language designers is that there are various alternatives for implementing the mechanism of *parameter passing*. Suppose that $f(x)$ is a function depending on a formal parameter x, and $f(e)$ represents a function call, where e is some expression. The question is when the argument e should be evaluated, and what should happen when the function f accesses its parameter. Different choices lead to different "ways" of executing programs. These ways are known as *evaluation strategies*. Some evaluation strategies may be more convenient than others in different contexts.

One very common evaluation strategy, *call-by-value*, establishes that e should be evaluated *before* performing the function call. Whenever f needs to use its parameter, it suffices to retrieve the parameter x, which is already bound to a fully evaluated *value*. Another possible evaluation strategy, *call-by-name*, establishes that the function call should be performed first, leaving the expression e unevaluated. If f ever needs to access its parameter, it must retrieve the parameter x, which is bound to an unevaluated expression e, and then proceed to evaluate e to a value.

For example, let $f(x) = x + x$ and suppose that we want to evaluate the function call $f(2 * 3)$. A call-by-value strategy would result in the following execution:

$$f(2 * 3) \quad \rightarrow \quad f(6) \quad \rightarrow \quad 6 + 6 \quad \rightarrow \quad 12$$

In contrast, a call-by-name strategy would result in the following execution:

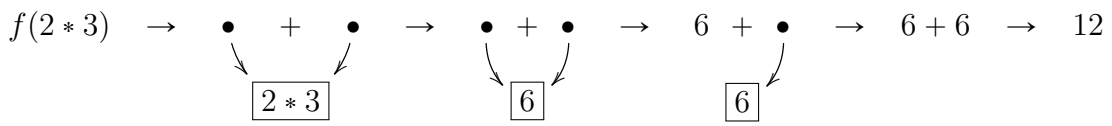
$$f(2 * 3) \quad \rightarrow \quad 2 * 3 + 2 * 3 \quad \rightarrow \quad 6 + 2 * 3 \quad \rightarrow \quad 6 + 6 \quad \rightarrow \quad 12$$

Note that, in call-by-value, the argument e is evaluated *exactly once*, and that holds even if $f(x)$ is a constant function not depending on x. On the other hand, in call-by-name, the argument e is evaluated *as many times as needed*, and that may mean zero, one, or more times. As a consequence, call-by-value may perform *unnecessary* computational work (if the parameter is never used), while call-by-name may *duplicate* computational work (if the parameter is used more than once).

In his PhD thesis, Christopher Wadsworth proposed a mechanism for implementing parameter passing that combined the benefits of call-by-value and call-by-name [145]. This results in the evaluation strategy known as *call-by-need*. Call-by-need establishes that, to evaluate a function call $f(e)$, the call itself should be performed first, leaving the expression e

unevaluated, similarly as in call-by-name. However, all occurrences of the formal parameter x become bound to the same copy of the expression e, which is shared by means of pointers. If f ever needs to access its parameter, the expression e is evaluated once and forever. Subsequent accesses to the parameter, after the first one, merely retrieve the value, similarly as in call-by-value.

Using the call-by-need strategy to evaluate $f(2 * 3)$, defining $f(x) = x + x$ as before, results in an execution that may be graphically depicted as follows, where $\bullet \to \boxed{e}$ represents a pointer to an expression e:



In call-by-need, expressions are not limited to being *trees* anymore. Instead, they must become *directed acyclic graphs*, to account for the sharing of subterms. This means that, in a language like the λ-calculus, it is technically not possible to formulate call-by-need as a strategy, because it relies on a special representation for λ-terms. In 1997, Zena Ariola and Matthias Felleisen [12] and, independently, John Maraist, Martin Odersky, and Philip Wadler [113], defined a variant of the λ-calculus that extends the syntax of λ-terms with a `let` construct that allows to encode the sharing of subterms explicitly. Thus they were able to formulate call-by-need as a strategy internally in this language.

In this thesis we work with the Linear Substitution Calculus, a variant of the λ-calculus extended with *explicit substitutions*. An explicit substitution is a construct of the form $t[x\backslash s]$ which means, informally, that all the occurrences of the variable x in the term t are pointers to the term s.[3] The call-by-need strategy is formulated on terms with explicit substitutions. For example, the call-by-need evaluation of $f(2 * 3)$ as above can be rendered using this notation as follows:

$$f(2 * 3) \to (x + x)[x\backslash 2 * 3] \to (x + x)[x\backslash 6] \to (6 + x)[x\backslash 6] \to 6 + 6 \to 12$$

Call-by-need, as we have mentioned, is a recurring theme of this thesis.

The relevance of formalisms based on rewriting, such as the λ-calculus, for the purpose of studying evaluation strategies, is that they allow to formulate strategies precisely. This in turn makes it possible to reason about their behavior. The somewhat vague notion of evaluation strategy has a rigorous counterpart, namely the notion of *reduction strategy*. A reduction strategy is a function that, given a term t, selects the computation step $t \to s$ that should be performed next, provided that t is not already an *answer*[4]. Reduction strategies have been studied from the theoretical point of view since long ago. Indeed, the notions of call-by-name and call-by-value in the λ-calculus were already known in the 1930s by Church and his collaborators. An influential work from 1975 is by Gordon Plotkin [125], in which

[3] Explicit substitutions and `let` constructs are, in fact, synonyms; an explicit substitution $t[x\backslash s]$ is nothing but a let construct (`let` $x = s$ `in` t), written with different syntax.

[4] The notion of answer may vary depending on the context. For call-by-name and call-by-value, the set of answers is typically the set of terms of the form $\lambda x.t$.

he established a precise relationship between call-by-name and call-by-value, showing how the call-by-name λ-calculus may be simulated in the call-by-value λ-calculus, and vice versa, using *continuation-passing style* translations.

There are other well-known reduction strategies, besides call-by-name and call-by-value, such as leftmost-outermost, innermost, and parallel-outermost, to name a few. Most of these strategies have been also extended to formalisms other than the λ-calculus, such as term rewriting systems and higher-order rewriting systems. Standard reference material dealing with reduction strategies and their properties may be found for example in [22, Ch. 13] or [135, Ch. 9].

Abstract Machines and Reasonable Cost Models

In this thesis, and in particular in Chapter 3 we study the question of whether the Linear Substitution Calculus can be regarded as a *cost model*. The λ-calculus is a fine model of computation from the point of view of mere *computability*, in the sense that a function $\mathbb{N}^k \to \mathbb{N}$ is computable in a Turing machine if and only if it is computable in the λ-calculus. On the other hand, it is not clear whether the λ-calculus is a reasonable model of computation from the quantitative point of view of *computational complexity*. For example, can complexity classes, such as P, NP, EXP, etc. be characterized in terms of reduction in the λ-calculus?

The λ-calculus is an abstract model of computation, but from the point of view of complexity it seems to be *too abstract*. One reason is that execution is based on performing "surgery" on terms, using the metalanguage operation of substitution. Substitution is more complex than it might seem at first sight. On one hand, substitution may cause duplication or erasure of arbitrarily large terms. For example, the reduction step $(\lambda x.x\,x)\,t \to t\,t$ duplicates the arbitrary term t. Moreover, substitution must avoid variable capture; for example $(\lambda y.x)\{x := y\}$ should *not* equal $\lambda y.y$, as that would result in *capturing* the free variable y. Rather, in the expression $(\lambda y.x)\{x := y\}$ the bound variable y should be renamed to a *fresh* variable, for example z. We then have that $(\lambda y.x)\{x := y\} = (\lambda z.x)\{x := y\}$ and, as a result, $(\lambda y.x)\{x := y\} = \lambda z.y$, thus avoiding capture. It is not immediate to implement this kind of operations in a traditional model of computation like a Turing machine or a random-access machine: their execution models are based on a radically different mechanism, namely mutating an array (or a "tape") whose entries contain one of a finite number of symbols.

The fact that the λ-calculus is "too abstract" is, on one hand, of practical concern. The mismatch between the high level of abstraction of the λ-calculus and the lower level of abstraction of actual computers means that implementing the λ-calculus efficiently is a non-trivial task. Understanding and reasoning about these implementations is hindered by the fact that actual computers are built on complex hardware. Instruction sets vary radically from one processor architecture to another, they are usually ridden with corner cases, and their specifications are seldom formal or complete.

In order to bridge the gap between higher-level languages like the λ-calculus and lower-level machines, many *abstract machines* have been proposed. An abstract machine is a formalism defined by a set of possible *states* that the machine might be in, and a binary relation

of *transition* between states. Formally speaking, this is not unlike the definition of a rewriting system (or a directed graph, for that matter), but there is an important difference in intent. Abstract machines are intended to model the relevant aspects of the behavior of an actual computer, thus serving as a stepping stone between programming languages and actual hardware. As a result, the state of an abstract machine is defined in terms of relatively primitive data structures such as stacks and environments. The transitions of the machine manipulate these structures in a similarly primitive fashion. Transitions are (usually) deterministic, and they are expected to be easily implementable in hardware in such a way that they typically take constant time.

Some of the better known abstract machines for the λ-calculus are the SECD machine, introduced by Peter J. Landin in [99] to implement call-by-value evaluation, and the Krivine machine, introduced by Jean-Louis Krivine [97] to implement call-by-name evaluation. Other well-known machines of our interest are Xavier Leroy's ZINC machine [106], designed as the target for an ML compiler, Peter Sestoft's machine for call-by-need evaluation [129], and Pierre Crégut's machine for evaluation to normal form [42].

On the other hand, there is a more profound and theoretical reason why the λ-calculus is, in a way, "too abstract". Consider, for example, the following families of terms $(t_n)_{n\in\mathbb{N}}$ and $(s_n)_{n\in\mathbb{N}}$:

$$\begin{array}{rcl@{\qquad\qquad}rcl} t_0 & \stackrel{\text{def}}{=} & y & s_0 & \stackrel{\text{def}}{=} & y \\ t_{n+1} & \stackrel{\text{def}}{=} & (\lambda x.xx)t_n & s_{n+1} & \stackrel{\text{def}}{=} & s_n s_n \end{array}$$

Observe that the size of the term t_n is $\Theta(n)$ and the size of the term s_n is $\Theta(2^n)$. By induction on n, it is easy to see that t_n reduces to s_n in exactly n steps:

$$t_{n+1} = (\lambda x.xx)t_n \quad \underset{n \text{ steps, by } \textit{i.h.}}{\rightarrow \ \ldots \rightarrow} \quad (\lambda x.xx)s_n \rightarrow s_n s_n = s_{n+1}$$

In contrast, in a traditional model like a Turing machine, the space complexity of a program is always bounded by its time complexity: in particular, the memory occupied by a program cannot grow exponentially relative to its execution time. As a consequence, if an implementation of the λ-calculus encodes λ-terms naïvely as trees, then the execution time required to simulate n reduction steps may require a number of elementary steps exponential in n.

Computational complexity theorists have attempted to capture what it means to be a *reasonable* model of computation. Peter van Emde Boas *Invariance Thesis* [140] proposes that

> *reasonable sequential models simulate each other with polynomial overhead in time and constant factor overhead in space.*

Is the λ-calculus a reasonable model of computation? By the preceding remark, we know that, naïvely encoding terms as trees, Turing-machines cannot simulate the λ-calculus with polynomial overhead. However, this does not forbid that there may exist smarter representations for terms. As a matter of fact, Beniamino Accattoli and Ugo dal Lago [11] have shown that *leftmost-outermost* reduction in the λ-calculus can be reasonably simulated with only polynomial overhead in time, by representing λ-terms in a way that avoids the size explosion

problem by suitably sharing subterms. The representation used in [11] is based on the same calculus that most of this thesis is about—the Linear Substitution Calculus.

Chapter 3 of this thesis is dedicated to studying abstract machines that implement various evaluation strategies in the Linear Substitution Calculus. In particular, the abstract machines that we propose implement their corresponding evaluation strategies preserving time complexity. For example, n steps of call-by-name evaluation in the Linear Substitution Calculus are simulated by $O(c \cdot n)$ transitions of the Krivine machine, where c is a factor proportional to the size of the starting term. Given that the Krivine machine can be reasonably implemented in a traditional model of computation, this in turn justifies that call-by-name evaluation in the Linear Substitution Calculus is a reasonable cost model.

Weak vs. Strong Reduction Strategies

In this thesis, and in particular in Chapters 3, 4 and 5, we design and we study evaluation strategies that are well-suited for the efficient implementation of reduction with open expressions.

In typical programming languages, only *closed* expressions are ever evaluated. An expression is *closed* when it has no free variables. For example, $(\lambda x.\ x + x)\,3$ is a closed expression that evaluates to 6. On the other hand, $(x + x)$ is not a closed expression, because it has free occurrences of the variable x. In a typical programming language like OCaml or Haskell, attempting to evaluate an expression with free variables does not even make sense, and it leads to a compile-time error.

In contrast, the β-reduction rule of the λ-calculus may take place under an *arbitrary context*. This means that, in the λ-calculus, reduction steps may be performed anywhere inside a term, so one may have to deal with *open* expressions, in which variables may occur free. Formally, β-reduction is defined as a binary relation $(\to)$ over the set of λ-terms $\mathcal{T}^{\mathsf{e}}$, that is, $\to \subseteq\ \mathcal{T}^{\mathsf{e}} \times \mathcal{T}^{\mathsf{e}}$, by means of a formal deductive system that includes four inductive rules:

$$\frac{}{(\lambda x.t)s \to t\{x := s\}}\,\beta \qquad \frac{t \to t'}{t\,s \to t'\,s}\,\mu \qquad \frac{s \to s'}{t\,s \to t\,s'}\,\nu \qquad \frac{t \to t'}{\lambda x.t \to \lambda x.t'}\,\xi$$

The first rule, (β), specifies the actual mechanism by which computations proceed. The remaining rules, (μ), (ν) and (ξ), merely specify what are the subexpressions in which computations may take place. Rules like (β), embodying the actual mechanism of computation, are called *rewriting* rules or *computation* rules. Rules like (μ), (ν) and (ξ) are called *congruence* rules. Congruence rules state that a reduction relation like $(\to)$ enjoys certain *closure properties* under a number of term constructors. For instance the (μ) rule allows one to embed a reduction step $t \to t'$ below the context $\Box s$, thus obtaining a reduction step $ts \to t's$. Likewise, the step $\lambda x.z((\lambda y.y)x) \to \lambda x.z\,x$ might be justified by applying the computation rule (β) to conclude that $(\lambda y.y)x \to x$ holds, and then applying the congruence rules (ν) and (ξ) to embed the computation under the context $\lambda x.z\,\Box$. It should be clarified that the distinction between *computation* and *congruence* rules is not clear-cut, but still it is a standard and very useful one.

We have mentioned before that the execution model of traditional programming languages does not make sense for terms with free variables: only *closed* terms may be evaluated. As

a necessary consequence, traditional programming languages do not allow performing computations under arbitrary contexts. The reason is that the subexpression t in the expression $\lambda x.t$ may not be a closed term, since it possibly involves free occurrences of x. As a result, traditional programming languages use *weak reduction*, meaning that the congruence rule corresponding to (ξ) is missing, so that evaluation under lambdas is forbidden. For example, a program like $\lambda x.(\lambda y.y)x$ is already considered to be an answer in a language like OCaml or Haskell, even though $\lambda x.(\lambda y.y)x \to \lambda x.x$ is a valid β-reduction step in the λ-calculus. Evaluation strategies for traditional programming languages, and specifically call-by-name, call-by-value, and call-by-need, all implement weak reduction. Whenever we wish to emphasize the opposition between the usual notion of reduction in the λ-calculus, which allows the (ξ) congruence rule, and weak reduction, which forbids the (ξ) rule, the former is called *strong reduction*.

Evaluation strategies for strong reduction are a mostly neglected topic. A few notable exceptions are Pierre Crégut's strong version of the Krivine Abstract Machine [42] and Benjamin Grégoire and Xavier Leroy's [66] formulation of strong reduction based on the recursive application of a weak evaluator.

However, strong reduction is a central component in the implementation of modern proof assistants based on constructive type theory, such as Coq and Agda. A distinctive characteristic of constructive type theory is the presence of dependent types: expressions that represent types and may depend on terms. For example the proposition stating the fact that the natural number 2 is even may be encoded as a type IsEven(2), whose inhabitants represent witnesses of this fact. In this kind of systems, a type typically has many possible representations; for example IsEven(2) and IsEven(1 + 1) are equal types *by definition*. As a consequence, the type checking engine must perform computations to decide type equality. Moreover, types may depend on *assumptions* in the form of symbolic (free) variables, and they may as well depend on *functions* written using lambda abstractions. This means that reduction must be able to deal with *open* terms, *i.e.* terms that might contain free variables, in full generality, making strong reduction an indispensable feature.

In the last decade, proof assistants have received increasing attention, as a result of such milestones as the formalization of the Four-Color theorem [62] and the Feit–Thompson theorem [64] by the team of Georges Gonthier, the formal verification of the C compiler CompCert [107] by the team of Xavier Leroy, the formalization of the Kepler conjecture [71] by the team of Thomas Hales, and the Univalent Foundations Program [138]. These developments indicate that proof assistant technology has become mature enough to undertake significant projects. On the other hand, the fact that formalization projects have grown larger and more complex has aroused concern regarding the efficiency of proof assistant implementations. Most proof assistants currently rely either on *ad hoc* evaluation mechanisms that are not well-documented, or on relatively straightforward but inefficient mechanisms. Given this situation, it seems apparent that proof assistants could benefit from solid theoretical foundations to support their efficient implementation.

We are concerned with the efficient implementation of strong reduction at various points in this thesis. In Chapter 3, we propose an abstract machine for strong call-by-name reduction,

based on a reformulation of Crégut's abstract machine. Chapter 4 is devoted to studying an extension of the call-by-need strategy for strong reduction.

Normalization

In this thesis we are interested in developing reduction strategies for evaluating programs, and in showing that these strategies are "good" in various precise senses. One important question is whether a reduction strategy is *normalizing*. Informally, a reduction strategy is normalizing if following the strategy always leads to a answer, whenever possible. For example, defining $I = \lambda x.x$ as the identity function, and $\Omega = (\lambda x.xx)(\lambda x.xx)$ as the non-terminating term *par excellence*, then following the call-by-value strategy does not terminate for the term $(\lambda x.I)\Omega$, for it repeatedly leads us to choose to evaluate the argument Ω:

$$(\lambda x.I)\Omega \to (\lambda x.I)\Omega \to (\lambda x.I)\Omega \to \dots$$

This example attests that the call-by-value strategy is not normalizing, because it fails to reach an answer, even though reaching an answer is possible. Contrast this with what happens using the call-by-name strategy, which reaches an answer in just one step:

$$(\lambda x.I)\Omega \to I$$

Indeed, the call-by-name strategy is normalizing in general, which is a well-known fact[5].

In Chapters 4 and 5 we prove, using a different technique, that a *strong* variant of the call-by-need strategy is normalizing. In Chapter 7 we give sufficient conditions to ensure that a strategy is normalizing, and we apply it to a variant of the call-by-need strategy.

Residuals and Developments

In order to prove that reduction strategies are "good" in various senses, one must confront more fundamental questions. For example, one may want to prove that a given computation requires to perform less computational work than any equivalent computation. This leads to a fundamental question: when can one say that two computations are equivalent?

In this thesis, especially in Chapter 7, we use an established notion of equivalence between sequences of rewriting steps, the notion of *permutation equivalence*, due to Jean-Jacques Lévy [109]. Informally, two sequences of rewriting steps are permutation equivalent if they perform essentially the same computation steps, although possibly in different order. For example the following three reduction sequences are permutation equivalent:

$$\begin{array}{lllllll}
(\lambda x.xx)((\lambda y.y)z) & \to & ((\lambda y.y)z)((\lambda y.y)z) & \to & z((\lambda y.y)z) & \to & zz \\
(\lambda x.xx)((\lambda y.y)z) & \to & ((\lambda y.y)z)((\lambda y.y)z) & \to & ((\lambda y.y)z)z & \to & zz \\
(\lambda x.xx)((\lambda y.y)z) & \to & (\lambda x.xx)z & \to & zz & &
\end{array}$$

To define permutation equivalence precisely, the notion of *residual* has to be introduced. First, a *redex* in the λ-calculus is a subterm of the form $(\lambda x.t)s$. For example, the term

[5] See for instance [130, Corollary 1.5.12 (i)]. In Coro. 7.56 we prove a related result.

$\underline{(\lambda x.xx)}(\overline{(\lambda y.y)z})$ has two redexes, respectively underlined and overlined. Each redex is associated with one—and only one—computation step. For example, contracting the underlined redex corresponds to the step:

$$(\lambda x.xx)((\lambda y.y)z) \to ((\lambda y.y)z)((\lambda y.y)z)$$

while contracting the overlined redex corresponds to the step:

$$(\lambda x.xx)((\lambda y.y)z) \to (\lambda x.xx)z$$

Conversely, each computation step is associated with one—and only one—redex, so sometimes, both in the literature and in this thesis, redexes and steps are identified. If $R : t \to s$ and $S : t \to u$ are steps going out from the same starting term t, the *set of residuals of* S *after* R, denoted by S/R, can be defined (semi-formally) as follows:

1. Mark the lambda of the redex S in the starting term t (for example by underlining it).
2. Execute the step R on the marked term, obtaining the target s of the step R, which now has some marked lambdas.
3. A step S' starting from s is a residual of S after R, that is $S' \in S/R$, if and only if the lambda of S' is marked.

For example, let:

$$\begin{array}{lrcl} R: & (\lambda x.xx)((\lambda y.y)z) & \to & ((\lambda y.y)z)((\lambda y.y)z) \\ S: & (\lambda x.xx)((\lambda y.y)z) & \to & (\lambda x.xx)z \end{array}$$

The following diagram justifies that R has exactly one residual after S, more precisely $R/S = \{R'\}$:

$$\begin{array}{ccc} (\underline{\lambda}x.xx)((\lambda y.y)z) & \xrightarrow{S} & (\underline{\lambda}x.xx)z \\ \Big\downarrow{\scriptstyle R} & & \Big\downarrow{\scriptstyle R'} \\ ((\lambda y.y)z)((\lambda y.y)z) & & zz \end{array}$$

In turn, the following diagram justifies that S has two residuals after R, more precisely $S/R = \{S_1, S_2\}$:

$$\begin{array}{ccccc} (\lambda x.xx)((\underline{\lambda}y.y)z) & & \xrightarrow{\qquad R \qquad} & & ((\underline{\lambda}y.y)z)((\underline{\lambda}y.y)z) \\ \Big\downarrow{\scriptstyle S} & & & \swarrow{\scriptstyle S_1} & \qquad\qquad \searrow{\scriptstyle S_2} \\ (\lambda x.xx)z & & z((\underline{\lambda}y.y)z) & & ((\underline{\lambda}y.y)z)z \end{array}$$

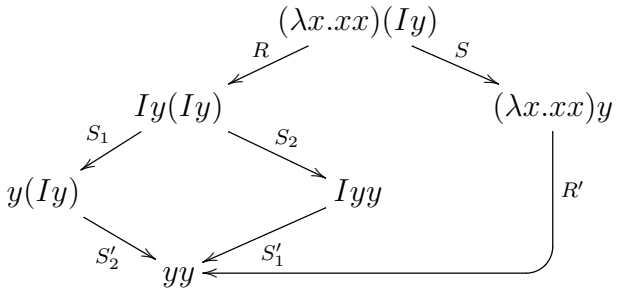
Note that a step may have zero, one, or more residuals. For example, S has more than one residual after R, in which case we say that R *duplicates* S. On the other hand, if we let $R : (\lambda x.y)(Iz) \to y$ and $S : (\lambda x.y)(Iz) \to (\lambda x.y)z$ then S has no residuals after R, that is, $S/R = \varnothing$, in which case we say that R *erases* S. If R' is a residual of R after S, we say that R is an *ancestor* of R' before S.

Another important phenomenon is *creation*. In a sequence of two steps RS, we say that R *creates* S if it has no ancestor before R. In the following three examples, the second step is created by the first step:

$$\begin{array}{llll} (\lambda x.x)(\lambda y.zy)t & \to & (\lambda y.zy)t & \to & zt \\ (\lambda x.\lambda y.zxy)ts & \to & (\lambda y.zty)s & \to & zts \\ (\lambda x.z(xz'))(\lambda y.t) & \to & z((\lambda y.t)z') & \to & z\,t\{y := z'\} \end{array}$$

To define residuals formally, one may proceed as above, introducing an auxiliary λ-calculus with marked lambdas (as we do in Def. 2.70), or directly by case analysis [46, pp. 115–116]. In any case, one obtains the same notion of residual.

The set of residuals S/R can be generalized to the case in which, rather than a single step R, one has a sequence $R_1 \dots R_n$. Namely, one declares that $S_n \in S_0/R_1 \dots R_n$ if and only if there exist steps $S_1, \dots, S_{n-1}$ such that $S_i \in S_{i-1}/R_i$ for all $1 \leqslant i \leqslant n$. This allows one to give the following notion of *development*. Let $\mathcal{M}$ be a set of steps, all starting from the same initial term t. A possibly infinite sequence $R_1 \dots R_n \dots$ is a *development* of $\mathcal{M}$ if for every $1 \leqslant i \leqslant n$ there exists a step $S \in \mathcal{M}$ such that $R_i \in S/R_1 \dots R_{i-1}$. For example, in the diagram below, the sequence RS_1S_2', the sequence RS_2, and the sequence SR' are three different developments of $\{R, S\}$:



Moreover, we say that a development of a set $\mathcal{M}$ is *complete* if it is maximal. For instance, RS_1S_2' is a complete development of $\{R, S\}$, while RS_2 is not a complete development of $\{R, S\}$ because it may be extended to form a longer development RS_2S_1' of the set $\{R, S\}$.

Note that some sequences are not developments of any set $\mathcal{M}$. For example, let RS be any sequence such that R creates S, *e.g.* $IIx \to Ix \to x$. Then RS cannot be the development of any set.

One of the most important theorems about developments is the Finite Developments theorem, stated below. The first item was already known to Church and Rosser [75], while the second and third items are due to Lévy [109]:

Theorem 1.2 (Finite Developments)**.** *Let $\mathcal{M}$ be a set of steps with the same source t in the λ-calculus. Then:*

1. **Finite.** *There are no infinite developments of $\mathcal{M}$.*
2. **Cofinal.** *If ρ and σ are two complete developments of $\mathcal{M}$, they have the same target, that is there exists a term s such that $\rho, \sigma : t \to^* s$.*

3. **Equivalent.** *If ρ and σ are two complete developments of $\mathcal{M}$ and $T : t \to t'$ is any step then T/ρ and T/σ are the same set.*

Proof. See [109, p. 33]. □

Relying on the Finite Developments theorem as a cornerstone, an equivalence relation on reduction sequences may be defined. Permutation equivalence ($\equiv$) is the least equivalence relation such that $\rho\sigma\tau \equiv \rho\sigma'\tau$ for any derivations $\rho, \sigma, \sigma', \tau$ such that σ and σ' are complete developments of the same set $\mathcal{M}$.

There are many alternative ways to characterize permutation equivalence [118, 135, 142].[6] One way is by proposing a *standardization procedure*, which converts an arbitrary sequence of rewriting steps into *standard form*. A reduction in standard form is the canonical representative of its permutation equivalence class, hence two reduction sequences are permutation equivalent if and only if they have the same standard form.

Sharing and Optimality

In Chapters 6 and 7 we will study reduction strategies from the point of view of optimality, *i.e.* on whether they yield *optimal* reductions. There are two related but slightly different senses of the word *optimality*. For clarity, we distinguish these two meanings by referring to them as *length-optimality* and *work-optimality* respectively.

On one hand, a reduction $t_0 \to t_1 \to \ldots \to t_n$ from a term t_0 to an answer t_n is said to be *length-optimal* if it is the shortest reduction leading from t_0 to an answer. In the λ-calculus, defining a reduction strategy that yields length-optimal reductions in this sense is trivial from a strictly mathematical point of view. Unfortunately, as one may suspect, there is no *computable* length-optimal strategy [22, Prop. 13.5.2].

On the other hand, a reduction $t_0 \to t_1 \to \ldots \to t_n$ from a term t_0 to an answer t_n is said to be *work-optimal* if it does not duplicate computational work and it does not perform unnecessary computational work.

Questions related to optimality, in both of the senses, are far from straightforward to answer. In fact, the notion of work-optimality is not even straightforward to define, as it requires to formally specify what it means to say that computational work be *duplicated* or *unnecessary*. For the moment we content ourselves with this informal definition of work-optimality. The question of optimal reduction was first studied in the 1970s by Jean Vuillemin [144], John Staples [132], and Jean-Jacques Lévy [109, 110] together with Gérard Berry [27], and later extended and generalized by others.

One may expect that the notions of length-optimality and work-optimality coincide. However, in the λ-calculus there are terms that admit length-optimal reductions, but no work-optimal reductions. Consider for example the following term, where $I = \lambda x.x$ stands for the identity as usual:

$$(\lambda x.x(xI))(\lambda y.(\lambda z.zz)(yI))$$

[6] In Def. 2.40 we give the formal definition of permutation equivalence that we use. In Lem. 2.59 we recall a useful alternative characterization.

There are only finitely many reductions $(\lambda x.x(xI))(\lambda y.(\lambda z.zz)(yI)) \to \ldots \to I$ so it is easy to see that there exists a length-optimal reduction. Moreover, there are only two possibilities for the first step, and it can be checked that none of them leads to a work-optimal reduction:

1. On one hand, we may reduce the expression that is underlined in the diagram below. But doing so duplicates the computational work required to evaluate the overlined expression, necessarily leading to a reduction that is not work-optimal:

$$\underline{(\lambda x.x(xI))(\lambda y.\overline{(\lambda z.zz)(yI)})} \to (\lambda y.\overline{(\lambda z.zz)(yI)})((\lambda y.\overline{(\lambda z.zz)(yI)})I)$$

2. On the other hand, we may reduce the expression that is underlined in the diagram below. But this duplicates the overlined subexpression yI, and this in turn leads to duplicating the computational work to evaluate II:

$$\begin{array}{rcl}(\lambda x.x(xI))(\lambda y.\underline{(\lambda z.zz)(\overline{yI})}) & \to & (\lambda x.x(xI))(\lambda y.(\overline{yI})(\overline{yI})) \\ & \to & (\lambda y.(yI)(yI))((\lambda y.(\overline{yI})(\overline{yI}))I) \\ & \to & (\lambda y.(yI)(yI))((\overline{II})(\overline{II}))\end{array}$$

With regard to the relationship between optimality and other evaluation strategies, it can be noted that call-by-name and call-by-value do not necessarily yield work-optimal reductions. The call-by-name strategy is not optimal because it may duplicate work, as in the following example, in which the underlined expression is duplicated and then evaluated twice:

$$(\lambda x.xx)(\underline{II}) \to \underline{II}(\underline{II}) \to I(II) \to II \to I$$

The call-by-value strategy is also not optimal, because it may perform unnecessary work, as in the following example, in which the underlined expression is evaluated, even though it is not needed:

$$(\lambda x.I)(\underline{II}) \to (\lambda x.I)I \to I$$

More in general, Lévy showed in his PhD thesis that no reduction strategy consistently yields work-optimal reductions for the λ-calculus [109]. Nevertheless, this does not rule out the possibility that an implementation of optimal reduction may exist. An optimal implementation, should it exist, would need to be based on another representation for λ-terms, other than trees. For example, one may conceive representing terms using graphs, as was done for call-by-need.

Considering the impossibility results that we have mentioned so far, it is perhaps surprising that it is actually possible to define an effective optimal implementation for the λ-calculus. In his thesis, Lévy gave sufficient conditions that an evaluation mechanism should meet in order to ensure work-optimality, whenever possible. John Lamping later proposed an effective implementation [98], based on *sharing graphs*, that fulfills these conditions, yielding an optimal implementation of the λ-calculus.

We return to the topic of optimality in later chapters. Studying optimality for the Linear Substitution Calculus is one of the primary motivations behind Chapters 6 and 7.

1.1.3 Explicit Substitutions

As we have mentioned before, the main object of study in this thesis is a variant of the λ-calculus called the Linear Substitution Calculus. The λ-calculus has one rewriting rule, the β-reduction rule:

$$(\lambda x.t)s \to t\{x := s\}$$

Its definition relies on the auxiliary operation of *substitution*, written $t\{x := s\}$, which belongs to the metalanguage.

The operation of substitution is too coarse-grained. The notation $t\{x := s\}$ suggests that all the free occurrences of x are simultaneously replaced by s. Implementations, however, rarely perform the textual replacement of the formal parameter x by the actual argument s simultaneously. Instead, they rely on an auxiliary data structure, called an *environment*, that keeps track of *variable bindings*. An implementation of the β rule would typically create an association $[x \mapsto s]$ in the environment, mapping the variable x to the value s. This creates a significant gap between theory and practice.

In order to bridge this gap, many works have considered extensions of the λ-calculus incorporating a construct to allow for *local definitions*, a feature known by various names (such as "`let` constructs", "closures", or "explicit substitutions", among other names, depending on the point of view). For example, Nicolas Goveert de Bruijn [51] extends the λ-calculus with a facility to define constants, and Pierre-Louis Curien [43] studies a calculus of closures in order to model *environments*. A milestone paper in this line was by Martin Abadi, Luca Cardelli, Pierre-Louis Curien and Jean-Jacques Lévy [1], in which they propose a calculus with explicit substitutions, the $\lambda\sigma$-calculus.

During the 1990s, a plethora of calculi with explicit substitutions emerged, including λx by Kristoffer Rose [127, 29], λs by Fairouz Kamareddine and Alejandro Ríos [84], $\lambda\chi$ by Pierre Lescanne and Jocelyne Rouyer-Degli [108], $\lambda\upsilon$ by Zine-El-Abidine Benaissa et al. [24], and many other calculi. Their defining characteristic is that they include a rewriting rule corresponding to the β-reduction rule in the λ-calculus, sometimes called `beta`:

$$(\texttt{beta}) \qquad (\lambda x.t)s \;\to\; t[x\backslash s]$$

with the difference that $t[x\backslash s]$ is an *explicit substitution* operator, internal to the object language. This formally means that the syntax of terms is extended to include not only variables, applications, and abstractions, but also explicit substitutions of the form $t[x\backslash s]$. In order to implement the explicit substitution operator, these calculi include also other rewriting rules that indicate the mechanism by which substitutions are performed. For example, a typical calculus with explicit substitutions may include rewriting rules to specify how substitutions should act when confronted with variables, and how they should propagate over abstractions and distribute over applications:

$$\begin{array}{lllll}
(\texttt{var1}) & x[x\backslash t] & \to & t & \\
(\texttt{var2}) & y[x\backslash t] & \to & y & \text{if } x \neq y \\
(\texttt{abs}) & (\lambda y.t)[x\backslash s] & \to & \lambda y.t[x\backslash s] & \text{if } x \neq y \text{ and } y \notin \mathsf{fv}(s) \\
(\texttt{app}) & (ts)[x\backslash u] & \to & t[x\backslash u]\, s[x\backslash u] &
\end{array}$$

In fact, the `beta` rule plus the four rewriting rules `var1`, `var2`, `abs`, and `app` form the system known as λx.

An interesting consequence of including substitutions explicitly in the object language is that it allows one to model the *sharing* of subterms. For example, in the following reduction sequence,

$$(\lambda x.yxx)((\lambda z.z)y) \to (yxx)[x\backslash(\lambda z.z)y] \to (yxx)[x\backslash z[z\backslash y]]$$

the first reduction step binds the variable x to a term $(\lambda z.z)y$. Here it is appropriate to think of the variable x as a pointer referencing a memory location, and of the explicit substitution $[x\backslash(\lambda z.z)y]$ as the memory cell itself. The second reduction step affects the term $(\lambda z.z)y$, modelling a destructive update of shared memory.

There are many desirable operational properties that an ideal calculus with explicit substitutions should meet. A crucial property is *simulation of β-reduction*: if a term t reduces to s in the λ-calculus, then t should also reduce to s in the calculus with explicit substitutions in question. For example, the β-reduction step $(\lambda x.\lambda y.x)z \to \lambda y.z$ is simulated by the following three reduction steps in λx:

$$(\lambda x.\lambda y.x)z \to (\lambda y.x)[x\backslash z] \to \lambda y.x[x\backslash z] \to \lambda y.z$$

A closely related property is known as *full composition*: a term built using the explicit substitution operator $t[x\backslash s]$ should reduce to the actual substitution $t\{x := s\}$. For example, in the calculus λx, the term $(xx)[x\backslash y]$ reduces to yy in three steps:

$$(xx)[x\backslash y] \to x[x\backslash y]\, x[x\backslash y] \to y\, x[x\backslash y] \to yy$$

The full composition property is subtler than it seems at first sight, since the term t may itself have other occurrences of the explicit substitution operator. For example, λx does not enjoy full composition—it is easy to check that $z[z\backslash x][x\backslash y]$ does not reduce to $z[z\backslash y]$. This suggests that the following rewrite rule should be added to have the full composition property:

$$(\texttt{sub}) \qquad t[x\backslash s][y\backslash u] \;\to\; t[y\backslash u][x\backslash s[y\backslash u]] \quad \text{if } x \notin \mathsf{fv}(u)$$

but unfortunately this rule leads to non-terminating behavior, since the right-hand side of the rule is an instance of the left-hand side:

$$\begin{aligned} t[x\backslash s][y\backslash u] &\to t[y\backslash u][x\backslash s[y\backslash u]] \\ &\to t[x\backslash s[y\backslash u]][y\backslash u[x\backslash s[y\backslash u]]] \\ &\to \ldots \end{aligned}$$

In fact, there is another important operational property that calculi with explicit substitutions should ideally enjoy, known as *preservation of strong normalization* (PSN). Recall that a term t is said to be *strongly normalizing* if there are no infinite reduction sequences $t \to t_1 \to t_2 \to \ldots$. A calculus with explicit substitutions is said to enjoy PSN if whenever t is strongly normalizing in the λ-calculus then t is also strongly normalizing in the given calculus with explicit substitutions. Around 1995, the question of whether the $\lambda\sigma$ calculus enjoyed PSN was open, and the community was hoping for a positive answer, when Paul-André Melliès famously exhibited a counterexample [117].

Most of the research on the field of explicit substitutions during the late 1990s and 2000s was concerned with finding a calculus with explicit substitutions verifying a number of desired good properties. Superficially, this means that the calculus should enjoy good operational properties, such as confluence, PSN, and full composition. More profoundly, this means that the operational behavior of the calculus should be backed up by an appropriate semantical justification.

As an answer to this quest, Delia Kesner and Stéphane Lengrand proposed an explicit substitution calculus $\lambda\mathtt{lxr}$ with explicit operators for *weakening* and *contraction*, whose operational semantics is justified by a sound and complete correspondence with linear logic proof nets [88, 85]. This calculus enjoys good operational properties. These ideas led Delia Kesner and her collaborators to develop further explicit substitution calculi in close correspondence with linear logic proof nets, the *prismoid of resources* [89]—with Fabien Renaud—, which in turn lead to the Linear Substitution Calculus [9]—with Beniamino Accattoli.

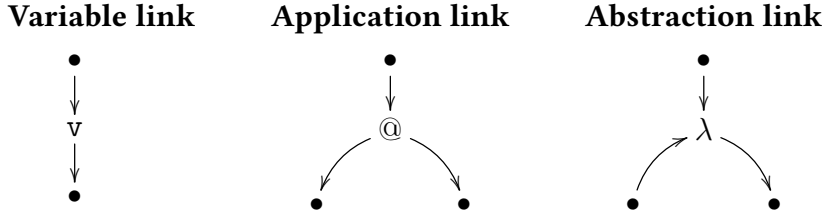
The Linear Substitution Calculus

The object of study of this thesis, the *Linear Substitution Calculus* (LSC), was introduced by Beniamino Accattoli and Delia Kesner in 2010 [9], inspired by previous calculi by Kesner et al. [88, 85, 89]. It also turns out to be similar to an earlier calculus by Robin Milner [120].

Why LSC?

- Its formulation is **simpler** than previous calculus of explicit substitutions, having only three rules.
- It is **semantically orthogonal** in the sense of residual theory [5]. Previous explicit substitution calculi do not have well-behaved residual theories.
- Its operational semantics can be justified via a translation into **linear logic** proof nets [2].

The starting point of the LSC is a representation of λ-calculus terms as *λ-graphs*. Roughly speaking, λ-graphs are λ-terms written using graph syntax. The syntax of λ-graphs is given by graphs that are built using *nodes* ($\bullet$) connected by three kinds of *links*: *variable links* (v), *application links* (@), and *abstraction links* (λ):



Variable occurrences in the λ-calculus are represented using variable links. The target of a

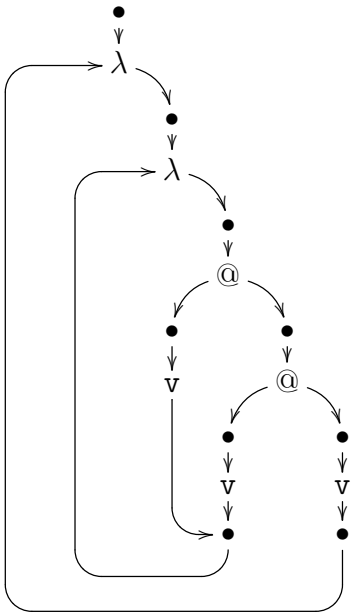
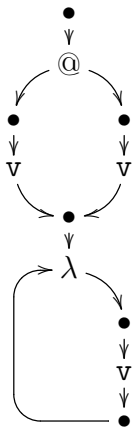


Figure 1.1: The λ-term $\lambda x.\lambda y.y(yx)$ represented as a λ-graph

variable link points to a node representing the current *binding* of the variable, *i.e.* its value. An application link corresponds to an application in the λ-calculus: the left target points to a node representing the function, and the right target points to a node representing the argument. An abstraction link corresponds to a lambda abstraction in the λ-calculus: the incoming arrow from the bottom left is connected to a node representing the name of the bound variable, while the target at the bottom right points to the body of the abstraction. For example, the λ-graph representation of the λ-term $\lambda x.\lambda y.y(yx)$ is shown in Figure 1.1.

An advantage of λ-graphs is that, much like explicit substitutions, they allow to easily represent shared subterms. For instance, the term $(\lambda x.x)(\lambda x.x)$ may be represented by the following λ-graph:



Compare this with a calculus with explicit substitutions, in which the term $(\lambda x.x)(\lambda x.x)$ may be rendered as $(yy)[y\backslash\lambda x.x]$.

In this thesis we are not interested in representing λ-terms directly using λ-graphs. We

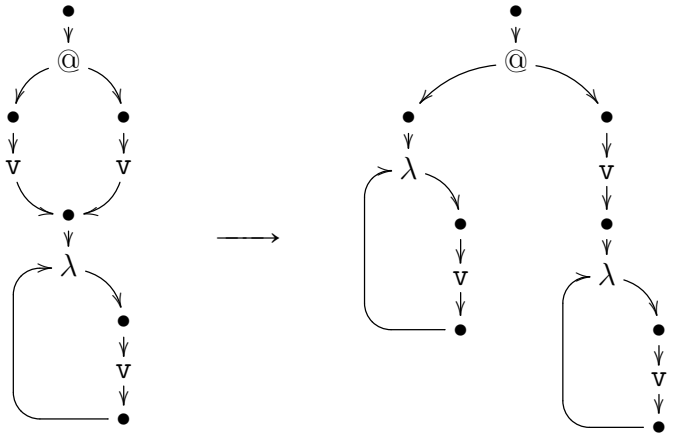


Figure 1.2: The portrayed rewrite step corresponds to a local interaction in the graph, but it is mapped to a non-local interaction when graphs are written back in term syntax: $(yy)[y\backslash\lambda x.x] \to ((\lambda x.x)y)[y\backslash\lambda x.x]$.

should warn the reader, however, that not every λ-graph is a valid λ-term: rather, λ-graphs must fulfill some *correctness conditions* to be considered valid. Moreover, depending on the exact representation chosen, other kinds of links besides variable, application, and abstraction may be needed—specifically, *weakening links* may be needed to represent an abstraction like $\lambda x.y$ in which the bound variable does not occur in the body. For the details, the interested reader should refer to Accattoli's PhD thesis [2].

The Linear Substitution Calculus results from the attempt at representing λ-graphs back in a more traditional term syntax, using an explicit substitution operator to allow the possibility of shared subterms. Terms of the LSC are thus variables x, y, z, ..., abstractions $\lambda x.t$, applications ts, and explicit substitutions $t[x\backslash s]$. However, LSC is not a typical calculus with explicit substitutions: there are two important traits that set LSC apart.

Distant Interaction. The first important difference between LSC and typical calculi with explicit substitutions is that rewriting rules in LSC operate *at a distance*. As already mentioned, terms in the LSC are intended to represent λ-graphs. Consequently, rewriting steps in the LSC are intended to model rewriting steps in a λ-graph, which correspond to *local interactions* in the graph. For example, Figure 1.2 depicts a rewrite step in a λ-graph, in which a variable link pointing to a subgraph A is replaced by a copy of A. When the same graph is rendered using term notation, the rewrite step becomes:

$$(yy)[y\backslash\lambda x.x] \to ((\lambda x.x)y)[y\backslash\lambda x.x]$$

Note that the affected occurrence of y and the explicit substitution $[y\backslash\lambda x.x]$ could, in principle, lie arbitrarily far away in the term. As a consequence, rewriting steps in the LSC may involve non-local interactions between distant parts of the term. The technical tool used by LSC to formally express rewriting rules at a distance is that of *contexts*. A context $\mathtt{C}$ is a term with exactly one free occurrence of a distinguished variable called a *hole*, and written $\Box$. If $\mathtt{C}$ is a context and t is a term, then $\mathtt{C}\langle t\rangle$ denotes the term that results from plugging the

term t into the hole of $\mathtt{C}$. For example, if $\mathtt{C} = (\Box y)[y\backslash\lambda x.x]$ then $\mathtt{C}\langle y\rangle = (yy)[y\backslash\lambda x.x]$ and $\mathtt{C}\langle\lambda x.x\rangle = ((\lambda x.x)y)[y\backslash\lambda x.x]$. Unlike the regular operation of substitution, plugging a term t into a context $\mathtt{C}$ may *capture* the free variables of t. For example, $(\lambda x.\Box)\langle x\rangle = \lambda x.x$. In LSC, sometimes we are interested in plugging a term into a context but *avoiding capture*. This operation is written $\mathtt{C}\langle\!\langle t\rangle\!\rangle$, and formally defined as $\mathtt{C}\langle\!\langle t\rangle\!\rangle \stackrel{\text{def}}{=} \mathtt{C}\{\Box := t\}$. For example, $(\lambda x.\Box)\langle\!\langle x\rangle\!\rangle = \lambda z.x$. A particular case of a context is one built exclusively from a list of zero or more explicit substitutions, that is, a context of the form $\Box[x_1\backslash t_1]\dots[x_n\backslash t_n]$. These are called *substitution contexts* and denoted by the letter $\mathtt{L}$. Given a substitution context $\mathtt{L}$ and a term t, we usually write $t\mathtt{L}$ to stand for $\mathtt{L}\langle t\rangle$.

We are now in condition to present the three rewriting rules of the LSC. Formally, the rewrite relation $(\to)$ is the least binary relation between terms that contains the three axioms below and which is closed by arbitrary contexts (*i.e.* $t \to s$ implies $\mathtt{C}\langle t\rangle \to \mathtt{C}\langle s\rangle$):

$$\begin{array}{lrcll}
(\mathtt{db}) & (\lambda x.t)\mathtt{L}\, s & \to & t[x\backslash s]\mathtt{L} & \\
(\mathtt{ls}) & \mathtt{C}\langle\!\langle x\rangle\!\rangle[x\backslash t] & \to & \mathtt{C}\langle\!\langle t\rangle\!\rangle[x\backslash t] & \\
(\mathtt{gc}) & t[x\backslash s] & \to & t & \text{if } x \notin \mathsf{fv}(t)
\end{array}$$

The first rewriting rule, called *distant beta* (db), corresponds to the β-reduction rule of the λ-calculus. It states that an interaction between a function $\lambda x.t$ and an argument s results in the creation of an explicit substitution operator $[x\backslash s]$ affecting the body of the function (t). The interaction is *distant* because in between the function $\lambda x.t$ and the argument s there may be an arbitrary number of explicit substitutions, represented by the substitution context $\mathtt{L}$. For instance, the following is a sequence of three db steps:

$$\begin{array}{rcl}
(\lambda x.\lambda y.\lambda z.x)\, t\, s\, u & \to & (\lambda y.\lambda z.x)[x\backslash t]\, s\, u \\
& \to & (\lambda z.x)[y\backslash s][x\backslash t]\, u \\
& \to & x[z\backslash u][y\backslash s][x\backslash t]
\end{array}$$

The second rewriting rule, called *linear substitution* (ls), states that any variable x bound by an explicit substitution to t may be replaced by a copy of t. The expression $\mathtt{C}\langle\!\langle x\rangle\!\rangle$ on the left-hand side of the ls rule represents a term with a (distinguished) free occurrence of the variable x. For instance, the following is a sequence of three ls steps:

$$\begin{array}{rcl}
(xx)[x\backslash yy][y\backslash z] & \to & (yyx)[x\backslash yy][y\backslash z] \\
& \to & (yyx)[x\backslash yz][y\backslash z] \\
& \to & (yy(yz))[x\backslash yz][y\backslash z]
\end{array}$$

The last rewriting rule, called *garbage collection* (gc), states that an explicit substitution $[x\backslash s]$ may be erased once the variable x is not referenced anywhere else in the term. The formal requirement is that the term be of the form $t[x\backslash s]$ and $x \notin \mathsf{fv}(t)$. Recall that $\mathsf{fv}(t)$ stands for the set of free variables of a term t. Also note that, in a calculus with explicit substitutions, $\mathsf{fv}(t[x\backslash s])$ is defined as $\mathsf{fv}(t) \cup (\mathsf{fv}(s)\backslash\{x\})$. For instance, the following is a sequence of three gc steps:

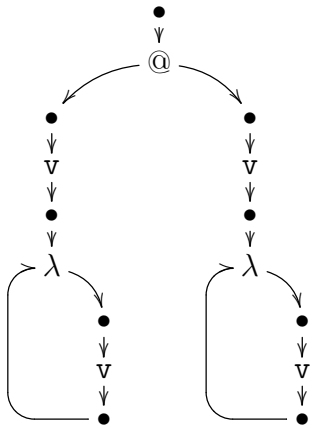
$$\begin{array}{rcl}
x[y\backslash z[w\backslash z]][z\backslash s] & \to & x[y\backslash z][z\backslash s] \\
& \to & x[z\backslash s] \\
& \to & x
\end{array}$$

When considered altogether, it is not difficult to show that the rules `db`, `ls`, and `gc` of the LSC simulate the β-reduction rule of the λ-calculus. For instance, the β-reduction step $(\lambda x.xx)\lambda y.y \to (\lambda y.y)\lambda y.y$ may be simulated by a `db` step, followed by two `ls` steps, plus a final `gc` step:

$$\begin{array}{rll} (\lambda x.xx)\lambda y.y & \to_{\texttt{db}} & (xx)[x\backslash\lambda y.y] \\ & \to_{\texttt{ls}} & ((\lambda y.y)x)[x\backslash\lambda y.y] \\ & \to_{\texttt{ls}} & ((\lambda y.y)\lambda y.y)[x\backslash\lambda y.y] \\ & \to_{\texttt{gc}} & (\lambda y.y)\lambda y.y \end{array}$$

As a matter of fact, the LSC enjoys all the desired properties for a calculus with explicit substitutions, including full composition and preservation of strong normalization [8, 10].

Graphical Equivalence. The second characteristic that sets the LSC apart from typical calculi with explicit substitutions is the presence of an equivalence relation of *graphical equivalence* between terms, written $t \sim s$. Graphical equivalence is intended to reflect *equality* of λ-graphs at the level of terms. The crucial point is that the rendering of a λ-graph as an LSC term is not a function—in some cases, a λ-graph may correspond to various different terms, depending on the order in which substitutions are written out. For instance, if we let $I = \lambda x.x$, the λ-graph below may be represented as any of the terms $(x[x\backslash I]y)[y\backslash I]$, $(xy)[x\backslash I][y\backslash I]$, or $(xy)[y\backslash I][x\backslash I]$:



Graphical equivalence is defined with the following three equations:

$$\begin{array}{rcll} (ts)[x\backslash u] & \sim & t[x\backslash u]s & \text{if } x \notin \mathsf{fv}(s) \\ (\lambda x.t)[y\backslash s] & \sim & \lambda x.t[y\backslash s] & \text{if } x \notin \mathsf{fv}(s) \text{ and } x \neq y \\ t[x\backslash s][y\backslash u] & \sim & t[y\backslash u][x\backslash s] & \text{if } x \notin \mathsf{fv}(u) \text{ and } y \notin \mathsf{fv}(s) \end{array}$$

Using these rules we have, for example:

$$(x[x\backslash I]y)[y\backslash I] \sim (xy)[x\backslash I][y\backslash I] \sim (xy)[y\backslash I][x\backslash I]$$

Observe that graphical equivalence does not identify $(ts)[x\backslash u]$ with $t[x\backslash u]s[x\backslash u]$, *i.e.* substitutions do not commute with applications in general. The intuitive reason is that one would like rewriting in LSC to be well-defined modulo graphical equivalence. A necessary condition for this is that graphical equivalence ($\sim$) should be a *strong bisimulation* with respect to

the rewriting relation ($\to$), that is, if $t' \sim t \to s$ then there should exist a term s' such that $t' \to s' \sim s$. If the terms $(ts)[x\backslash u]$ and $t[x\backslash u]s[x\backslash u]$ were identified, it would not be clear how to simulate a step $(ts)[x\backslash u] \to (ts)[x\backslash u']$ using a single step $t[x\backslash u]s[x\backslash u] \xrightarrow{?} t[x\backslash u']s[x\backslash u']$.

The deeper reason is that graphical equivalence intends to capture exactly those permutations of substitutions that are valid in λ-graphs. In fact, the LSC modulo graphical equivalence turns out to be isomorphic to the language of λ-graphs for the λ-calculus with sharing. The set of terms modulo graphical equivalence is in 1–1 correspondence with λ-graphs, and rewriting sequences in LSC can be transported functorially. Again, for the low-level details we refer the reader to Accattoli's PhD thesis [2].

1.2 This Work

This thesis is concerned with *evaluation strategies* in the *Linear Substitution Calculus*. In the following subsections we summarize our contributions and lay out the structure of this document. Generally speaking, the document is split into the main body and a technical appendix. Some proofs have been omitted from the main body; their details can be found in the technical appendix. In these cases the statement of the theorem includes the symbol ♣ with a reference to the appendix.

1.2.1 Background

In Chapter 2 (**Background**), we fix the notation and we recapitulate well-known definitions and theorems from rewriting theory and the λ-calculus that are relevant to our work. The experienced reader may want to skip this chapter.

1.2.2 Distilling Abstract Machines

Chapter 3 (**Distilling Abstract Machines**) is the result of joint work with Beniamino Accattoli and Damiano Mazza. In this chapter, we propose the Linear Substitution Calculus as an "abstract abstract machine".

To this aim, we study reduction strategies in the LSC and we show that they distill the essence of various abstract machines. To do this we formally define the notion of *distillery*. Roughly speaking, a reduction strategy in the LSC distills an abstract machine if:

- Each state S of the abstract machine can be *decoded* to a term $[\![S]\!]$ of the LSC.
- There is a binary relation $(\equiv)$ of structural equivalence between terms, which is a strong bisimulation.
- Transitions of the abstract machine can be classified in two types: *search transitions*, which change the focus of evaluation but are otherwise computationally irrelevant, and *principal transitions*, which perform the actual computation, in such a way that:
  - If $S \rightsquigarrow S'$ is a search transition, then $[\![S]\!] \equiv [\![S']\!]$.

– If $S \rightsquigarrow S'$ is a principal transition, then $[\![S]\!] \to\!\!\equiv [\![S']\!]$.

We then show that various reduction strategies in the LSC distill various (variations of) well-known abstract machines:

Reduction strategy	**Abstract machine**
call-by-name	Krivine abstract machine [97]
left-to-right call-by-value	CEK machine [57]
right-to-left call-by-value	ZINC machine [106],
call-by-need	Sestoft's machine [129],
strong call-by-name	Crégut's machine [42],

Moreover, we propose new abstract machines, suggested by the process of distillery, which are based on *flat global environments* rather than on *nested local environments*. In all of these cases, the process of distillation ensures that the abstract machine correctly implements the given reduction strategy.

Moreover, in each case, we show that simulating n reduction steps requires $O(c \cdot n)$ transitions of the machine, where c is a factor proportional to the size of the starting term. This justifies that the LSC—with any of the studied reduction strategies—is a reasonable model of computation, in the sense that execution can be simulated in a random-access machine with at most polynomial overhead in time.

1.2.3 Foundations of Strong Call-by-Need

Chapter 4 (**Foundations of Strong Call-by-Need**) is the result of joint work with Thibaut Balabonski, Eduardo Bonelli, and Delia Kesner. In this chapter, we turn our attention to an extension of the call-by-need strategy adapted for strong reduction.

The very definition of a *strong call-by-need* strategy is challenging. The crux of the matter is that call-by-need evaluation in the strong case is highly context-dependent. For example, in a term like $\lambda x.y[y\backslash xt]$ the strong call-by-need strategy should evaluate the term t:

$$\lambda x.y[y\backslash xt] \to \lambda x.y[y\backslash xt']$$

because reduction is strong and we seek to obtain the *full* normal form of the term. In contrast, in a term like $z[z\backslash\lambda x.y[y\backslash xt]]s$ the strong call-by-need strategy should perform the following linear substitution step:

$$z[z\backslash\lambda x.y[y\backslash xt]]s \to (\lambda x.y[y\backslash xt])[z\backslash\lambda x.y[y\backslash xt]]s$$

in order to stay faithful to its "by-need" nature. In this chapter:

- **Theory of Sharing.** We define a theory of strong reduction, the Theory of Sharing (Def. 4.4). The Theory of Sharing is a (non-deterministic) calculus whose rewriting rules induce an equational theory that characterizes the operational equivalence of programs with explicit substitutions, enforcing sharing.
- **Strong Call-by-Need Strategy.** We define a strategy for strong call-by-need-reduction (Def. 4.13), including various related notions such as normal forms and evaluation contexts. Strong call-by-need reduction is a deterministic strategy contained in the Theory of Sharing.

Its definition relies on the notion of *evaluation context*. Evaluation contexts are parameterized by a set ϑ of variables that are "frozen", *i.e.* symbolic, and by a binary flag indicating whether the evaluation context may be composed with an applicative context in such a way that the result is still an evaluation context.

- **Basic Properties of the Strong Call-by-Need Strategy.** We prove four basic principles that our strong call-by-need strategy enjoys, namely that the normal forms of the strategy are strong β-normal forms, up to unfolding, (Prop. 4.16), that the strategy is deterministic (Prop. 4.18), that it is a conservative extension of previously known notions of weak call-by-need (Thm. 4.23), and that it is correct with respect to β-reduction (Prop. 4.25), *i.e.* that if the strategy finds a normal form then the term has a strong β-normal form.

- **Completeness of the Strong Call-by-Need Strategy.** We study the *completeness* of our strong call-by-need strategy with respect to β-reduction, *i.e.* if a λ-term has a strong β-normal form, then our strong call-by-need strategy also reaches a normal form. We establish a precise relationship between the normal form in the λ-calculus and the normal form in our calculus with explicit substitutions (unfolding all of the explicit substitutions). The proof of normalization combines a logical argument and a syntactical argument, extending previous work by Kesner [87]. More specifically:

  - **Typability vs. Normalization.** We propose a non-idempotent intersection type system for the Theory of Sharing (Def. 4.27). This is a simple adaptation of existing systems, following the line of work proposed by Kesner [91]. We also show that typability in this system implies normalization in the Theory of Sharing. (Thm. 4.43).
  - **Completeness of the Theory.** We use the type system to argue that the Theory of Sharing is complete with respect to β-reduction (Prop. 4.45), *i.e.* that β-normalizing terms are also normalizing in the Theory of Sharing.
  - **Completeness of the Strategy.** Using an abstract factorization result by Accattoli [3], we argue that the strong call-by-need strategy is complete with respect to the Theory of Sharing (Prop. 4.54). The proof of this fact relies on an exhaustive case analysis of permutation diagrams.

1.2.4 Strong Call-by-Need for Pattern Matching and Fixed Points

Chapter 5 (**Strong Call-by-Need for Pattern Matching and Fixed Points**) is the result of joint work with Eduardo Bonelli and Kareem Mohamed. In this chapter, we extend the results of the previous chapter to incorporate pattern matching and recursion (terms are extended with constructors, a `case` construct, and a fixed point operator). Specifically:

- **Extended Theory of Sharing.** Our starting point is Grégoire and Leroy's extended λ-calculus (which we recall in Def. 5.3). We generalize the Theory of Sharing for the extended λ-calculus (Def. 5.7), and we provide a syntactic characterization of its normal forms (Def. 5.7).

- **Extended Type System.** We propose a non-idempotent intersection type system for the Extended Theory of Sharing. (Def. 5.10). We show that weakly normalizing terms are typable (Thm. 5.13) and that typable terms are weakly normalizing (Thm. 5.14). This requires defining a subtle property on typing judgments (Def. 5.12).

- **Extended Strong Call-by-Need Strategy.** We propose an extended strong call-by-need strategy for the Extended Theory of Sharing (Def. 5.17), and we show that the strategy enjoys good properties as in the previous chapter. Namely, the strategy is deterministic (Prop. 5.21), it conservatively extends the strong call-by-need strategy of the previous chapter (Prop. 5.21), and it is correct (Prop. 5.22) and complete (Thm. 5.23) with respect to reduction in the extended λ-calculus.

1.2.5 A Labeled Linear Substitution Calculus

Chapter 6 (**A Labeled Linear Substitution Calculus**) is the result of joint work with Eduardo Bonelli. In this chapter, we develop a variant of the LSC in which terms are decorated with labels, following the course set out by Lévy [110] when studying optimal reduction in the λ-calculus.

We go on by studying the metatheory of the labeled LSC, showing that it has most of the good properties that one would expect in a calculus with Lévy labels. More precisely:

- **A Labeled Linear Substitution Calculus.** We motivate some design decisions behind a calculus with Lévy labels, and we define a variant of the LSC with Lévy labels, the LLSC (Def. 6.6). Each reduction step in the labeled calculus has a *name*. We show some basic syntactical properties of LLSC.

- **Residuals and Orthogonality.** We show that the LLSC is an orthogonal axiomatic rewriting system (Prop. 6.32).

- **Weak Normalization of Bounded Reduction.** We prove that the LLSC is weakly normalizing if reduction is restricted to contracting steps whose names are labels of bounded height (Prop. 6.45).

- **Strong Normalization of Bounded Reduction (FFD).** We strengthen the aforementioned result, proving that the LLSC is strongly normalizing if reduction is restricted to contracting steps whose names are labels of bounded height (Thm. 6.51). This means the LSC enjoys a strong variant of the Finite Developments theorem, known as *Finite Family Developments* (FFD).

- **Confluence.** We provide two different proofs that the LLSC is confluent (Thm. 6.53).

1.2.6 Applications of the Labeled Linear Substitution Calculus

Chapter 7 (**Applications of the Labeled Linear Substitution Calculus**) is a continuation of Chapter 6, and also the result of joint work with Eduardo Bonelli.

In this chapter, we apply the labeled LSC developed in the previous chapter to derive further results about the LSC (without labels). One key tool from the previous chapter is the Finite Family Developments theorem:

- **Stability.** We show that the LSC without the gc rule enjoys Lévy's redex stability property (Prop. 7.1).
- **Deterministic Family Structure.** A Deterministic Family Structure (DFS) is an abstract rewriting system that verifies a set of particular axioms. We show that the LSC without gc forms a DFS (Thm. 7.13).
- **Optimal reduction.** We obtain an optimal reduction result for the LSC, as an immediate consequence of the fact that the LSC without gc is a DFS, using work of Glauert and Khasidashvili (which we review in Thm. 7.24).
- **Standardization.** Standardization, generally speaking, refers to a mechanism that converts a reduction sequence into standard form, in such a way that two reduction sequences are permutation equivalent if and only if they have the same standard form.

  We propose a standardization procedure for Deterministic Family Structures (Prop. 7.39), inspired on a standardization result by Klop. As a corollary, we obtain a standardization result for the LSC without gc (Coro. 7.43).
- **Normalization.** We prove a normalization result for Deterministic Family Structures (Prop. 7.54), giving sufficient conditions under which a reduction strategy is normalizing. As a corollary, we conclude that, in the LSC without gc the call-by-name strategy (Coro. 7.56) and a variant of the call-by-need strategy (Coro. 7.59) are both normalizing.

1.2.7 Publications and Work Not Included in This Thesis

The following publications correspond to results described in this thesis:

- B. Accattoli, P. Barenbaum, D. Mazza. **Distilling Abstract Machines.** *Proceedings of the International Conference on Functional Programming* (ICFP), ACM SIGPLAN Notices 49(9):363–376, 2014.
- B. Accattoli, P. Barenbaum, D. Mazza. **A Strong Distillery.** *Asian Symposium on Programming Languages and Systems* (APLAS), LNCS 9458:1–20, 2015.
- P. Barenbaum, E. Bonelli. **Optimality and the Linear Substitution Calculus.** *Formal Structures for Computation and Deduction* (FSCD), 9:1–9:16, 2017.
- T. Balabonski, P. Barenbaum, E. Bonelli, D. Kesner. **Foundations of Strong Call by Need.** *Proceedings of the International Conference on Functional Programming* (ICFP), ACM SIGPLAN Notices 20:1–20:29, 2017.

- P. Barenbaum, E. Bonelli, K. Mohamed. **Pattern Matching and Fixed Points: Resource Types and Strong Call-By-Need.** *Principles and Practice of Declarative Programming* (PPDP), 6:1–6:12, 2018.

There is another work in which I was involved during my PhD that is not described in detail in this manuscript. Jointly with Gonzalo Ciruelos, we used a confluent calculus based on a non-idempotent intersection type system to study derivation spaces in the pure (untyped) λ-calculus. This was the topic of Gonzalo's Master Thesis and also resulted in a publication:

- P. Barenbaum, G. Ciruelos. **Factoring Derivation Spaces via Intersection Types.** *Asian Symposium on Programming Languages and Systems* (APLAS), 24–44, 2018.

Chapter 2

Background

In this chapter we give an overview of some of the most important notions and results which our work builds upon. The presentation does not intend to be original nor exhaustive. The intention is rather to provide basic reference material, sketching a few well-known but hopefully interesting proofs, and pointing to references when appropriate.

2.1 Abstract Rewriting

Mathematical objects can be written in many different ways. A *term* or *expression* is a finite object, usually a string or a tree, intended to *represent* or *denote* a value. For example, in a multiplicative group, the expressions "$x \cdot x^{-1}$" and "1" are expected to denote the same mathematical object: they have different syntax but the same semantics.

Rewriting arises from the need to *decide* the equivalence of expressions, that is, to bridge the gap between syntax and semantics by providing a mechanical procedure that determines whether two expressions represent the same value. In rewriting theory one frequently starts by formulating an *equational theory*, that is, a set of equations that characterize the semantic equivalence of syntactic expressions. For example, the equational theory E defined below is composed of seven equation schemas, which characterize the equivalence of expressions representing elements of a free multiplicative group G:

$$E : \begin{cases} x \cdot 1 \equiv x & \forall x \in G \\ 1 \cdot x \equiv x & \forall x \in G \\ x \cdot (y \cdot z) \equiv (x \cdot y) \cdot z & \forall x, y, z \in G \\ x \cdot x^{-1} \equiv 1 & \forall x \in G \\ 1^{-1} \equiv 1 & \forall x, y \in G \\ (x \cdot y)^{-1} \equiv y^{-1} \cdot x^{-1} & \forall x, y \in G \\ (x^{-1})^{-1} \equiv x & \forall x \in G \end{cases}$$

An equational theory provides us with a way to *prove* that two expressions are equivalent. For example, one may justify that $x^{-1} \cdot x$ and 1 are equivalent expressions in E with the chain

of equalities:

$$\begin{aligned} x^{-1} \cdot x &\equiv ((x^{-1} \cdot x)^{-1})^{-1} \\ &\equiv (x \cdot x^{-1})^{-1} \\ &\equiv 1^{-1} \\ &\equiv 1 \end{aligned}$$

However, this proof requires a bit of ingenuity. In general, there may not exist an algorithm that decides whether two arbitrary expressions are equivalent, in a given equational theory.

The central idea behind rewriting theory is that equations of the form $x \equiv y$ may be *oriented*, that is, turned into *rewriting rules* of the form $x \to y$. A rewriting rule not only expresses the fact that the expressions on the left-hand side and the right-hand side are equivalent, but also endow the theory with computational meaning. Informally, a rewriting rule $x \to y$ means that any expression of the form given by x should be replaced by an expression of the form given by y. For example, the equational theory E may be oriented as follows, obtaining a *rewriting system* R:

$$R : \begin{cases} x \cdot 1 & \to & x & \forall x \in G \\ 1 \cdot x & \to & x & \forall x \in G \\ x \cdot (y \cdot z) & \to & (x \cdot y) \cdot z & \forall x, y, z \in G \\ x \cdot x^{-1} & \to & 1 & \forall x \in G \\ 1^{-1} & \to & 1 & \forall x, y \in G \\ (x \cdot y)^{-1} & \to & y^{-1} \cdot x^{-1} & \forall x, y \in G \\ (x^{-1})^{-1} & \to & x & \forall x \in G \end{cases}$$

Observe that in general there are exponentially many ways to orient an equational theory, since each equation $x \equiv y$ may be oriented as $x \to y$ or as $y \to x$. Now given any expression x representing an element of a free group, we may *rewrite* it by selecting some rule $x_i \to y_i$ in the rewriting system R and replacing a subexpression of the form x_i by a subexpression of the form y_i. Usually, rewriting is performed repeatedly, until there are no more rules to apply, and one arrives to a *normal form*.

For example, starting from the expression $x \cdot (y \cdot (y^{-1} \cdot x^{-1}))$ we may rewrite it as follows:

$$\begin{aligned} x \cdot (y \cdot (y^{-1} \cdot x^{-1})) &\to x \cdot ((y \cdot y^{-1}) \cdot x^{-1}) \\ &\to x \cdot (1 \cdot x^{-1}) \\ &\to x \cdot x^{-1} \\ &\to 1 \end{aligned}$$

A system of rewriting rules is said to be *terminating* if the procedure of repeatedly rewriting an expression $x_0 \to x_1 \to x_2 \to \ldots$ eventually terminates, arriving to a normal form. It can be shown that the rewriting system R given above is indeed terminating. On the other hand, a system of rewriting rules is said to have the *unique normal form property* if whenever x_1 and x_2 are equivalent expressions in the original equational theory such that x_1 and x_2 are normal forms, then $x_1 = x_2$. The rewriting system R above does *not* have the unique normal form property. For example, we have already proved that $x^{-1} \cdot x \equiv 1$ in the equational theory E, but they are normal forms, *i.e.* there are no rules in the system R that may be applied to rewrite the expressions $x^{-1} \cdot x$ and 1.

The foundational theorem of rewriting theory is a simple observation. Suppose that a system of rewrite rules R is terminating and it has the unique normal form property. Then the corresponding equational theory E may be decided as follows: to decide if $x_1 \equiv x_2$ holds in E, repeatedly apply rewriting rules $x_1 \to \ldots \to x_1'$ until obtaining a normal form x_1'. This procedure always arrives to a normal form because R is terminating. Similarly, repeatedly apply rewriting rules $x_2 \to \ldots \to x_2'$ until obtaining a normal form x_2'. Now since R has the unique normal form property, the equality $x_1' \equiv x_2'$ holds in E if and only if x_1' and x_2' are syntactically equal.

In the remainder of this section we give several definitions and some results, to make these ideas more precise and fix notation. We start by observing that there are two different, but compatible, views of a rewriting system that coexist in the literature, which we call the "propositional" and the "relevant" view.

Definition 2.1 (Propositional abstract rewriting system)**.** A *propositional abstract rewriting system* is a pair $(\mathsf{Obj}, \to)$ where Obj is a set whose elements are called *objects*, and $\to \subseteq A^2$ is a binary relation called the *rewriting relation*. Given two objects $x, y \in \mathsf{Obj}$ one writes $x \to y$ if $(x, y) \in \to$.

Definition 2.2 (Relevant abstract rewriting system)**.** A *relevant abstract rewriting system* is a 4-uple $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ where Obj is a set whose elements are called *objects*, Stp is a set whose elements are called *steps*, and $\mathsf{src}, \mathsf{tgt} : \mathsf{Stp} \to \mathsf{Obj}$ are functions indicating, respectively, the *source* and the *target* of each step. Given two objects $x, y \in \mathsf{Obj}$ and a step $R \in \mathsf{Stp}$, we write $x \xrightarrow{R}_{\mathcal{A}} y$ or $R : x \to_{\mathcal{A}} y$ if $\mathsf{src}(R) = x$ and $\mathsf{tgt}(R) = y$. Sometimes we drop the subscript and write $x \xrightarrow{R} y$ or $R : x \to y$ when $\mathcal{A}$ is clear from the context.

Remark 2.3. A relevant abstract rewriting system can always be regarded as a propositional abstract rewriting system by *propositional truncation*, by declaring the relation $x \to y$ to hold if and only if there exists a step $R \in \mathsf{Stp}$ such that $x \xrightarrow{R} y$.

Remark 2.4 (Steps vs. redexes). In relevant abstract rewriting systems that have *terms*, like the λ-calculus, a *redex* of a term t is any reducible subterm of t. More precisely, a redex is any subterm that is an instance of the left-hand side of some rewriting rule. For example the underlined subterm of the term $\lambda x.x\underline{((\lambda y.yy)z)}$ is a redex, because $(\lambda y.yy)z$ is an instance of the left-hand side of the β-reduction rule. Usually, there is an obvious bijection between the set of steps R starting from a term t and the set of redexes of t. In this situation, we may speak of steps and redexes interchangeably.

Throughout this thesis we speak of *abstract rewriting systems*, or *rewriting systems* for short, to refer to relevant rewriting systems. However, we liberally alternate between the *propositional* point of view, in which rewriting rules are defined as mere relations, and the *relevant* point of view, in which we care about the witness that justifies a rewriting step.

Definition 2.5 (Composition of rewriting relations)**.** From the relevant point of view, two arbitrary rewriting systems $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ and $\mathcal{B} = (\mathsf{Obj}, \mathsf{Stp}', \mathsf{src}', \mathsf{tgt}')$, can be

composed to obtain a rewriting system $(\mathcal{A}\cdot\mathcal{B}) = (\mathsf{Obj}, \mathsf{Stp}'', \mathsf{src}'', \mathsf{tgt}'')$ whose steps are defined by the following bijection $\mathsf{Stp} \times \mathsf{Stp}' \to \mathsf{Stp}''$:

$$(R : x \to_{\mathcal{A}} y, S : y \to_{\mathcal{B}} z) \quad \mapsto \quad R \cdot S : x \to_{(\mathcal{A}\cdot\mathcal{B})} z$$

From the propositional point of view, this corresponds to the *composition* of rewriting relations $\to_1$ and $\to_2$, defined as usual for binary relations:

$$x\ (\to_1 \circ \to_2)\ z \qquad \stackrel{\text{def}}{\iff} \qquad (\exists y.\ x \to_1 y\ \wedge\ y \to_2 z)$$

Definition 2.6 (Union of rewriting relations)**.** From the relevant point of view, two arbitrary rewriting systems $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ and $\mathcal{B} = (\mathsf{Obj}, \mathsf{Stp}', \mathsf{src}', \mathsf{tgt}')$, can be *added* to obtain a rewriting system $(\mathcal{A} \uplus \mathcal{B}) = (\mathsf{Obj}, \mathsf{Stp}'', \mathsf{src}'', \mathsf{tgt}'')$ whose steps are defined by the following bijection $\mathsf{Stp} \uplus \mathsf{Stp}' \to \mathsf{Stp}''$:

$$\begin{aligned} R : x \to_{\mathcal{A}} y \quad &\mapsto \quad R^{\mathsf{left}} : x \to_{\mathcal{A}\uplus\mathcal{B}} y \\ R : x \to_{\mathcal{B}} y \quad &\mapsto \quad R^{\mathsf{right}} : x \to_{\mathcal{A}\uplus\mathcal{B}} y \end{aligned}$$

From the propositional point of view, this corresponds to the *union* of rewriting relations $\to_1$ and $\to_2$, defined as usual for binary relations:

$$x(\to_1 \cup \to_2)y \qquad \stackrel{\text{def}}{\iff} \qquad x \to_1 y \vee x \to_2 y$$

Definition 2.7 (Inverse rewriting relation)**.** From the relevant point of view, a rewriting system $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ has an associated *opposite rewriting system* $\mathcal{A}^{\mathsf{op}} = (\mathsf{Obj}, \mathsf{Stp}', \mathsf{src}', \mathsf{tgt}')$ whose steps are defined by the following bijection $\mathsf{Stp} \to \mathsf{Stp}'$:

$$R : x \to_{\mathcal{A}} y \quad \mapsto \quad R^{-1} : y \to_{\mathcal{A}^{\mathsf{op}}} x$$

From the propositional point of view, this corresponds to the *inverse relation* of a rewriting relation $\to$, which is written $\to^{-1}$ or $\leftarrow$ and defined as follows:

$$x \to^{-1} y \qquad \stackrel{\text{def}}{\iff} \qquad y \to x \quad \text{for all } x, y \in \mathcal{A}$$

Definition 2.8 (Closure of a rewriting relation — propositional point of view)**.** Let $\mathcal{A}$ be a rewriting system, and let P be a predicate about binary relations on the set of objects Obj, *i.e.* given a binary relation $R \subseteq \mathsf{Obj} \times \mathsf{Obj}$ there is a proposition $P(R)$. From the propositional point of view, the P-closure of a relation R is the least relation R' such that $R \subseteq R'$ and such that $P(R')$ holds. Explicitly:

$$R' = \bigcap \{R'' \mid R \subseteq R'' \wedge P(R'')\}$$

In rewriting, there are various frequent cases of closures, for example:

1. The *transitive* closure of a rewriting relation $\to$ is written $\to^+$. It can be shown that $x \to^+ y$ if and only if $x \to \ldots \to y$ in *one or more* steps.

2. The *reflexive–transitive* closure of a rewriting relation $\to$ is written $\to^*$ or $\twoheadrightarrow$. It can be shown that $x \to^* y$ if and only if $x \to \ldots \to y$ in *zero or more* steps.

3. The *symmetric* closure of a rewriting relation $\to$ is written $\leftrightarrow$. It can be shown that $x \leftrightarrow y$ if and only if $x \to y$ or $y \to x$.

4. The *symmetric–reflexive–transitive* closure of a rewriting relation $\to$ is written $\leftrightarrow^*$. It can be shown that $x \leftrightarrow^* y$ if and only if $x \leftrightarrow \ldots \leftrightarrow y$ in zero or more steps.

5. In rewriting systems involving some notion of context, a rewriting relation $\to$ is *contextual* if $x \to y$ implies that $\mathtt{C}\langle x\rangle \to \mathtt{C}\langle y\rangle$ for any context $\mathtt{C}$. Recall that $\mathtt{C}\langle x\rangle$ represents the result of plugging the expression x inside the context $\mathtt{C}$. The *contextual* closure of $\to$ is sometimes written $\mathtt{C}\langle\to\rangle$. It can be shown that $x\ \mathtt{C}\langle\to\rangle\ y$ if and only if there exists a context $\mathtt{C}'$ and two objects x', y' such that:

$$x = \mathtt{C}'\langle x'\rangle \quad y = \mathtt{C}'\langle y'\rangle \quad x' \to y'$$

6. In rewriting systems involving some notion of context, a *congruence* is a binary relation which is simultaneously an equivalence relation (symmetric, reflexive, and transitive) and contextual. Sometimes we speak of the *congruence generated by a binary relation* R to mean the symmetric–reflexive–transitive–contextual closure of R.

Example 2.9 (Rewriting relations and closure)**.** *Let $\mathcal{A}$ be the rewriting system whose objects are sets of natural numbers and there is a step $X \to_{\mathcal{A}} Y$ if and only if $X = Y \cup \{n\}$ for some $n \in \mathbb{N}\backslash X$. Then:*

$$\begin{array}{rcll}
\{1,2,3\} & \to & \{1,3\} & \\
\{1,2,3\} & \to^{-1} & \{1,2,3,4\} & \\
\{1,2,3\} & \to^{+} & \{1\} & \\
X & \to & \varnothing & \textit{if and only if } X \textit{ is a singleton} \\
X & \to^* & \varnothing & \textit{if and only if } X \textit{ is finite} \\
X & \to^* & Y & \textit{if and only if } Y \subseteq X \textit{ and } X\backslash Y \textit{ is finite} \\
X & \leftrightarrow^* & Y & \textit{if and only if } X\backslash Y \textit{ and } Y\backslash X \textit{ are finite}
\end{array}$$

The various notions of closure of a rewriting relation can also be interpreted from a relevant point of view. For example:

1. A witness of a step in the transitive closure $S : x \to^+ y$ is a *non-empty* list of steps $S = [R_1, \ldots, R_n]$ where $x = x_0 \xrightarrow{R_1} x_1 \ldots \xrightarrow{R_n} x_n = y$.

2. A witness of a step in the reflexive–transitive closure $S : x \to^* y$ is a *possibly empty* list of steps $S = [R_1, \ldots, R_n]$ where $x = x_0 \xrightarrow{R_1} x_1 \ldots \xrightarrow{R_n} x_n = y$.

3. A witness of a step in the contextual closure $S : x\ \mathtt{C}\langle\to\rangle\ y$ is given by a pair $S = (\mathtt{C}', R)$ where $\mathtt{C}'$ is a context, $R : x' \to y'$ is a step, and we have that $x = \mathtt{C}'\langle x'\rangle$ and $y = \mathtt{C}'\langle y'\rangle$.

Except for Chapter 6, in which we work with residual theory, we usually take the issue of relevance lightly.

Definition 2.10 (Coinitial and cofinal steps)**.** Two steps $R : x_1 \to_{\mathcal{A}} y_1$ $S : x_2 \to_{\mathcal{A}} y_2$ are *coinitial* if $x_1 = x_2$ and *cofinal* if $y_1 = y_2$.

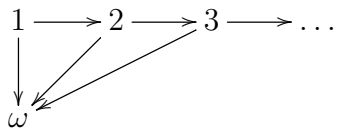
An important property that we are usually interested in, when studying a rewriting system, is that of (weak and strong) *normalization*. From the computational point of view, normalization ensures that a procedure defines a total function, that is, that the program does not "hang". From the logical point of view, normalization entails some forms of consistency.

Definition 2.11 (Normal forms, weak and strong normalization)**.** Let $\mathcal{A}$ be a rewriting system. Then:

1. An object x is a *normal form* if there is no step R in $\mathcal{A}$ such that $\mathsf{src}(R) = x$. We write $\mathsf{NF}(\mathcal{A})$ for the set of normal forms of $\mathcal{A}$.
2. An object x is *weakly normalizing* (WN) if there exists a normal form y such that $x \to^* y$.
3. An object x is *strongly normalizing* (SN) or *terminating* if there is no infinite sequence of steps $x = x_0 \to x_1 \to x_2 \to \ldots$.
4. The rewriting system $\mathcal{A}$ is called WN (resp. SN) if every object x in $\mathcal{A}$ is WN (resp. SN).

A strongly normalizing rewriting system is always weakly normalizing, but the converse does not hold.

Example 2.12 (Weak normalization without strong normalization)**.** *The rewriting system $\mathcal{A}$ whose objects are $\mathbb{N} \cup \{\omega\}$ and there are steps $n \to n+1$ and $n \to \omega$ for all $n \in \mathbb{N}$. Graphically:*



is weakly normalizing since for every $x \in \mathbb{N} \cup \{\omega\}$ we have $x \to^\star \omega$ which is a normal form. However, $\mathcal{A}$ is not strongly normalizing since $1 \to 2 \to 3 \to \ldots$ is an infinite sequence of steps.

Definition 2.13 (Finite branching)**.** Let $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ be a rewriting system. An object x is *finitely branching (from the relevant point of view)*, abbreviated FB, if the set $\{R \in \mathsf{Stp} \mid \mathsf{src}(R) = x\}$ is finite. A rewriting system is FB if every object is FB.

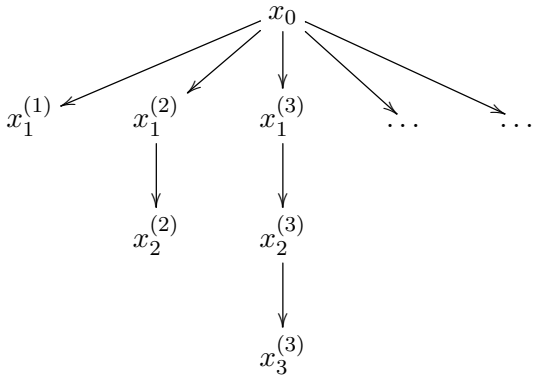
Remark 2.14. An object x is defined to be finitely branching from the *propositional* point of view, abbreviated $\mathrm{FB}_{\mathsf{prop}}$, if the set $\{y \in \mathsf{Obj} \mid x \to y\}$ is finite. It is easy to show that the implication FB $\implies$ $\mathrm{FB}_{\mathsf{prop}}$ holds in general. Moreover, in all the rewriting systems in this thesis, the set of steps $\{R \mid R : x \to y\}$ is always finite for any two fixed objects $x, y \in \mathsf{Obj}$. This means that throughout our work we may always assume that the converse implication $\mathrm{FB}_{\mathsf{prop}} \implies$ FB also holds, so we usually speak of a system being *finitely branching* without specifying in which sense.

In general, even if a rewriting system is strongly normalizing, there may not be a bound for the length of sequences of steps $x_1 \to x_2 \to \ldots \to x_n$. For instance:

Example 2.15 (Unbounded terminating rewriting system)**.** *Let $\mathcal{A}$ be the rewriting system whose objects are $\{x_0\} \cup \{x_i^{(n)} \mid n \in \mathbb{N}, 1 \leqslant i \leqslant n\}$, and there are steps:*

$$\begin{array}{rcll} x_0 & \to & x_1^{(n)} & \textit{for all } n \in \mathbb{N} \\ x_i^{(n)} & \to & x_{i+1}^{(n)} & \textit{for all } n \in \mathbb{N}, 1 \leqslant i \leqslant n-1 \end{array}$$

Graphically:



Then $\mathcal{A}$ is strongly normalizing but the length of a sequence of steps starting from x_0 is not bounded.

The following (non-constructive) result for finitely branching rewriting systems is known as König's lemma. It serves as a principle to justify that, in a system which is both strongly normalizing and finitely branching, inductive constructions are well defined:

Lemma 2.16 (König's Lemma)**.** *Let* $\mathsf{span}(x)$ *denote the set of objects* reachable *from x in a rewriting system $\mathcal{A}$:*

$$\mathsf{span}(x) \stackrel{\text{def}}{=} \{y \mid x \to^* y\}$$

If $\mathcal{A}$ is strongly normalizing and finitely branching, then $\mathsf{span}(x)$ *is finite for all x.*

Proof. We claim that if $\mathsf{span}(x)$ is infinite for some object x, then there exists an object x' such that $\mathsf{span}(x')$ is infinite and $x \to x'$. Indeed, since $\mathcal{A}$ is finitely branching, there is a finite set $Y = \{y_1, \ldots, y_n\}$ such that $x \to y$ if and only $y \in Y$. Then $\mathsf{span}(x) = \{x\} \cup \mathsf{span}(y_1) \cup \ldots \cup \mathsf{span}(y_n)$, so $\mathsf{span}(y_i)$ must be infinite for some y_i. It suffices to take $x' := y_i$ to finish the proof of the claim.

Now suppose that there is an object x_1 such that $\mathsf{span}(x_1)$ is infinite. By repeatedly applying the claim, we obtain an infinite sequence of steps $x_1 \to x_2 \to x_3 \to \ldots$ such that $\mathsf{span}(x_i)$ is infinite for all i. This contradicts that $\mathcal{A}$ is strongly normalizing. □

A consequence of König's lemma is that, in a finitely branching and strongly normalizing system, there is a bound for the length of sequences of steps going out from an object. We stress, however, that this does not provide a *constructive* bound.

Proposition 2.17 (Bound for strong normalization)**.** *Let $\mathcal{A}$ be a strongly normalizing and finitely branching rewriting system. Let x_0 be an object of $\mathcal{A}$. Then there exists a bound $M \in \mathbb{N}$ for the length of sequences of steps $x_0 \to x_1 \to \ldots \to x_n$ starting from x_0.*

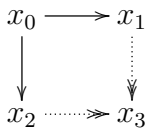
Proof. By König's Lemma (Lem. 2.16), the set $\mathsf{span}(x_0)$ is finite, so $M = \#\mathsf{span}(x_0)$ is a natural number. Let $x_0 \to x_1 \to \ldots \to x_n$ be any sequence starting on x_0. Note that the objects $x_0, x_1, \ldots, x_n$ are all different, for otherwise there is a loop $x_i \to \ldots \to x_i$ which contradicts the fact that $\mathcal{A}$ is strongly normalizing. Moreover, $\{x_0, x_1, \ldots, x_n\} \subseteq \mathsf{span}(x_0)$ since $x_0, x_1, \ldots, x_n$ are all reachable from x_0. Hence $n < n + 1 = \#\{x_0, \ldots, x_n\} \leqslant \#\mathsf{span}(x_0) = M$, as required. □

Definition 2.18 (Confluence)**.** A rewriting system $\mathcal{A}$ is said to be:

1. *Weakly Church–Rosser* (WCR) or *locally confluent* if given objects x_0, x_1, x_2 such that $x_0 \to x_1$ and $x_0 \to x_2$ there exists an object x_3 such that $x_1 \to^* x_3$ and $x_2 \to^* x_3$.

2. *Church–Rosser* (CR) or *confluent* if given objects x_0, x_1, x_2 such that $x_0 \to^* x_1$ and $x_0 \to^* x_2$ there exists an object x_3 such that $x_1 \to^* x_3$ and $x_2 \to^* x_3$.

A situation in which there are three objects and two steps $x_1 \leftarrow x_0 \to x_2$ is sometimes called a *peak*. When we complete a peak by constructing a fourth object and two sequences of steps as in $x_1 \to^* x_3 \leftarrow^* x_2$, we say that we *close the peak*. Peaks are drawn as squares which we occasionally call *tiles*. Following the standard convention in rewriting theory, steps that are universally quantified (given) are drawn with whole lines, whereas steps that are existentially quantified (proven) are often drawn with dotted lines. Graphically:



It is immediate to see that if a rewriting system is Church–Rosser, it is also weakly Church–Rosser. But the converse does not hold, as can be seen in this well-known example:

Example 2.19 (Non-confluent WCR system)**.** *Let $\mathcal{A}$ be the rewriting system:*

Then $\mathcal{A}$ is WCR *since the peak $1 \leftarrow 2 \to 3$ can be closed with $1 \leftarrow^* 3$, and similarly the peak $2 \leftarrow 3 \to 4$ can be closed with $2 \to^* 4$. But $\mathcal{A}$ is not* CR*, since the peak $1 \leftarrow 2 \to^* 4$ cannot be closed.*

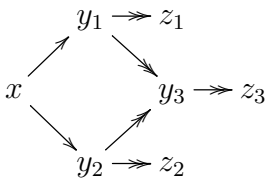
The following result is due to Max Newman and it is known in the literature as *Newman's lemma* or the *diamond lemma*. It is a useful tool to show that certain rewriting systems are confluent. Its importance lies in the fact that it allows to reduce the proof of *confluence*, which involves a universal quantifier over any peak of the form $y \twoheadleftarrow x \twoheadrightarrow z$ to the simpler property of *local confluence*, which only involves a universal quantifier over peaks of the form $y \leftarrow x \to z$. Local confluence can usually be checked by exhaustive case analysis on all possible peaks, while doing the same for confluence is usually impracticable.

Lemma 2.20 (Newman's lemma)**.** *If $\mathcal{A}$ is strongly normalizing and weakly Church–Rosser, then $\mathcal{A}$ is confluent.*

Proof. We say that an object x is *ambiguous* if it has two normal forms, *i.e.* $x \to^* x_1$ and $x \to^* x_2$ where $x_1 \neq x_2$ are different normal forms. We prove two claims.

- **Claim I:** If there are no ambiguous objects in $\mathcal{A}$, then $\mathcal{A}$ is CR.
  Proof of Claim I. Let $x_0 \to^* x_1$ and $x_0 \to^* x_2$. Since $\mathcal{A}$ is SN, let us normalize $x_1 \to^* x_1'$ until we obtain a normal form x_1', and similarly let us normalize $x_2 \to^* x_2'$ until we obtain a normal form x_2'. Since x_0 is not ambiguous, we have that $x_1' = x_2'$. This shows that $\mathcal{A}$ is CR, proving Claim I.

- **Claim II:** If x is ambiguous, there is an ambiguous object y such that $x \to y$.
  Proof of Claim II. Since x is ambiguous, let $x \to^* x_1$ and $x \to^* x_2$ where $x_1 \neq x_2$ are different normal forms. Note that $x \neq x_1$ and $x \neq x_2$, so $x \to^* x_1$ consists of at least one step, *i.e.* $x \to y_1 \to^* x_1$, and similarly $x \to y_2 \to^* x_2$. Since $\mathcal{A}$ is WN, we may close the peak $y_1 \leftarrow x \to y_2$ to obtain $y_1 \to^* y_3 \leftarrow^* y_2$ for some object y_3. Moreover, since $\mathcal{A}$ is SN, we may normalize $y_3 \to^* z_3$ until we obtain a normal form z_3. Graphically:

  Now z_1, z_2, and z_3 are normal forms and we know that $z_1 \neq z_2$ so either $z_3 \neq z_1$ or $z_3 \neq z_1$. If $z_3 \neq z_1$ then y_1 turns out to be ambiguous and it suffices to take $y := y_1$. If $z_3 \neq z_2$ then y_2 turns out to be ambiguous and it suffices to take $y := y_2$. This concludes the proof of Claim II.

It is now easy to prove Newman's lemma using the law of excluded middle. If $\mathcal{A}$ has no ambiguous objects, then $\mathcal{A}$ is CR by Claim I. If $\mathcal{A}$ has an ambiguous object x_1 then by repeatedly applying Claim II we construct a sequence of steps $x_1 \to x_2 \to x_3 \to \dots$ such that each x_i is ambiguous. This contradicts the fact that $\mathcal{A}$ is SN. □

The following result, due to Klop and Nederpelt, is a tool to show that a system is strongly normalizing. A different proof can be found in [135, Theorem 1.2.3 (iii)]:

Definition 2.21. A rewriting system $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ is *increasing* (Inc) if there is a function $f : \mathsf{Obj} \to \mathbb{N}$ such that $x \to y$ implies $f(x) < f(y)$ for all $x, y \in \mathsf{Obj}$.

Lemma 2.22 (Klop–Nederpelt)**.** *Let $\mathcal{A}$ be increasing, weakly Church–Rosser and weakly normalizing. Then $\mathcal{A}$ is strongly normalizing. In short:*

$$\text{Inc} \wedge \text{WCR} \wedge \text{WN} \implies \text{SN}$$

Proof. Let $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ be increasing, WCR and WN. Let $f : \mathsf{Obj} \to \mathbb{N}$ be the witness that $\mathcal{A}$ is increasing, *i.e.* if $x \to y$ then $f(x) < f(y)$. We prove the following claim:

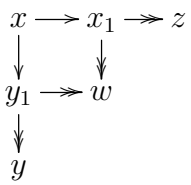
- **Claim:** Let $z \in \mathsf{Obj}$ be an object in normal form. If x and y are objects such that $x \twoheadrightarrow z$ and $x \twoheadrightarrow y$ then $y \twoheadrightarrow z$. Graphically:

Proof of the claim. In general, note that if $x_1 \twoheadrightarrow x_2$ then $f(x_1) \leqslant f(x_2)$ so $f(x_2) \leqslant f(x_1) \in \mathbb{N}_0$ is a natural number. Let z be a fixed normal form. Given a peak $y \twoheadleftarrow x \twoheadrightarrow z$ we define its weight as:

$$W(y \twoheadleftarrow x \twoheadrightarrow z) \stackrel{\text{def}}{=} f(z) - f(x)$$

The proof proceeds by complete induction on the weight of a peak.

1. **Base case, weight 0.** Then $f(z) - f(x) = 0$ so $x \twoheadrightarrow z$ consists of zero steps. This means that $x = z$, so x is in normal form. Since $x \twoheadrightarrow y$, we have that also $x = y$, and it is trivial to conclude.
2. **Induction, positive weight.** Then $x \twoheadrightarrow z$ consists of at least one step, that is, $x \to x_1 \twoheadrightarrow z$. We consider two subcases:
   - **If $x \twoheadrightarrow y$ consists of zero steps.** Then trivially $y = x \twoheadrightarrow z$.
   - **If $x \twoheadrightarrow y$ consists of at least one step.** Then $x \to y_1 \twoheadrightarrow y$. By hypothesis, $\mathcal{A}$ is WCR so we may close the peak $y_1 \leftarrow x \to x_1$ with an object w such that $y_1 \twoheadrightarrow w \twoheadleftarrow x_1$. The situation is as follows:



Note that we may apply the inductive hypothesis on the peak $w \twoheadleftarrow x_1 \twoheadrightarrow z$ since:

$$\begin{array}{rcll} W(w \twoheadleftarrow x_1 \twoheadrightarrow z) & = & f(z) - f(x_1) & \\ & < & f(z) - f(x) & \text{since } x \to x_1 \text{ so } f(x) < f(x_1) \\ & = & W(y \twoheadleftarrow x \twoheadrightarrow z) & \end{array}$$

So by *i.h.* we have that $w \twoheadrightarrow z$. Now observe that we may also apply the inductive hypothesis on the peak $y \twoheadleftarrow y_1 \twoheadrightarrow w \twoheadrightarrow z$, since:

$$\begin{array}{rcll} W(y \twoheadleftarrow y_1 \twoheadrightarrow w \twoheadrightarrow z) & = & f(z) - f(y_1) & \\ & < & f(z) - f(x) & \text{since } x \to y_1 \text{ so } f(x) < f(y_1) \\ & = & W(y \twoheadleftarrow x \twoheadrightarrow z) & \end{array}$$

So by *i.h.* we have that $y \twoheadrightarrow z$, which concludes the proof of the claim.

The proof of Klop–Nederpelt's lemma proceeds as follows: let x be any object. Since $\mathcal{A}$ is WN let $x \twoheadrightarrow z$ be a sequence of steps such that z is in normal form. By contradiction, suppose that $\mathcal{A}$ is not SN. That is, suppose that $x = x_0 \to x_1 \to x_2 \to \dots$ is an infinite sequence of steps. Note that $f(x_0) < f(x_1) < f(x_2) < \dots$ is a strictly increasing sequence of natural numbers. By the previous claim, we have that $x_n \twoheadrightarrow z$ for all $n \in \mathbb{N}$, so $f(x_n) < f(z)$ for all $n \in \mathbb{N}$. Thus $f(z)$ is an upper bound for the strictly increasing sequence $(f(x_n))_{n\in\mathbb{N}}$, which is a contradiction. □

Recall that a strict partial order $>$ on a set X is said to be *well-founded* if there are no infinite descending chains $x_1 > x_2 > x_3 > \dots$. The three following results are widely known and useful tools to show that a rewriting system is strongly normalizing:

Lemma 2.23 (Termination by interpretation)**.** *Let $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ be a rewriting system and let $>$ be a well-founded order on a set X. Suppose that there is a function $f : \mathsf{Obj} \to X$ such that $x \to y$ implies $f(x) > f(y)$. Then $\mathcal{A}$ is strongly normalizing.*

Proof. Suppose by contradiction that there is an infinite sequence $x_1 \to x_2 \to x_3 \to \dots$. Then $f(x_1) > f(x_2) > f(x_3) > \dots$, is an infinite descending chain. □

Lemma 2.24 (Lexicographic termination)**.** *Let $>_1$ and $>_2$ be strict partial orders on the sets X and Y respectively. Define the* lexicographic order $>$ *on the set $X \times Y$ as follows:*

$$(x, y) > (x', y') \quad \textit{if} (x >_1 x') \vee (x = x' \wedge y >_2 y')$$

If $>_1$ and $>_2$ are well-founded then $>$ is well-founded.

Proof. It is routine to check that $>$ is a strict order. Suppose by contradiction that there is an infinite descending chain $(x_1, y_1) > (x_2, y_2) > (x_3, y_3) > \dots$. Since $>_1$ is well-founded, the first component must eventually stabilize, that is, there is an $n \geqslant 1$ such that $x_n = x_m$ for all $m \geqslant n$. Hence $y_n >_2 y_{n+1} >_2 y_{n+2} > \dots$ is an infinite descending chain. □

Remark 2.25. Lem. 2.24 may be generalized for n-uples writing $X_1 \times X_2 \times \dots \times X_{n-1} \times X_n$ as $X_1 \times (X_2 \times \dots (X_{n-1} \times X_n))$.

Definition 2.26 (Finite multisets)**.** A *finite multiset* over a set X is, formally, a function $\mathfrak{m} : X \to \mathbb{N} \cup \{0\}$ such that $\mathfrak{m}(x)$ is non-zero for a finite number of elements $x \in X$. We write $[x_1, \dots, x_n]$ for the multiset $\mathfrak{m}$ such that $\mathfrak{m}(x)$ counts the number of occurrences of x in the sequence $x_1, \dots, x_n$. Sometimes we may write $\{x_1, \dots, x_n\}$ if it is clear from the context that we are working with multisets. We say that $\mathfrak{m} \subseteq \mathfrak{n}$ holds if $\mathfrak{m}(x) \leqslant \mathfrak{n}(x)$ for all $x \in X$. The notation $\mathfrak{m} \uplus \mathfrak{n}$ denotes the (additive) *union* of multisets, *i.e.* the function such that $(\mathfrak{m} \uplus \mathfrak{n})(x) = \mathfrak{m}(x) + \mathfrak{n}(x)$. The notation $\mathfrak{m} \ominus \mathfrak{n}$ denotes the difference of multisets, *i.e.* the function such that $(\mathfrak{m} \ominus \mathfrak{n})(x) = \mathfrak{m}(x) \dot{-} \mathfrak{n}(x)$ where $x \dot{-} y \stackrel{\text{def}}{=} \max\{0, x - y\}$.

Definition 2.27 (Multiset order)**.** Let $>$ be a strict partial order on a set X. Define the *multiset order* $\succ$ on the set of finite multisets of X as the transitive closure of $\succ^1$, where:

$$\mathfrak{m} \succ \mathfrak{n} \quad \text{if} \quad \mathfrak{m} \neq \mathfrak{n} \wedge \ \forall x \in X, \\ (\mathfrak{n}(x) > \mathfrak{m}(x) \implies \exists y \in X, y > x \wedge \mathfrak{m}(y) > \mathfrak{n}(y))$$

Lemma 2.28 (Characterization of the multiset order)**.** *The relation* $\mathfrak{m} > \mathfrak{n}$ *holds if and only if there exist multisets* $\mathfrak{a}, \mathfrak{b}$ *such that* $\mathfrak{n} = (\mathfrak{m} \ominus \mathfrak{a}) \uplus \mathfrak{b}$*, where* $\mathfrak{a} \subseteq \mathfrak{m}$ *is a non-empty multiset, and for every* $x \in \mathfrak{b}$ *there is an element* $y \in \mathfrak{a}$ *such that* $y > x$*.*

Proof. ($\Rightarrow$) Take $\mathfrak{a} := \mathfrak{m} \ominus \mathfrak{n}$ and $\mathfrak{b} := \mathfrak{n} \ominus (\mathfrak{m} \ominus \mathfrak{a})$. It is straightforward to check that all the conditions hold. ($\Leftarrow$) Let $x_0 \in \mathfrak{a}$ be a maximal element of $\mathfrak{a}$. We have that $x_0 \notin \mathfrak{b}$, for otherwise there would be an element $y \in \mathfrak{a}$ such that $y > x_0$. This means that $\mathfrak{m}(x_0) > \mathfrak{n}(x_0)$, so $\mathfrak{m} \neq \mathfrak{n}$. Moreover, suppose that $\mathfrak{n}(x) > \mathfrak{m}(x)$ for some $x \in X$. Then $x \in \mathfrak{b}$, so there is an element $y \in \mathfrak{a}$ such that $y > x$. Let $y_0 \in \mathfrak{a}$ be a maximal element such that $y_0 > x$. As before, we have that $y_0 \notin \mathfrak{b}$, and this means that $\mathfrak{m}(y_0) > \mathfrak{n}(y_0)$, as required. □

Theorem 2.29 (Multiset termination)**.** *If* $>$ *is well-founded then* $\succ$ *is a well-founded strict order.*

Proof. We sketch a proof due to Nachum Dershowitz and Zohar Manna; see [17, Section 2.5] for more details. Let $>$ be well-founded and suppose that $\succ$ is not well-founded. The proof proceeds by constructing a tree whose nodes are elements of the extended set $X \cup \{\bot\}$. The invariant is that in the n-th step we build a tree some of whose leaves may be decorated with $\bot$ and the remaining leaves are in 1–1 correspondence with the elements of $\mathfrak{m}_n$, accounting for multiplicities. Moreover, each branch of the tree is a decreasing sequence in X.

Let $\mathfrak{m}_1 \succ \mathfrak{m}_2 \succ \ldots$ be an infinite decreasing sequence of multisets. In the first step, the tree starts with a root with one children per each element of $\mathfrak{m}_1$. In the $(n+1)$-th step, we have that $\mathfrak{m}_n \succ \mathfrak{m}_{n+1}$, so by Lem. 2.28 $\mathfrak{m}_{n+1} = (\mathfrak{m}_n \ominus \mathfrak{a}) \uplus \mathfrak{b}$, where $\mathfrak{a} \subseteq \mathfrak{m}_n$ is a non-empty multiset, and for every $x \in \mathfrak{b}$ there is an element $y \in \mathfrak{a}$ such that $y > x$. For each element $x \in \mathfrak{a}$, the node for x is extended with a child decorated with $\bot$. For each element $x \in \mathfrak{b}$, let $y \in \mathfrak{a}$ be the corresponding element such that $y > x$; the node for y is extended with a child decorated with x. The resulting tree is infinite, since each step adds at least one node, but it is finitely branching since all multisets are finite. By König's Lemma (Lem. 2.16), it must have an infinite branch, contradicting the well-foundedness of $>$. □

2.2 Residual Theory

Consider a confluent abstract rewriting system $\mathcal{A}$. From the *propositional* point of view, confluence can be summarized in the inclusion of binary relations $(\twoheadleftarrow \circ \twoheadrightarrow) \subseteq (\twoheadrightarrow \circ \twoheadleftarrow)$. It merely means that all peaks can be closed:

From the *relevant* point of view, confluence means that if $\rho : x \twoheadrightarrow y$ and $\sigma : x \twoheadrightarrow z$ are sequences of rewriting steps, there exists an object w and two sequences of rewrite steps $\sigma' : y \twoheadrightarrow w$ and $\rho' : z \twoheadrightarrow w$. Graphically:

In fact, if a rewriting system is *orthogonal*, one can give a constructive account of the sequences σ' and ρ', and it can be shown that the confluence diagram is universal, *i.e.* a pushout. In particular, the diagram can be closed in such a way that the sequences $\rho\sigma'$ and $\sigma\rho'$ are equivalent in a precise sense. This relevant view of orthogonal rewriting systems can be attributed to Jean-Jacques Lévy and Gérard Huet [78, 79]. An axiomatic generalization of this theory was developed by Paul-André Melliès [118]. This theory relies crucially on the notion of *residual*, informally introduced in Sec. 1.1.2. In this section, we recapitulate some definitions and results from axiomatic residual theory that we will use throughout this thesis. They are especially important for Chapter 6.

Definition 2.30 (Axiomatic rewriting system)**.** An *axiomatic rewriting system* is a rewriting system $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ provided with a ternary *residual relation* $- \langle - \rangle -$ between steps such that:

$$R_1 \langle S \rangle R_2 \quad \text{implies} \quad \mathsf{src}(R_1) = \mathsf{src}(S) \wedge \mathsf{src}(R_2) = \mathsf{tgt}(S) \quad \text{for all } R_1, R_2, S \in \mathsf{Stp}$$

As customary, sometimes we subscript operations with $\mathcal{A}$ when the ambient rewriting system is not clear from the context, *e.g.* we may write $\mathsf{src}_{\mathcal{A}}(R)$ or $R \langle S \rangle_{\mathcal{A}} T$.

Definition 2.31 (Residual theory concepts)**.** The following notions can be defined for any axiomatic rewriting system $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt}, - \langle - \rangle -)$:

1. A *derivation* is a sequence of composable steps $R_1 \dots R_n$. By composable we mean that $\mathsf{tgt}(R_i) = \mathsf{src}(R_{i+1})$ for all $i \in \{1, \dots, n-1\}$. The length of a derivation is written $|\rho|$. The notions of source and target are extended for derivations, so that $\mathsf{src}(R_1 \dots R_n) = \mathsf{src}(R_1)$ and $\mathsf{tgt}(R_1 \dots R_n) = \mathsf{tgt}(R_n)$. The empty derivation, when $n = 0$, is written ϵ. The set of all derivations is written Deriv.

   Strictly speaking, an empty derivation is annotated with an object, so that there is one empty derivation ϵ_x for each object $x \in \mathsf{Obj}$, such that $\mathsf{src}(\epsilon_x) = \mathsf{tgt}(\epsilon_x) = x$.[1] The *composition* of the derivations ρ and σ is written $\rho \cdot \sigma$ or just $\rho\,\sigma$, and it is defined whenever $\mathsf{tgt}(\rho) = \mathsf{src}(\sigma)$.

2. The residual relation is generalized when the step in the middle is a derivation. More precisely if $\mathsf{src}(R) = \mathsf{src}(\sigma)$ and $\mathsf{src}(R') = \mathsf{tgt}(\sigma)$ the ternary relation $R \langle \sigma \rangle R'$ is declared to hold if and only if there exist steps $R_0, \dots, R_n, S_1, \dots, S_n$ such that

   - $\sigma = S_1 \dots S_n$,
   - $R = R_0$,
   - $R' = R_n$, and
   - $R_i \langle S_{i+1} \rangle R_{i+1}$ holds for all $i \in \{0, \dots, n-1\}$.

   Remark that $R \langle \epsilon_{\mathsf{src}(R)} \rangle R$.

3. We write $\langle \sigma \rangle$ for the binary relation $\{(R, R') \mid R \langle \sigma \rangle R'\}$.

[1] In other words, derivations are morphisms in the free category generated by $\mathcal{A}$, seen as a directed graph.

4. If $R_1 \langle\sigma\rangle R_2$ holds, we say that R_2 is a *residual* of R_1 after σ, and R_1 is an *ancestor* of R_2 before σ. If R and σ are coinitial, we write R/σ for the set $\{R' \mid R \langle\sigma\rangle R'\}$ of residuals of R after σ.

5. If $R/\sigma = \varnothing$ we say that σ *erases* R.

6. If $\#(R/\sigma) > 1$ we say that σ *duplicates* R.

7. If $\mathsf{tgt}(\sigma) = \mathsf{src}(R)$ and there is no R_0 such that $R_0 \langle\sigma\rangle R$, we say that σ *creates* R.

8. An axiomatic rewriting system has the *autoerasure* (AE) property if $R/R = \varnothing$ for all $R \in \mathsf{Stp}$.

9. An axiomatic rewriting system has the *finite residuals* (FR) property if the set R/S is finite for all coinitial $R, S \in \mathsf{Stp}$.

10. An axiomatic rewriting system has the *unique ancestor* (UA) property if a step has at most one ancestor, *i.e.* if $R_1 \langle S\rangle R$ and $R_2 \langle S\rangle R$ then $R_1 = R_2$ for all $R_1, R_2, R, S \in \mathsf{Stp}$.

11. An axiomatic rewriting system has the *acyclicity* property if whenever $R \neq S$ and $R/S = \varnothing$ then $S/R \neq \varnothing$.

12. A *set of coinitial steps* is a set $\mathcal{M}$ of steps such that if $R, S \in \mathcal{M}$ then $\mathsf{src}(R) = \mathsf{src}(S)$. The empty set of coinitial steps is written $\varnothing$. Strictly speaking, an empty set of coinitial steps is annotated with an object, so that there is one empty set of coinitial steps $\varnothing_x$ for each object x. The source of a set of coinitial steps is well-defined: if $\mathcal{M}$ is a non-empty set of coinitial steps, then $\mathsf{src}(\mathcal{M}) = \mathsf{src}(R)$ for any $R \in \mathcal{M}$. If $\mathcal{M}$ is empty, then $\mathsf{src}(\varnothing_x) = x$. Below, we argue that the *target* of a set of coinitial steps is also well-defined, and in particular $\mathsf{tgt}(\varnothing_x) = x$.

13. If $\mathcal{M}$ is a set of coinitial steps and $\mathsf{src}(\mathcal{M}) = \mathsf{src}(\sigma)$, we write $\mathcal{M}/\sigma$ for the set of coinitial steps $\{R' \mid R \in \mathcal{M} \text{ and } R \langle\sigma\rangle R'\}$. Remark that $\varnothing_{\mathsf{src}(\sigma)}/\sigma = \varnothing_{\mathsf{tgt}(\sigma)}$.

Definition 2.32 (Development)**.** Let $\mathcal{M}$ be a set of coinitial steps in an axiomatic rewriting system $\mathcal{A}$. A *development* of $\mathcal{M}$ is a possibly infinite sequence $R_1 R_2 \dots R_n \dots$ such that $R_i \in \mathcal{M}/R_1 \dots R_{i-1}$ for all $i \in \{1, \dots, n\}$. A development is *complete* if it is maximal.

Definition 2.33 (Finite developments property)**.** An axiomatic rewriting system $\mathcal{A}$ has the *finite developments* property (FD) if given a finite set of coinitial steps $\mathcal{M}$, there are no infinite developments of $\mathcal{M}$.

Let $\mathcal{A}$ be an axiomatic rewriting system, and let $\mathcal{M}$ be a finite set of coinitial steps. Consider the rewriting system $\mathcal{D}_\mathcal{M}$ whose objects are developments ρ of $\mathcal{M}$ and there is a step $R : \rho \to_{\mathcal{D}_\mathcal{M}} \rho R$ if and only if ρR results from extending the development ρ with a step $R \in \mathcal{M}/\rho$. Observe that if $\mathcal{A}$ has the finite developments property then $\mathcal{D}_\mathcal{M}$ is strongly normalizing. Moreover, if $\mathcal{A}$ has the finite residuals property, then $\mathcal{D}_\mathcal{M}$ is finitely branching. By Prop. 2.17 this means that given a finite set of coinitial steps $\mathcal{M}$, there is a bound for the length of any development of $\mathcal{M}$. This motivates the following definition:

Definition 2.34 (Depth of a set of coinitial steps)**.** Let $\mathcal{A}$ be an axiomatic rewriting system with finite developments and finite residuals. Then the length of the longest development of a set of coinitial steps $\mathcal{M}$ is called the *depth of* $\mathcal{M}$.

Remark 2.35 (Decreasing depth). Let $\mathcal{A}$ be an axiomatic rewriting system with finite developments and finite residuals. Suppose that $\mathcal{M}$ is a set of coinitial steps and $R \in \mathcal{M}$. If ρ is a development of $\mathcal{M}/R$ then $R\rho$ is a development of $\mathcal{M}$. This means that the depth of $\mathcal{M}$ is strictly greater than the depth of $\mathcal{M}/R$. This property allows one to give arguments and constructions on sets of coinitial steps by induction on their depth.

Proposition 2.36 (Existence of complete developments)**.** *If an axiomatic rewriting system $\mathcal{A}$ verifies* FD *then any finite set of coinitial steps $\mathcal{M}$ has a complete development.*

Proof. Construct a development $R_1 \dots R_i \dots$ by taking some $R_i \in \mathcal{M}/R_1 \dots R_{i-1}$ until the set $\mathcal{M}/R_1 \dots R_{i-1}$ is empty. This process must terminate for otherwise we would have an infinite development, contradicting FD. □

Definition 2.37 (Permutation tile)**.** Let $R\sigma$ and $S\rho$ be two (non-empty) derivations in an axiomatic rewriting system $\mathcal{A}$. The pair $(R\sigma, S\rho)$ is called a *permutation tile* if all the following conditions hold:

1. $R\sigma$ and $S\rho$ are coinitial and cofinal,
2. ρ is a complete development of R/S, and σ is a complete development of S/R,
3. $\langle R\sigma\rangle = \langle S\rho\rangle$ are equal as binary relations.

Definition 2.38 (Semantic orthogonality property)**.** An axiomatic rewriting system $\mathcal{A}$ has the *semantic orthogonality* property (SO) if given two coinitial steps R, S, a complete development ρ of R/S, and a complete development σ of S/R, then the pair $(R\sigma, S\rho)$ is a permutation tile.

Definition 2.39 (Orthogonal axiomatic rewriting system)**.** An axiomatic rewriting system is *orthogonal* if it has autoerasure (AE), finite residuals (FR), finite developments (FD), and semantic orthogonality (SO).

In the following subsection we study abstract properties of orthogonal axiomatic rewriting systems.

2.2.1 Properties of Orthogonal Axiomatic Rewriting Systems

Throughout this subsection, we assume that we are working within an orthogonal axiomatic rewriting system.

Definition 2.40 (Permutation equivalence)**.** Two coinitial derivations ρ and σ are said to be *permutation equivalent*, if $\rho \equiv \sigma$ holds, where $\equiv$ is a binary relation obtained from the reflexive–symmetric–transitive closure of the following relation $\equiv^1$:

$$\tau_1 R\sigma\tau_2 \equiv^1 \tau_1 S\rho\tau_2 \quad \text{if } (R\sigma, S\rho) \text{ is a permutation tile}$$

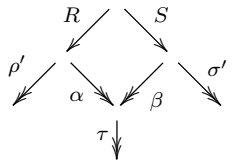
Lemma 2.41. *If $\rho \equiv \sigma$ then:*

- *For any τ_1, τ_2, we have $\tau_1\rho\tau_2 \equiv \tau_1\sigma\tau_2$.*
- *The derivations ρ and σ are coinitial and cofinal.*
- *The binary relations $\langle\rho\rangle$ and $\langle\sigma\rangle$ are equal.*

Proof. All the items are straightforward by induction on the derivation of $\rho \equiv \sigma$. □

Proposition 2.42 (Uniqueness of complete developments, modulo permutation equivalence)**.** *Let ρ, σ be complete developments of $\mathcal{M}$ in an orthogonal axiomatic rewriting system. Then $\rho \equiv \sigma$.*

Proof. By induction on the depth of $\mathcal{M}$. If $\mathcal{M}$ has depth 0, then $\mathcal{M} = \varnothing_x$, so ρ and σ are the empty derivation ϵ_x and $\rho \equiv \sigma$. If $\mathcal{M}$ has strictly positive depth then ρ and σ cannot be empty, for they would not be complete. So let $\rho = R\rho'$ and $\sigma = S\sigma'$. By the fact that complete developments exist (Prop. 2.36), let α be a complete development of S/R, and let β be a complete development of R/S. By semantic orthogonality, $(R\alpha, S\beta)$ is a permutation tile, so in particular $\mathcal{M}/R\alpha = \mathcal{M}/S\beta$. Consider a complete development τ of $\mathcal{M}/R\alpha$, which again exists by Prop. 2.36. The situation is:



Now observe that $\alpha\tau$ is a complete development of $\mathcal{M}/R$ so by *i.h.* (using Rem. 2.35). we have that $\rho' \equiv \alpha\tau$. Symmetrically, $\beta\tau$ is a complete development of $\mathcal{M}/S$ so $\beta\tau \equiv \sigma'$. Using Lem. 2.41 and the fact that $(R\alpha, S\beta)$ is a permutation tile, we conclude that $R\rho' \equiv R\beta\tau \equiv S\alpha\tau \equiv S\sigma'$, as required. □

The proposition above (Prop. 2.42) is the cornerstone of the axiomatic residual theory developed by Lévy, Huet and Melliès. Given any set of coinitial steps $\mathcal{M}$, we know that there is a complete development of $\mathcal{M}$. Moreover, if ρ, σ are two complete developments of $\mathcal{M}$ we know that they have the same source and the same target. In particular, the *target* of a set of coinitial steps, written $\mathsf{tgt}(\mathcal{M})$, may now be defined as $\mathsf{tgt}(\rho)$, and this does not depend on the choice of the complete development ρ. This means that $\mathcal{M}$ may be regarded as a *multistep* $\mathcal{M} : x \Rightarrow y$, where $x = \mathsf{src}(\mathcal{M})$ and $y = \mathsf{tgt}(\mathcal{M})$. Moreover, ρ and σ induce the same residual relation, *i.e.* $R/\rho = R/\sigma$, so the notation $R/\mathcal{M}$ may stand for R/ρ, and this is also well-defined.

Using the properties of existence and uniqueness of complete developments, the following definition shows that for any orthogonal axiomatic rewriting system $\mathcal{A}$, one may construct an orthogonal axiomatic rewriting system $\mathcal{A}^{\mathfrak{m}}$ whose steps are *multisteps* of $\mathcal{A}$.

Definition 2.43 (Multisteps and multiderivations)**.** Let $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ be an orthogonal axiomatic rewriting system. Then:

- A *multistep* $\mathcal{M}$ is a finite, non-empty, set of coinitial steps, that is, there is an object $x \in \mathsf{Obj}$ such that for every $R \in M$ we have $\mathsf{src}(R) = x$.

- If $\mathcal{M}$ is a multistep, we write $\mathsf{src}(\mathcal{M})$ for the source object x of the multistep, and $\mathsf{tgt}(\mathcal{M})$ for the target y of any complete development $\rho : x \twoheadrightarrow y$ of the set $\mathcal{M}$. Recall that there is always at least one complete development of a set M (Prop. 2.36), and that complete developments are unique modulo permutation equivalence (Prop. 2.42), so their targets always coincide by Lem. 2.41.

- Let $\mathsf{Multistep}$ the set of all multisteps starting on all possible objects $x \in \mathsf{Obj}$. Then the 4-uple $(\mathsf{Obj}, \mathsf{Multistep}, \mathsf{src}, \mathsf{tgt})$ is an abstract rewriting system, which we call the *abstract rewriting system of multisteps of* $\mathcal{A}$, and we denote by $\mathcal{A}^{\mathrm{m}}$.

- Sometimes we write $\mathcal{M} : x \Rightarrow y$ for a step $\mathcal{M} : x \to_{\mathcal{A}^{\mathrm{m}}} y$. To avoid confusion with derivations of $\mathcal{A}$, derivations of $\mathcal{A}^{\mathrm{m}}$ are sometimes called *multiderivations* of $\mathcal{A}$. When working with both derivations and multiderivations, we write $D, E, \ldots$ to range over multiderivations.

- We say that a derivation ρ is a *complete development* of a multiderivation $\mathcal{M}_1 \ldots \mathcal{M}_n$ if ρ is of the form $\rho_1 \ldots \rho_n$, where for each i, the derivation ρ_i is a complete development of the set $\mathcal{M}_i$.

Another consequence of the properties of existence and uniqueness of complete developments is the following abuse of notation, usually found in the literature, that we will frequently use.

Convention 2.44. *A multistep $\mathcal{M}$ (resp. multiderivation D) can be implicitly coerced to a derivation ρ, by taking ρ to be some complete development of $\mathcal{M}$ (resp. D). We assume that for each multistep $\mathcal{M}$ we deterministically choose a complete development $\partial\mathcal{M}$, which we call the* canonical *complete development of $\mathcal{M}$. Similarly for multiderivations, by setting $\partial(\mathcal{M}_1 \ldots \mathcal{M}_n) \stackrel{\text{def}}{=} \partial\mathcal{M}_1 \ldots \partial\mathcal{M}_n$.*

In the following lemma, we write $\mathcal{M} \sqcup \mathcal{N}$ for the derivation $\mathcal{M}(\mathcal{N}/\mathcal{M})$, where, as noted in Convention 2.44, "$\mathcal{M}$" stands for the canonical complete development of the set $\mathcal{M}$, and "$\mathcal{N}/\mathcal{M}$" stands for the canonical complete development of the set $\mathcal{N}/\mathcal{M} = \bigcup\{R/\mathcal{M} \mid R \in \mathcal{N}\}$.

Lemma 2.45 (Cube identity for multisteps)**.** *Let $\mathcal{M}$ and $\mathcal{N}$ denote sets of coinitial steps with the same source. Then $\mathcal{M} \sqcup \mathcal{N} \equiv \mathcal{N} \sqcup \mathcal{M}$.*

Proof. Let ρ be the canonical complete development of $\mathcal{M}$ and let σ be the canonical complete development of $\mathcal{N}/\mathcal{M}$. We claim that $\rho\sigma$ is a complete development of the set $\mathcal{M} \cup \mathcal{N}$, where $\cup$ is the set-theoretical union. Indeed:

- **Development.** Note that ρ is a development of $\mathcal{M} \subseteq \mathcal{M} \cup \mathcal{N}$ and σ is a development of $\mathcal{N}/\rho \subseteq (\mathcal{M} \cup \mathcal{N})/\rho$, so $\rho\sigma$ is a development of $\mathcal{M} \cup \mathcal{N}$.

- **Complete.** Suppose that $\rho\sigma$ is not maximal. Then there is a step $R \in (\mathcal{M} \cup \mathcal{N})/\rho\sigma$ that extends σ. But $(\mathcal{M} \cup \mathcal{N})/\rho\sigma = (\mathcal{M}/\rho\sigma) \cup (\mathcal{N}/\rho\sigma) = \mathcal{N}/\rho\sigma$ since ρ is a complete development of $\mathcal{M}$, which means that $\mathcal{M}/\rho\sigma = (\mathcal{M}/\rho)/\sigma = \varnothing/\sigma = \varnothing$. So we have that $R \in \mathcal{N}/\rho\sigma$ extends σ, contradicting that σ is a complete development of $\mathcal{N}/\rho$.

Hence the derivation $\mathcal{M} \sqcup \mathcal{N}$ is a complete development of $\mathcal{M} \cup \mathcal{N}$. Symmetrically, the derivation $\mathcal{N} \sqcup \mathcal{M}$ is also a complete development of $\mathcal{M} \cup \mathcal{N}$. By the uniqueness of complete developments (Prop. 2.42) we obtain that $\mathcal{M} \sqcup \mathcal{N} \equiv \mathcal{N} \sqcup \mathcal{M}$ as required. □

Definition 2.46 (Residual of a derivation after a set)**.** If ρ is a derivation and $\mathcal{M}$ is a set of coinitial steps with the same source as ρ, then $\rho/\mathcal{M}$ is a derivation defined as follows, by induction on the length of ρ:

$$\begin{array}{rcl} \epsilon_{\mathsf{src}(\mathcal{M})}/\mathcal{M} & \stackrel{\text{def}}{=} & \epsilon_{\mathsf{tgt}(\mathcal{M})} \\ R\rho/\mathcal{M} & \stackrel{\text{def}}{=} & (R/\mathcal{M})(\rho/(\mathcal{M}/R)) \end{array}$$

Note that $\mathcal{M}/R$ is a set of coinitial steps with the same source as ρ so the second equation typechecks. Note also that, "$R/\mathcal{M}$" stands for the canonical complete development of the multiset $R/\mathcal{M}$.

Lemma 2.47. *Let ρ be a complete development of $\mathcal{M}$. Then $\rho/\mathcal{N}$ is a complete development of $\mathcal{M}/\mathcal{N}$.*

Proof. By induction on the depth of $\mathcal{M}$. If the depth is 0, then $\mathcal{M}$ is empty and it is immediate to conclude. If the depth of $\mathcal{M}$ is positive, then $\mathcal{M}$ is non-empty and ρ is of the form $R\rho'$ where $R \in \mathcal{M}$ and ρ' is a complete development of $\mathcal{M}/R$. Recall that the depth of $\mathcal{M}/R$ is strictly smaller than the depth of $\mathcal{M}$, as observed in Rem. 2.35, so by inductive hypothesis $\rho'/(\mathcal{N}/R)$ is a complete development of $(\mathcal{M}/R)/(\mathcal{N}/R)$. Moreover, the cube identity (Lem. 2.45) ensures that

$$(\mathcal{M}/R)/(\mathcal{N}/R) = \mathcal{M}/(R \sqcup \mathcal{N}) = \mathcal{M}/(\mathcal{N} \sqcup R) = (\mathcal{M}/\mathcal{N})/(R/\mathcal{N})$$

To conclude, observe that $\rho/\mathcal{N} = R\rho'/\mathcal{N} = (R/\mathcal{N})(\rho'/(\mathcal{N}/R))$, where "$R/\mathcal{N}$" stands for the canonical complete development of the set $R/\mathcal{N}$, which is a subset of $\mathcal{M}/\mathcal{N}$, and $\rho'/(\mathcal{N}/R)$ is a complete development of the set $(\mathcal{M}/\mathcal{N})/(R/\mathcal{N})$. Hence $\rho/\mathcal{N}$ is a complete development of $\mathcal{M}/\mathcal{N}$, as required. □

Definition 2.48 (Residual of a derivation after a derivation)**.** If ρ and σ are coinitial derivations, ρ/σ is a derivation defined as follows, by induction on the length of ρ:

$$\begin{array}{rcl} \epsilon_{\mathsf{src}(\sigma)}/\sigma & \stackrel{\text{def}}{=} & \epsilon_{\mathsf{tgt}(\sigma)} \\ R\rho/\sigma & \stackrel{\text{def}}{=} & (R/\sigma)(\rho/(\sigma/\{R\})) \end{array}$$

Note that $\sigma/\{R\}$ is the residual of a derivation after a set of coinitial steps according to the previous definition (Def. 2.46). Note also that "R/σ" stands for the canonical complete development of the multiset R/σ.

Remark 2.49. By autoerasure, we have that $\rho/\rho = \epsilon$ for any derivation ρ. This can be formally proved by induction on ρ.

Lemma 2.50 (Properties of residuals)**.** *The following hold in any orthogonal axiomatic rewriting system:*

1. $\rho/\sigma\tau = (\rho/\sigma)/\tau$
2. $\rho\sigma/\tau = (\rho/\tau)(\sigma/(\tau/\rho))$

Proof. Getting the proof right is a bit delicate. Before doing so, we state and prove some slightly less general claims:

- **Claim I.** For all $\rho, \sigma, \mathcal{M}$ we have $\rho\sigma/\mathcal{M} = (\rho/\mathcal{M})(\sigma/(\mathcal{M}/\rho))$.
  Proof of Claim I. By induction on ρ. If ρ is empty it is immediate. If $\rho = R\rho'$ then:

$$\begin{array}{rcll} R\rho'\sigma/\mathcal{M} & = & (R/\mathcal{M})(\rho'\sigma/(\mathcal{M}/R)) & \text{by definition} \\ & = & (R/\mathcal{M})(\rho'/(\mathcal{M}/R))(\sigma/(\mathcal{M}/R\rho')) & \text{by } \textit{i.h.} \\ & = & (R\rho'/\mathcal{M})(\sigma/(\mathcal{M}/R\rho')) & \text{by definition} \end{array}$$

- **Claim II.** Let ρ be a complete development of a set $\mathcal{M}$. Then for any σ, we have that $\sigma/\rho = \sigma/\mathcal{M}$.
  Proof of Claim II. By induction on σ. If σ is empty, it is immediate. If $\sigma = S\sigma'$ observe that Lem. 2.47 ensures that ρ/S is a complete development of $\mathcal{M}/S$:

$$\begin{array}{rcll} S\sigma'/\rho & = & (S/\rho)(\sigma'/(\rho/S)) & \text{by definition} \\ & = & (S/\rho)(\sigma'/(\mathcal{M}/S)) & \text{by } \textit{i.h.} \text{ using Lem. 2.47} \\ & = & (S/\mathcal{M})(\sigma'/(\mathcal{M}/S)) & \text{since } \rho \text{ is a complete development of } \mathcal{M} \\ & = & S\sigma'/\mathcal{M} & \text{by Claim I} \end{array}$$

- **Claim III.** If ρ is a complete development of a set $\mathcal{M}$, then $\rho\sigma/\tau = (\rho/\tau)(\sigma/(\tau/\rho))$.
  Proof of Claim III. By induction on ρ. If ρ is empty, it is immediate. Otherwise $\rho = R\rho'$ is a complete development of $\mathcal{M}$ so ρ' is a complete development of $\mathcal{M}/R$. Then:

$$\begin{array}{rcll} R\rho'\sigma/\tau & = & (R/\tau)(\rho'\sigma/(\tau/R)) & \text{by definition} \\ & = & (R/\tau)(\rho'/(\tau/R))(\sigma/((\tau/R)/\rho')) & \text{by } \textit{i.h.} \\ & = & (R\rho'/\tau)(\sigma/((\tau/R)/\rho')) & \text{by definition} \\ & = & (R\rho'/\tau)(\sigma/((\tau/R)/(\mathcal{M}/R))) & \text{by Claim II} \\ & = & (R\rho'/\tau)(\sigma/(\tau/\mathcal{M})) & \text{by the cube identity (Lem. 2.45)} \\ & & & \text{since } R \sqcup \mathcal{M} \equiv \mathcal{M} \sqcup R = \mathcal{M} \\ & = & (R\rho'/\tau)(\sigma/(\tau/R\rho')) & \text{by Claim II} \end{array}$$

Having established these claims, we are able to prove items 1. and 2. in the statement:

1. Let us prove that $\rho/\sigma\tau = (\rho/\sigma)/\tau$. By induction on ρ. If ρ is empty, it is immediate. If $\rho = R\rho'$ then:

$$\begin{array}{rcll} R\rho'/\sigma\tau & = & (R/\sigma\tau)(\rho'/(\sigma\tau/R)) & \text{by definition} \\ & = & ((R/\sigma)/\tau)(\rho'/(\sigma\tau/R)) & \text{by definition} \\ & = & ((R/\sigma)/\tau)(\rho'/(\sigma/R)(\tau/(R/\sigma))) & \text{by Claim I} \\ & = & ((R/\sigma)/\tau)((\rho'/(\sigma/R))/(\tau/(R/\sigma))) & \text{by } \textit{i.h.} \\ & = & (R/\sigma)(\rho'/(\sigma/R))/\tau & \text{by Claim III} \\ & & & \text{since } R/\sigma \text{ is a complete development} \\ & = & (R\rho'/\sigma)/\tau & \text{by definition} \end{array}$$

2. Let us prove that $\rho\sigma/\tau = (\rho/\tau)(\sigma/(\tau/\rho))$. By induction on ρ. If ρ is empty, it is immediate. If $\rho = R\rho'$ then:

$$\begin{array}{rcll} R\rho'\sigma/\tau & = & (R/\tau)(\rho'\sigma/(\tau/R)) & \text{by definition} \\ & = & (R/\tau)(\rho'/(\tau/R))(\sigma/((\tau/R)/\rho')) & \text{by } \textit{i.h.} \\ & = & (R/\tau)(\rho'/(\tau/R))(\sigma/(\tau/R\rho')) & \text{by item 1.} \\ & = & (R\rho'/\tau)(\sigma/(\tau/R\rho')) & \text{by definition} \end{array}$$

□

Proposition 2.51 (Orthogonality of multisteps)**.** *Let $\mathcal{A}$ be an orthogonal axiomatic rewriting system. Then the abstract rewriting system of multisteps $\mathcal{A}^{\mathfrak{m}}$ is also an orthogonal axiomatic rewriting system.*

Proof. Given three multisteps:

$$\mathcal{M} : x \Rightarrow y \quad \mathcal{N} : x \Rightarrow x' \quad \mathcal{M}' : x' \Rightarrow y'$$

declare the residual relation $\mathcal{M} \left\langle \mathcal{N} \right\rangle_{\mathcal{A}^{\mathfrak{m}}} \mathcal{M}'$ to hold in $\mathcal{A}^{\mathfrak{m}}$ whenever the equality $\mathcal{M}' = \mathcal{M}/\mathcal{N}$ holds when $\mathcal{M}$, $\mathcal{N}$, and $\mathcal{M}'$ are seen as sets of coinitial steps in $\mathcal{A}$. Note that residuals are *affine*, that is, given coinitial multisteps $\mathcal{M}, \mathcal{N}$ the set of residuals $\{\mathcal{P} \mid \mathcal{M} \left\langle \mathcal{N} \right\rangle_{\mathcal{A}^{\mathfrak{m}}} \mathcal{P}\}$ is either empty or consists of exactly one multistep $\mathcal{M}/\mathcal{N}$.

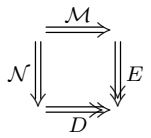
Let us check that this definition verifies the axioms of an orthogonal axiomatic rewriting system:

1. *Autoerasure.* Note that $\mathcal{M}/\mathcal{M} = \varnothing$ in $\mathcal{A}$. Since a multistep is defined to be a non-empty set of coinitial steps, there is no multistep $\mathcal{N}$ such that $\mathcal{M} \left\langle \mathcal{M} \right\rangle_{\mathcal{A}^{\mathfrak{m}}} \mathcal{N}$.

2. *Finite residuals.* Even more strongly, residuals are affine.

3. *Finite Developments.* By the fact that residuals are affine, the length of any development of a set $X = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ is bounded by n.

4. *Semantic Orthogonality.* Let $\mathcal{M}, \mathcal{N}$ be coinitial multisteps. Then the peak may be closed as follows:



   If the set $\mathcal{M}/\mathcal{N}$ is empty, the multiderivation D is chosen to be the empty multiderivation ϵ. If the set $\mathcal{M}/\mathcal{N}$ is non-empty, the multiderivation D is chosen to be the multiderivation $\mathcal{M}/\mathcal{N}$ of length 1. Symmetrically for E.

   One can then check that the pair $(\mathcal{M}E, \mathcal{N}D)$ is a permutation tile, which is an immediate consequence of the cube identity for multisteps (Lem. 2.45).

□

Lemma 2.52. *In an orthogonal axiomatic rewriting system $\mathcal{A}$, let $\mathcal{M}$ be a multistep and let $R_1 \dots R_n$ be a complete development of $\mathcal{M}$. Then in the rewriting system of multisteps we have that $\mathcal{M} \equiv_{\mathcal{A}^{\mathfrak{m}}} \{R_1\} \dots \{R_n\}$.*

Proof. By induction on the depth of $\mathcal{M}$. Recall that $\mathcal{M}$ is a multistep so it is non-empty. We consider two cases:

1. *If $n = 1$.* Then ϵ is a complete development of $\mathcal{M}/R_1$, so $\mathcal{M}/R_1 = \varnothing$. Moreover, $R_1 \in \mathcal{M}$, so $\{R_1\}/\mathcal{M} = \varnothing$. This means that $(\mathcal{M}, \{R_1\})$ is a permutation tile in $\mathcal{A}^{\mathfrak{m}}$ and we have $\mathcal{M} \equiv_{\mathcal{A}^{\mathfrak{m}}} \{R_1\}$.

2. *If $n > 1$.* Then $R_2 \dots R_n$ is a complete development of $\mathcal{M}/R_1$, so $\mathcal{M}/R_1$ is non-empty. Moreover, $R_1 \in \mathcal{M}$, so $\{R_1\}/\mathcal{M} = \varnothing$. This means that $(\mathcal{M}, \{R_1\}(\mathcal{M}/R_1))$ is a permutation tile in $\mathcal{A}^{\mathfrak{m}}$, so $\mathcal{M} \equiv_{\mathcal{A}^{\mathfrak{m}}} \{R_1\}(\mathcal{M}/R_1) \equiv_{\mathcal{A}^{\mathfrak{m}}} \{R_1\}\{R_2\} \dots \{R_n\}$ by *i.h.*, relying on Rem. 2.35.

□

Proposition 2.53. *If $\mathcal{A}$ is an orthogonal axiomatic rewriting system, the mappings:*

$$\begin{array}{rlcl} \mathtt{i}: & \mathsf{Deriv}_{\mathcal{A}} & \to & \mathsf{Deriv}_{\mathcal{A}^{\mathfrak{m}}} \\ & R_1 \dots R_n & \mapsto & \{R_1\} \dots \{R_n\} \end{array} \qquad \begin{array}{rlcl} \partial: & \mathsf{Deriv}_{\mathcal{A}^{\mathfrak{m}}} & \to & \mathsf{Deriv}_{\mathcal{A}} \\ & \mathcal{M}_1 \dots \mathcal{M}_n & \mapsto & \partial(\mathcal{M}_1) \dots \partial(\mathcal{M}_n) \end{array}$$

Induce a bijection $(\mathsf{Deriv}_{\mathcal{A}}/\equiv_{\mathcal{A}}) \simeq (\mathsf{Deriv}_{\mathcal{A}^{\mathfrak{m}}}/\equiv_{\mathcal{A}^{\mathfrak{m}}})$. Recall that $\partial\mathcal{M}$ denotes the canonical complete development of $\mathcal{M}$,

Proof. First we prove that $\mathtt{i}$ and ∂ are well-defined over permutation-equivalence classes.

($\to$) We claim that if $\rho \equiv_{\mathcal{A}} \sigma$ then $\mathtt{i}(\rho) \equiv_{\mathcal{A}^{\mathfrak{m}}} \mathtt{i}(\sigma)$. Indeed, by induction on the derivation of $\rho \equiv \sigma$, the interesting case is one-step permutation, *i.e.* when $\rho = \tau_1 A\beta\tau_2$ and $\sigma = \tau_1 B\alpha\tau_2$ where $(A\beta, B\alpha)$ is a permutation tile in $\mathcal{A}$, *i.e.* α is a complete development of A/B and β is a complete development of B/A. Then by Lem. 2.52 we have $\mathtt{i}(\rho) = \mathtt{i}(\tau_1)f(A)\mathtt{i}(\beta)\mathtt{i}(\tau_2) \equiv_{\mathcal{A}^{\mathfrak{m}}} \mathtt{i}(\tau_1)\mathtt{i}(B)\mathtt{i}(\alpha)\mathtt{i}(\tau_2) = \mathtt{i}(\sigma)$. This mapping defines a function $\mathtt{i} : \mathsf{Deriv}_{\mathcal{A}}/\equiv_{\mathcal{A}} \to \mathsf{Deriv}_{\mathcal{A}^{\mathfrak{m}}}/\equiv_{\mathcal{A}^{\mathfrak{m}}}$ also noted $\mathtt{i}$.

($\leftarrow$) We claim that if $D \equiv_{\mathcal{A}^{\mathfrak{m}}} E$ then $\partial(D) \equiv_{\mathcal{A}} \partial(E)$. Indeed, by induction on the derivation of $D \equiv E$, the interesting case is one-step permutation, *i.e.* when $D = F_1\mathcal{M}VF_2$ and $E = F_1\mathcal{N}UF_2$ where $(\mathcal{M}V, \mathcal{N}U)$ is a permutation tile in $\mathcal{A}^{\mathfrak{m}}$, *i.e.* U is a complete development of $\mathcal{M}/\mathcal{N}$ and V is a complete development of $\mathcal{N}/\mathcal{M}$. Then by the cube identity for multisteps (Lem. 2.45) we have $\partial D = \partial(F_1)\partial(\mathcal{M})\partial(V)\partial(F_2) \equiv_{\mathcal{A}} \partial(F_1)\partial(\mathcal{N})\partial(U)\partial(F_2) = \partial E$. This mapping defines a function $\partial : \mathsf{Deriv}_{\mathcal{A}^{\mathfrak{m}}}/\equiv_{\mathcal{A}^{\mathfrak{m}}} \to \mathsf{Deriv}_{\mathcal{A}}/\equiv_{\mathcal{A}}$ also noted ∂.

To conclude, let us show that they are mutual inverses. One side is immediate, namely $\partial(\mathtt{i}(R_1 \dots R_n)) = \partial(\{R_1\} \dots \{R_n\}) = R_1 \dots R_n$. For the other side, $\mathtt{i}(\partial(\mathcal{M}_1 \dots \mathcal{M}_n)) = \mathtt{i}(\rho_1 \dots \rho_n)$ where ρ_i is a complete development of $\mathcal{M}_i$ for all i. By Lem. 2.52, we have that $\mathtt{i}(\rho_1 \dots \rho_n) \equiv_{\mathcal{A}^{\mathfrak{m}}} \mathcal{M}_1 \dots \mathcal{M}_n$ as required. □

Lemma 2.54. *Let $\mathcal{A}$ be an orthogonal axiomatic rewriting system. Then* $\mathtt{i}(\rho/\sigma) \equiv_{\mathcal{A}^{\mathtt{m}}} \mathtt{i}(\rho)/\mathtt{i}(\sigma)$.

Proof. Let us write ρ as of the form $\rho = R_1 \dots R_n$. Let us moreover define $\sigma_i := \sigma/R_1 \dots R_{i-1}$ for all $1 \leqslant i \leqslant n$. Then:

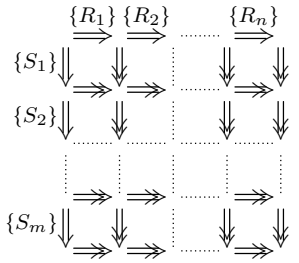
$$\begin{array}{lcll} \mathtt{i}(\rho/\sigma) & = & \mathtt{i}(R_1 \dots R_n/\sigma) & \\ & = & \mathtt{i}(\partial(R_1/\sigma_1) \dots \partial(R_n/\sigma_n)) & \\ & = & \mathtt{i}(\partial(R_1/\sigma_1)) \dots \mathtt{i}(\partial(R_n/\sigma_n)) & \\ & \equiv_{\mathcal{A}^{\mathtt{m}}} & (R_1/\sigma_1) \dots (R_n/\sigma_n) & \text{by Prop. 2.53} \\ & = & \{R_1\} \dots \{R_n\}/\mathtt{i}(\sigma) & \\ & = & \mathtt{i}(\rho)/\mathtt{i}(\sigma) & \end{array}$$

□

Theorem 2.55 (Cube identity for derivations)**.** *The following holds in any orthogonal axiomatic rewriting system $\mathcal{A}$:*

$$\rho(\sigma/\rho) \equiv \sigma(\rho/\sigma)$$

Proof. The proof of this fact requires working in the rewriting system of multisteps $\mathcal{A}^{\mathtt{m}}$, which is orthogonal by Prop. 2.51. The sketch of the proof is as follows: let $\rho = R_1 \dots R_n$ and $\sigma : S_1 \dots S_m$. Consider them as sequences of multisteps $D = \{R_1\} \dots \{R_n\}$ and $E = \{S_1\} \dots \{S_m\}$. A peak $(\Leftarrow\Rightarrow)$ in the rewriting system of multisteps may be closed with at most one step on each side $(\Rightarrow\!\!\!\!\Rightarrow\Leftarrow\!\!\!\!\Leftarrow)$, as a consequence of the cube identity for multisteps (Lem. 2.45). So the peak formed by D and E may be closed with square tiles (each side of a tile may be a single multistep or the empty derivation ϵ):



The complete development of the multisteps on one side of the diagram is precisely $\rho(\sigma/\rho)$, and the complete development of the multisteps on the other side of the diagram is precisely $\sigma(\rho/\sigma)$. Moreover, the cube identity for multisteps (Lem. 2.45) ensures that the diagram commutes, *i.e.* that the sides of each tile are permutation equivalent in $\mathcal{A}^{\mathtt{m}}$. By Prop. 2.53 this implies that $\rho(\sigma/\rho) \equiv \sigma(\rho/\sigma)$ in $\mathcal{A}$. In Paul-André Melliès' PhD thesis [118, Chapter 2] the reader may find a more detailed proof. □

As a consequence of this theorem we obtain a strong version of confluence, called *algebraic confluence* by Melliès:

Corollary 2.56 (Algebraic confluence)**.** *Let $\rho : x \twoheadrightarrow y$ and $\sigma : x \twoheadrightarrow z$. Then there is an object w such that $\sigma/\rho : y \twoheadrightarrow w$ and $\rho/\tau : z \twoheadrightarrow w$.*

Proof. Immediate since by Thm. 2.55 we have that $\rho(\sigma/\rho) \equiv \sigma(\rho/\sigma)$. □

Another consequence of Thm. 2.55 is that the set of derivations in any orthogonal axiomatic rewriting system can be given the higher-order structure of a category with pushouts. More precisely, one can define a transitive relation between derivations $\rho \sqsubseteq \sigma$, called the *prefix order*, and the binary operation of *join* of derivations $\rho \sqcup \sigma$ which is the least upper bound of $\{\rho, \sigma\}$ with respect to the order $\sqsubseteq$, up to permutation equivalence.

Definition 2.57 (Prefix order)**.** Given two coinitial derivations ρ, σ, we say that ρ is a *prefix* of σ, written $\rho \sqsubseteq \sigma$, if and only if $\rho/\sigma = \epsilon$.

Lemma 2.58 (Characterization of the prefix order)**.** *The following are equivalent:*

1. $\rho \sqsubseteq \sigma$,
2. $\rho(\sigma/\rho) \equiv \sigma$,
3. $\rho\tau \equiv \sigma$ *for some derivation* τ.

Proof. $(1 \implies 2)$ Suppose that $\rho \sqsubseteq \sigma$, *i.e.* $\rho/\sigma = \epsilon$. Then by Thm. 2.55 we have $\rho(\sigma/\rho) \equiv \sigma(\rho/\sigma) = \sigma$. $(2 \implies 3)$ Immediate by taking $\tau := \sigma/\rho$. $(3 \implies 1)$ Suppose that $\rho\sigma \equiv \rho$. Then $\rho/\sigma = \rho/\rho\tau = (\rho/\rho)/\tau = \epsilon$ by Lem. 2.50. □

Lemma 2.59 (Projection equivalence)**.** *The equivalence* $\rho \equiv \sigma$ *holds if and only if* $\rho \sqsubseteq \sigma$ *and* $\sigma \sqsubseteq \rho$.

Proof. $(\Rightarrow)$ Suppose that $\rho \equiv \sigma$. Then $\rho/\sigma = \rho/\rho = \epsilon$, so indeed $\rho \sqsubseteq \sigma$. Symmetrically, $\sigma \sqsubseteq \rho$. $(\Leftarrow)$ Suppose that $\rho \sqsubseteq \sigma$ and $\sigma \sqsubseteq \rho$, that is to say $\rho/\sigma = \epsilon$ and $\sigma/\rho = \epsilon$. Then by Thm. 2.55 we have that $\rho = \rho(\sigma/\rho) \equiv \sigma(\rho/\sigma) = \sigma$. □

Definition 2.60 (Join of derivations)**.** If ρ and σ are coinitial derivations, their *join*, written $\rho \sqcup \sigma$ is defined as $\rho \sqcup \sigma := \rho(\sigma/\rho)$.

Lemma 2.61 (Properties of $\sqcup$)**.** *The join of derivations has the following properties:*

1. $\rho \sqcup \sigma \equiv \sigma \sqcup \rho$.
2. $\rho \sqsubseteq \rho \sqcup \sigma$ *and* $\sigma \sqsubseteq \rho \sqcup \sigma$, i.e. $\rho \sqcup \sigma$ *is an upper bound of* $\{\rho, \sigma\}$.
3. $(\rho \sqcup \sigma)/\tau = (\rho/\tau) \sqcup (\sigma/\tau)$.
4. *If* $\rho \sqsubseteq \tau$ *and* $\sigma \sqsubseteq \tau$ *then* $\rho \sqcup \sigma \sqsubseteq \tau$, i.e. $\rho \sqcup \sigma$ *is the least upper bound of* $\{\rho, \sigma\}$*, up to permutation equivalence.*

Proof. Let us prove each item separately:

1. We have $\rho \sqcup \sigma = \rho(\sigma/\rho) \equiv \sigma(\rho/\sigma) = \sigma \sqcup \rho$ by Thm. 2.55.
2. Note that $\rho/(\rho \sqcup \sigma) = \rho/\rho(\sigma/\rho) = (\rho/\rho)/(\sigma/\rho) = \epsilon$ by Lem. 2.50 so indeed $\rho \sqsubseteq \rho \sqcup \sigma$. Moreover, using item 1. we have that $\sigma/(\rho \sqcup \sigma) = \sigma/(\sigma \sqcup \rho) = \epsilon$.

3. Note that, using Lem. 2.50 and Thm. 2.55:

$$\begin{array}{rcll}(\rho \sqcup \sigma)/\tau & = & \rho(\sigma/\rho)/\tau & \\ & = & (\rho/\tau)((\sigma/\rho)/(\tau/\rho)) & \\ & = & (\rho/\tau)(\sigma/\rho(\tau/\rho)) & \\ & = & (\rho/\tau)(\sigma/\tau(\rho/\tau)) & \text{since } \rho(\tau/\rho) \equiv \tau(\rho/\tau) \\ & = & (\rho/\tau)((\sigma/\tau)/(\rho/\tau)) & \\ & = & (\rho/\tau) \sqcup (\sigma/\tau) & \end{array}$$

4. Let $\rho/\tau = \epsilon$ and $\sigma/\tau = \epsilon$. Then using item 3. we have that $(\rho \sqcup \sigma)/\tau = (\rho/\tau) \sqcup (\sigma/\tau) = \epsilon \sqcup \epsilon = \epsilon$.

□

Proposition 2.62 (Compatibility with projection for multisteps)**.** *Let $\mathcal{A}$ be an orthogonal axiomatic rewriting system. If $D \equiv_{\mathcal{A}^{\mathfrak{m}}} E$ then $D/F \equiv_{\mathcal{A}^{\mathfrak{m}}} E/F$.*

Proof. It suffices to show the property when F is a single multistep; the main result then follows by induction on F. So let $D \equiv_{\mathcal{A}^{\mathfrak{m}}} E$ and let us show that $D/\mathcal{P} \equiv_{\mathcal{A}^{\mathfrak{m}}} E/\mathcal{P}$. We proceed by induction on the derivation of $D \equiv_{\mathcal{A}^{\mathfrak{m}}} E$. The interesting case is one-step permutation, *i.e.* when $D = F_1\mathcal{M}VF_2$ and $E = F_1\mathcal{N}UF_2$ where $(\mathcal{M}V, \mathcal{N}U)$ is a permutation tile in $\mathcal{A}^{\mathfrak{m}}$, *i.e.* U is a complete development of $\mathcal{M}/\mathcal{N}$ and V is a complete development of $\mathcal{N}/\mathcal{M}$.

The multiderivations U and V may be empty or they may consist of exactly one multistep. In any case, for any multistep $\mathcal{P}$ we have that $\langle \mathcal{M}V/\mathcal{P}\rangle = \langle \mathcal{N}U/\mathcal{P}\rangle$, because given an arbitrary step R, we have that:

$$\begin{array}{rcll}R/(\mathcal{M}V/\mathcal{P}) & = & R/(\mathcal{M}/\mathcal{P})(V/(\mathcal{P}/\mathcal{M})) & \\ & = & R/(\mathcal{M}/\mathcal{P})((\mathcal{N}/\mathcal{M})/(\mathcal{P}/\mathcal{M})) & \\ & = & R/(\mathcal{M}/\mathcal{P})((\mathcal{N}/\mathcal{P})/(\mathcal{M}/\mathcal{P})) & \text{by the cube identity (Lem. 2.45)} \\ & = & R/(\mathcal{N}/\mathcal{P})((\mathcal{M}/\mathcal{P})/(\mathcal{N}/\mathcal{P})) & \text{by the cube identity (Lem. 2.45)} \\ & = & R/(\mathcal{N}/\mathcal{P})((\mathcal{M}/\mathcal{N})/(\mathcal{P}/\mathcal{N})) & \text{by the cube identity (Lem. 2.45)} \\ & = & R/(\mathcal{N}/\mathcal{P})(U/(\mathcal{P}/\mathcal{N})) & \\ & = & R/(\mathcal{N}U/\mathcal{P}) & \end{array}$$

From this fact, we may conclude that $\mathcal{M}V/\mathcal{P} \equiv \mathcal{N}U/\mathcal{P}$ for any multistep $\mathcal{P}$. Then:

$$\begin{array}{rcl}D/\mathcal{P} & = & F_1\mathcal{M}VF_2/\mathcal{P} \\ & = & (F_1/\mathcal{P})(\mathcal{M}V/(\mathcal{P}/F_1))(F_2/(\mathcal{P}/F_1\mathcal{M}V)) \\ & \equiv & (F_1/\mathcal{P})(\mathcal{N}U/(\mathcal{P}/F_1))(F_2/(\mathcal{P}/F_1\mathcal{M}V)) \\ & = & F_1\mathcal{N}UF_2/\mathcal{P} \\ & = & E/\mathcal{P} \end{array}$$

□

Proposition 2.63 (Compatibility with projection)**.** *Let $\mathcal{A}$ is an orthogonal axiomatic rewriting system. If $\rho \equiv_{\mathcal{A}} \sigma$ then $\rho/\tau \equiv_{\mathcal{A}} \sigma/\tau$.*

Proof. Let $\rho \equiv_{\mathcal{A}} \sigma$ and let us show that $\rho/\tau \equiv_{\mathcal{A}} \sigma/\tau$. By Prop. 2.53, we know that $\mathtt{i}(\rho) \equiv_{\mathcal{A}^{\mathrm{m}}} \mathtt{i}(\sigma)$, and it suffices to show that $\mathtt{i}(\rho/\tau) \equiv_{\mathcal{A}^{\mathrm{m}}} \mathtt{i}(\sigma/\tau)$. Indeed:

$$\begin{array}{lll} \mathtt{i}(\rho/\tau) & \equiv_{\mathcal{A}^{\mathrm{m}}} \mathtt{i}(\rho)/\mathtt{i}(\tau) & \text{by Lem. 2.54} \\ & \equiv_{\mathcal{A}^{\mathrm{m}}} \mathtt{i}(\sigma)/\mathtt{i}(\tau) & \text{by compatibility in } \mathcal{A}^{\mathrm{m}} \text{ (Prop. 2.62)} \\ & \equiv_{\mathcal{A}^{\mathrm{m}}} \mathtt{i}(\sigma/\tau) & \text{by Lem. 2.54} \end{array}$$

□

Corollary 2.64 (Left cancellation)**.** *If $\rho\sigma \equiv \rho\tau$ then $\sigma \equiv \tau$.*

Proof. An immediate consequence of Prop. 2.63, projecting by ρ. □

2.3 The λ-Calculus

Much has been written about the λ-calculus. The syntax of the (pure, untyped) λ-calculus and the β-reduction rule have already been informally discussed in Section 1.1. We omit its formal definition, referring the reader to standard reference material on the topic [22, 95, 130, 74]. In particular we omit the formal treatment of the equivalence of terms modulo renaming of bound variables (α-equivalence). In the following, we do recall a few important concepts.

Convention 2.65 (Barendregt's free variable convention)**.** *During theorems and proofs, we may always assume that bound variables have been renamed apart from bound variables and from each other.*

2.3.1 Positions and Contexts

Two simple but important notions are those of *positions* and *contexts*. Generally speaking, a position is a string of symbols intended to represent a location in a tree. The empty string ϵ represents the root of the tree. A position p representing a node in the tree may be extended with an integer i to represent its i-th child. Given two positions p, q we write $p \cdot q$ so stand for their *concatenation*. Each term t of the λ-calculus has a set of positions $\mathsf{pos}(t)$:

Definition 2.66 (Positions of a λ-term)**.** If t is a λ-term, the set of positions $\mathsf{pos}(t)$ is defined as follows by induction on t:

$$\begin{array}{rcl} \mathsf{pos}(x) & \stackrel{\text{def}}{=} & \{\epsilon\} \\ \mathsf{pos}(ts) & \stackrel{\text{def}}{=} & \{\epsilon\} \cup \{1 \cdot p \mid p \in \mathsf{pos}(t)\} \cup \{2 \cdot p \mid p \in \mathsf{pos}(s)\} \\ \mathsf{pos}(\lambda x.t) & \stackrel{\text{def}}{=} & \{\epsilon\} \cup \{1 \cdot p \mid p \in \mathsf{pos}(t)\} \end{array}$$

For example, the positions of $\lambda x.xy$ are $\{\epsilon, 1, 11, 12\}$. Positions are used to perform "surgery" on terms: if $p \in \mathsf{pos}(t)$ is a position then $t|_p$ denotes the subterm of t at position p, and $t[s]_p$ denotes the term obtained by replacing the subterm at position p of t by s. For example, $(\lambda x.xy)|_{12} = y$ and $(\lambda x.xy)[xy]_{11} = \lambda x.xyy$. We will freely use the notion of position in other settings, besides the λ-calculus, without always giving an explicit definition.

The notion of position is very closely related with the notion of *context*. A context is a term with exactly one occurrence of a free variable $\Box$, called a *hole*.

Definition 2.67 (Contexts in the λ-calculus)**.** A context is defined by the following abstract syntax:

$$\mathtt{C} ::= \Box \mid \lambda x.\mathtt{C} \mid \mathtt{C}\, t \mid t\, \mathtt{C}$$

Each position $p \in \mathsf{pos}(t)$ corresponds to a context $t[\Box]_p$. For example, $(\lambda x.xy)[\Box]_{11}$ is the context $\lambda x.\Box y$. Contexts may also be used to perform surgery: if $\mathtt{C}$ is a context and t is a term, then $\mathtt{C}\langle t\rangle$ denotes the *capturing* substitution of $\Box$ by t in $\mathtt{C}$, so that, for example, $(\lambda x.x\Box)\langle xx\rangle = \lambda x.x(xx)$. In this thesis we also write $\mathtt{C}\langle\langle t\rangle\rangle$ for the *capture-avoiding* substitution of $\Box$ by t in $\mathtt{C}$, so that, for example, $(\lambda x.x\Box)\langle xx\rangle = \lambda z.z(xx)$. As for positions, we will freely use the notion of context for other families of terms, besides λ-terms, without always giving an explicit definition.

Contexts are useful to decompose a term by writing it as $\mathtt{C}\langle t\rangle$, where $\mathtt{C}$ is a partially known term with an unknown subterm $\Box$, in which one plugs the subterm t. For example, a term is of the form $\mathtt{C}\langle\langle x\rangle\rangle$ if and only if it has a free occurrence of x. Sometimes it is also useful to decompose terms using contexts with more than one hole. A *context with n holes* is a term with exactly one occurrence of the free variable $\Box_i$ for each $i \in \{1, \dots, n\}$. If $\mathtt{C}$ is a context with n holes, we write $\mathtt{C}\langle t_1, \dots, t_n\rangle$ for the result of substituting each $\Box_i$ for t_i in $\mathtt{C}$. For example, $((\lambda x.x\Box_2)\Box_1)\langle t, s\rangle = (\lambda x.xs)t$.

2.3.2 Residual Theory for the λ-Calculus

From the *propositional* point of view, the β-reduction rule of the λ-calculus is a binary relation between λ-terms:

$$\mathtt{C}\langle(\lambda x.t)\, s\rangle \to_\beta \mathtt{C}\langle t\{x := s\}\rangle$$

From the *relevant* point of view, however, the λ-calculus is an abstract rewriting system whose objects are λ-terms and whose steps are defined as follows:

Definition 2.68 (Step in the λ-calculus)**.** A step in the λ-calculus is a 4-uple $R = (\mathtt{C}, x, t, s)$. The source of R is $\mathtt{C}\langle(\lambda x.t)s\rangle$ and its target is $\mathtt{C}\langle t\{x := s\}\rangle$.

For example, there is a step $(\lambda x.xx)yz \to_\beta yyz$, given by the 4-uple $(\Box z, x, xx, y)$. An important observation is that a step is *not* uniquely determined by its source and target. For example, if $I = \lambda x.x$, there are two different steps $I(Iy) \to_\beta Iy$ namely $R = (\Box, x, x, Iy)$ and $S = (I\Box, x, x, y)$. This kind of situation is called a *syntactic accident* by Lévy [109]. In spite of the possibility of syntactic accidents, we usually do not work formally with 4-uples, since the step is usually clear from the context, *e.g.* when we say "the step $(\lambda x.x)t \to_\beta t$" we actually mean "the step $(\Box, x, x, t)$".

The λ-calculus can be endowed with the structure of an orthogonal axiomatic rewriting system (as defined in Def. 2.39). There are many (equivalent) ways to define the notion of *residual* in the λ-calculus. One way is by tracking *descendants*, using positions. Here we define residuals by means of an auxiliary calculus in which some redexes may be *marked*.

Definition 2.69 (Marked λ-calculus)**.** Assume given a denumerable set of *marks* $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}, \dots$. The set of *marked* terms is given by:

$$t ::= x \mid \lambda x.t \mid t\, t \mid (\lambda x^{\mathfrak{a}}.t)\, t$$

The *marked λ-calculus* has a single rule, closed by arbitrary contexts, that allows to contract any marked redex. The notation $t\{x := s\}$ stands for the capture-avoiding substitution of x by s in t.

$$(\lambda x^{\mathfrak{a}}.t)s \to_{\mathsf{M}} t\{x := s\}$$

A step in the marked λ-calculus is a 5-uple $(\mathtt{C}, x, \mathfrak{a}, t, s)$ whose source is $\mathtt{C}\langle(\lambda x^{\mathfrak{a}}.t)\, s\rangle$ and its target is $\mathtt{C}\langle t\{x := s\}\rangle$. The *name* of a step $R = (\mathtt{C}, x, \mathfrak{a}, t, s)$ in the marked λ-calculus is the mark $\mathfrak{a}$. A term is *initially marked* if it has no subterms of the form $(\lambda x.t)s$ and marks are pairwise distinct. A marked term t is a *variant* of an (unmarked) term t' if t' is the result of erasing all marks from t. Similarly, a marked step $R = (\mathtt{C}, x, \mathfrak{a}, t, s)$ is a variant of an (unmarked) step $R' = (\mathtt{C}', x, t', s')$ if $\mathtt{C}$, t, and s are variants of $\mathtt{C}'$,t', and s' respectively. If the marked term t is an initially marked variant of t' and $R' : t' \to_\beta s'$ is an unmarked step, there is a unique marked step $R : t \to_{\mathsf{M}} s$ such that R is a variant of R', we say that R is the *marked lift* of R' with respect to t.

For example, $(\lambda x^{\mathfrak{a}}.x)((\lambda y^{\mathfrak{b}}.y)z)$ is an initially marked term, but the terms $(\lambda x^{\mathfrak{a}}.x)((\lambda y^{\mathfrak{a}}.y)z)$ and $(\lambda x^{\mathfrak{a}}.x)((\lambda y.y)z)$ are not initially marked. The marked step

$$(\lambda x^{\mathfrak{a}}.x)((\lambda y^{\mathfrak{b}}.y)z) \to_{\mathsf{M}} ((\lambda y^{\mathfrak{b}}.y)z)$$

is the marked lift of the unmarked step

$$(\lambda x.x)((\lambda y.y)z) \to_\beta ((\lambda y.y)z)$$

with respect to the marked term $(\lambda x^{\mathfrak{a}}.x)((\lambda y^{\mathfrak{b}}.y)z)$. The notion of residual in the λ-calculus may be defined using the marked λ-calculus as an auxiliary tool, as follows.

Definition 2.70 (Residuals in the λ-calculus)**.** Let $R : t \to_\beta s$ and $S : t \to_\beta u$ be coinitial steps in the λ-calculus. The set of residuals R/S is defined as follows:

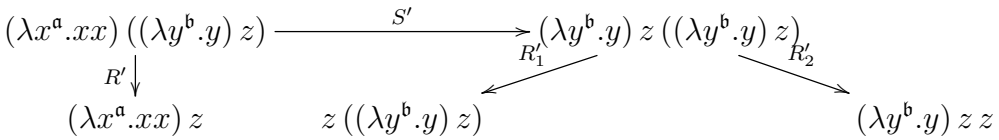
1. Let t' be an initially marked variant of t.
2. Let $R' : t' \to_{\mathsf{M}} s'$ and $S' : t' \to_{\mathsf{M}} u'$ be the marked lifts of R and S respectively.
3. A step $T : u \to_\beta r$ is a *residual* of R after S if and only if it has a marked variant $T' : u' \to_{\mathsf{M}} r'$ with the same name as R'.

Remark 2.71. The definition of residual does not depend on the choice of the initially marked variant t'.

Example 2.72. *Let $\Delta = \lambda x.xx$ and $I = \lambda x.x$, and let moreover*

$$\begin{array}{rlll} R: & \Delta\,(I\,z) & \to_\beta & \Delta\,z \\ S: & \Delta\,(I\,z) & \to_\beta & I\,z\,(I\,z) \\ R_1: & I\,z\,(I\,z) & \to_\beta & z\,(I\,z) \\ R_2: & I\,z\,(I\,z) & \to_\beta & I\,z\,z \end{array}$$

then $R/S = \{R_1, R_2\}$, as witnessed by the following diagram in the marked λ-calculus:



Theorem 2.73. *The λ-calculus is an orthogonal axiomatic rewriting system.*

Proof. The properties of autoerasure (AE) and finite residuals (FR) are easy to check. The property of finite developments (FD) of the λ-calculus can be reduced to the property that the marked λ-calculus is strongly normalizing. To prove SN of the marked λ-calculus, one may assign a *weight* $m(t) \in \mathbb{N}_0$ to any marked term t, representing the length of the longest sequence of steps starting from t:

$$\begin{aligned} m(x) &\stackrel{\text{def}}{=} 0 \\ m(\lambda x.t) &\stackrel{\text{def}}{=} m(t) \\ m(ts) &\stackrel{\text{def}}{=} m(t) + m(s) \\ m((\lambda x^{\mathfrak{a}}.t)s) &\stackrel{\text{def}}{=} 1 + m(t) + \max\{1, m_x(t)\} \cdot m(s) \end{aligned}$$

where in turn $m_x(t)$ represents the maximum *potential multiplicity* of x along any sequence starting from t:

$$\begin{aligned} m_x(y) &\stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \\ m_x(\lambda y.t) &\stackrel{\text{def}}{=} m_x(t) && \text{if } x \neq y \\ m_x(ts) &\stackrel{\text{def}}{=} m_x(t) + m_x(s) \\ m_x((\lambda y^{\mathfrak{a}}.t)s) &\stackrel{\text{def}}{=} 1 + m_x(t) + \max\{1, m_y(t)\} \cdot m_x(s) && \text{if } x \neq y \end{aligned}$$

It can then be checked that $t \to_{\mathsf{M}} s$ implies $m(t) > m(s)$, which in turn means that the marked calculus is SN. The key fact is that $m(t\{x := s\}) \leqslant m(t) + m_x(t) \cdot m(s)$ holds for all t, x, s, which can be proved by induction on t.[2]

The property of semantic orthogonality (SO) can be reduced to the property that the marked λ-calculus is weakly Church–Rosser. The difficult case is when a step R *nests* another step S, that is, when the subterm contracted by the step S lies inside the argument of the application contracted by R. Then the peak may be closed with a diagram of the form:

$$\begin{array}{ccc} (\lambda x^{\mathfrak{a}}.t)\, \mathtt{C}\langle(\lambda y^{\mathfrak{b}}.s)u\rangle & \xrightarrow{R} & t\{x := \mathtt{C}\langle(\lambda y^{\mathfrak{b}}.s)u\rangle\} \\ \Big\downarrow{\scriptstyle S} & & \vdots \\ (\lambda x^{\mathfrak{a}}.t)\, \mathtt{C}\langle s\{y := u\}\rangle & \dashrightarrow & t\{x := \mathtt{C}\langle s\{y := u\}\rangle\} \end{array}$$

See [22, Lemma 11.2.23] for a detailed proof that the marked λ-calculus is WCR. □

2.4 The Linear Substitution Calculus

The syntax of the Linear Substitution Calculus (LSC) and its reduction rules have been informally discussed in Section 1.1. Below we briefly state these definitions.

[2] This direct definition of a bound for the length of the longest development of a term is due to de Vrijer [53]. See [22, Theorem 11.2.21] for a different proof.

Definition 2.74 (Terms and contexts)**.** Terms ($t, s, \ldots$), arbitrary contexts ($\mathtt{C}, \mathtt{C}_2, \ldots$), and substitution contexts ($\mathtt{L}, \mathtt{L}_2, \ldots$) are defined as follows:

$$\begin{array}{rcl} t & ::= & x \mid t\,t \mid \lambda x.t \mid t[x\backslash t] \\ \mathtt{C} & ::= & \Box \mid \mathtt{C}\,t \mid t\,\mathtt{C} \mid \lambda x.\mathtt{C} \mid \mathtt{C}[x\backslash t] \mid t[x\backslash \mathtt{C}] \\ \mathtt{L} & ::= & \Box \mid \mathtt{L}[x\backslash t] \end{array}$$

A *pure term* is a term without explicit substitutions. Recall that if $\mathtt{L}$ is a substitution context, we write $t\mathtt{L}$ rather than $\mathtt{L}\langle t\rangle$ for the result of plugging t into the hole of $\mathtt{L}$. The underlined occurrences of x in the terms $\lambda\underline{x}.t$ and $t[\underline{x}\backslash s]$ are supposed to be *binding* occurrences. More precisely, the set of free variables $\mathsf{fv}(t)$ of a term t is defined as follows:

$$\begin{array}{rcl} \mathsf{fv}(x) & \stackrel{\text{def}}{=} & \{x\} \\ \mathsf{fv}(t\,s) & \stackrel{\text{def}}{=} & \mathsf{fv}(t) \cup \mathsf{fv}(s) \\ \mathsf{fv}(\lambda x.t) & \stackrel{\text{def}}{=} & \mathsf{fv}(t)\backslash\{x\} \\ \mathsf{fv}(t[x\backslash s]) & \stackrel{\text{def}}{=} & (\mathsf{fv}(t)\backslash\{x\}) \cup \mathsf{fv}(s) \end{array}$$

As in the λ-calculus, terms are considered up to α-equivalence, *i.e.* renaming of bound variables.

Definition 2.75 (Reduction rules)**.** From the *propositional* point of view, the rewriting relation between terms ($\to_{\mathtt{LSC}}$) is defined as $\to_{\mathtt{LSC}} \stackrel{\text{def}}{=} \to_{\mathtt{db}} \cup \to_{\mathtt{ls}} \cup \to_{\mathtt{gc}}$, where $\to_x$ is defined as the contextual closure of $\mapsto_x$ for each $x \in \{\mathtt{db}, \mathtt{ls}, \mathtt{gc}\}$:

$$\begin{array}{rcll} (\lambda x.t)\mathtt{L}\,s & \mapsto_{\mathtt{db}} & t[x\backslash s]\mathtt{L} & \\ \mathtt{C}\langle\!\langle x\rangle\!\rangle[x\backslash t] & \mapsto_{\mathtt{ls}} & \mathtt{C}\langle\!\langle t\rangle\!\rangle[x\backslash t] & \\ t[x\backslash s] & \mapsto_{\mathtt{gc}} & t & \text{if } x \notin \mathsf{fv}(t) \end{array}$$

From the *relevant* point of view, steps in the LSC are given by the disjoint union of $\mathtt{db}$ steps, $\mathtt{ls}$ steps, and $\mathtt{gc}$ steps where:

- A $\mathtt{db}$ step is a 5-uple $R = (\mathtt{C}, x, t, \mathtt{L}, s) : \mathtt{C}\langle(\lambda x.t)\mathtt{L}\,s\rangle \to_{\mathtt{LSC}} \mathtt{C}\langle t[x\backslash s]\mathtt{L}\rangle$.
- A $\mathtt{ls}$ step is a 4-uple $R = (\mathtt{C}_1, \mathtt{C}_2, x, t) : \mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash t]\rangle \to_{\mathtt{LSC}} \mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle t\rangle\!\rangle[x\backslash t]\rangle$.
- A $\mathtt{gc}$ step is a 4-uple $R = (\mathtt{C}, t, x, s) : \mathtt{C}\langle t[x\backslash s]\rangle \to_{\mathtt{LSC}} \mathtt{C}\langle t\rangle$ such that $x \notin \mathsf{fv}(t)$.

A useful notion is that of the *anchor* of a step. The anchor of a $\mathtt{db}$ step $(\lambda\underline{x}.t)\mathtt{L}\,s \to t[x\backslash s]\mathtt{L}$ is the underlined (binding) occurrence of x. The anchor of a $\mathtt{ls}$ step $\mathtt{C}\langle\!\langle\underline{x}\rangle\!\rangle[x\backslash t] \to \mathtt{C}\langle\!\langle t\rangle\!\rangle[x\backslash t]$ is the underlined occurrence of x. The anchor of a $\mathtt{gc}$ step $t[\underline{x}\backslash s] \to t$ is the underlined (binding) occurrence of x.

Definition 2.76 (Graphical equivalence)**.** Terms of the LSC are provided with a binary relation $t \sim s$ called *graphical equivalence*. It is defined as the least congruence containing the three axioms below:

$$\begin{array}{rcll} (ts)[x\backslash u] & \sim_{@} & t[x\backslash u]s & \text{if } x \notin \mathsf{fv}(s) \\ (\lambda x.t)[y\backslash s] & \sim_{\lambda} & \lambda x.t[y\backslash s] & \text{if } x \notin \mathsf{fv}(s) \text{ and } x \neq y \\ t[x\backslash s][y\backslash u] & \sim_{com} & t[y\backslash u][x\backslash s] & \text{if } x \notin \mathsf{fv}(u) \text{ and } y \notin \mathsf{fv}(s) \end{array}$$

Recall that a congruence is an equivalence relation which is closed by arbitrary contexts, *i.e.* $t \sim s$ implies $\mathtt{C}\langle t\rangle \sim \mathtt{C}\langle s\rangle$.

In the following, we state a few results that justify that the LSC is a quite well-behaved explicit substitution calculus, and we sketch the ideas behind their proofs. For their formal proofs the reader should refer to [8, 2, 5].

Proposition 2.77 (Full Composition). *If t, s are terms in the LSC, then $t[x\backslash s] \twoheadrightarrow_{\mathtt{LSC}} t\{x := s\}$.*

Proof. Suppose that there are exactly n free occurrences of x in t, and write $t = \mathtt{C}\langle x, x, \ldots, x\rangle$ where $\mathtt{C}$ is an n-hole context, for $n \geqslant 0$. Then with a sequence of n `ls` steps and one `gc` step we have:

$$\begin{array}{rcll} t[x\backslash s] & = & \mathtt{C}\langle x, x, \ldots, x\rangle[x\backslash s] & \\ & \twoheadrightarrow_{\mathtt{LSC}} & \mathtt{C}\langle s, s, \ldots, s\rangle[x\backslash s] & \text{with } n \texttt{ ls} \text{ steps} \\ & \to_{\mathtt{LSC}} & \mathtt{C}\langle s, s, \ldots, s\rangle & \text{with a single } \texttt{gc} \text{ step} \\ & = & t\{x := s\} & \end{array}$$

□

Corollary 2.78 (Simulation of β-reduction). *The LSC simulates the λ-calculus, that is if $t \to_\beta s$ then $t \to^+_{\mathtt{LSC}} s$.*

Proof. A β-reduction step $\mathtt{C}\langle(\lambda x.t)\, s\rangle \to \mathtt{C}\langle t\{x := s\}\rangle$ can be simulated in the LSC as follows:

$$\begin{array}{rcll} \mathtt{C}\langle(\lambda x.t)\, s\rangle & \to_{\mathtt{LSC}} & \mathtt{C}\langle t[x\backslash s]\rangle & \text{with a } \texttt{db} \text{ step} \\ & \twoheadrightarrow_{\mathtt{LSC}} & \mathtt{C}\langle t\{x := s\}\rangle & \text{by Full Composition (Prop. 2.77)} \end{array}$$

□

Lemma 2.79 (Unfolding is terminating). *The relation $\to_{\mathtt{ls,gc}} \stackrel{\text{def}}{=} \to_{\mathtt{ls}} \cup \to_{\mathtt{gc}}$ is* SN.

Proof. A bound $m(t)$ for the length of the longest sequence of $\to_{\mathtt{ls,gc}}$ steps going out from a term t can be obtained as follows:

$$\begin{array}{rcl} m(x) & \stackrel{\text{def}}{=} & 0 \\ m(\lambda x.t) & \stackrel{\text{def}}{=} & m(t) \\ m(t\, s) & \stackrel{\text{def}}{=} & m(t) + m(s) \\ m(t[x\backslash s]) & \stackrel{\text{def}}{=} & m(t) + (1 + m_x(t)) \cdot (1 + m(s)) \end{array}$$

where in turn $m_x(t)$ represents the maximum *potential multiplicity* of x along any sequence of $\to_{\mathtt{ls,gc}}$ steps starting from t:

$$\begin{array}{rcl} m_x(y) & \stackrel{\text{def}}{=} & \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \\ m_x(\lambda y.t) & \stackrel{\text{def}}{=} & m_x(t) \\ m_x(t\, s) & \stackrel{\text{def}}{=} & m_x(t) + m_x(s) \\ m_x(t[y\backslash s]) & \stackrel{\text{def}}{=} & m_x(t) + (1 + m_y(t)) \cdot m_x(s) \end{array}$$

It can then be shown that if $t \to_{\mathtt{ls,gc}} s$ then $m(t) > m(s)$ which entails termination. □

Let us write SN_x for the set of strongly normalizing terms for the rewriting relation $\to_x$.

Theorem 2.80 (Preservation of strong normalization)**.** *If t is a pure term and $t \in \mathrm{SN}_\beta$ then $t \in \mathrm{SN}_{\mathsf{LSC}}$.*

Proof. We sketch a proof of PSN, without going into all the technical details, which would require quite a few auxiliary lemmas. A proof of PSN for the structural λ-calculus—a calculus closely related with the LSC— may be found in [8], and the proof can be easily adapted.

Define the *unfolding* of an LSC term t as the λ-term $t^\diamond$ that results from performing all substitutions, that is, the $\to_{\mathtt{ls},\mathtt{gc}}$-normal form of t. In a term of the form $\mathtt{C}\langle t[x\backslash s]\rangle$, we say that the substitution $[x\backslash s]$ under the context $\mathtt{C}$ is *sterile* if $x \notin \mathsf{fv}(t^\diamond)$. A subterm is *unreachable* if it lies inside a sterile substitution, and *reachable* otherwise. For example, in $x[y\backslash z][z\backslash t]$ the subterm t is unreachable because the underlined substitution $x[y\backslash z]\underline{[z\backslash t]}$ is sterile. A step $R : t \to s$ is *unreachable* if the anchor of R lies inside an unreachable subterm of t, and *reachable* otherwise. The rewriting relation of *unreachable reduction* $t \to_{\mathsf{U}} s$ is defined as the restriction of $t \to_{\mathsf{LSC}} s$ to unreachable steps. In turn, *reachable unfolding* $t \to_{\mathsf{R}(\mathtt{ls},\mathtt{gc})}$ is the restriction of $t \to_{\mathtt{ls},\mathtt{gc}} s$ to reachable steps.

Let X be the set of LSC terms such that $t^\diamond \in \mathrm{SN}_\beta$ and every unreachable subterm of t is in $\mathrm{SN}_{\mathsf{LSC}}$. Observe in particular that if t is pure and $t \in \mathrm{SN}_\beta$, then $t^\diamond = t \in \mathrm{SN}_\beta$ and t has no unreachable subterms, so in fact $t \in X$. The proof of the main statement can then be reduced to the claim that $X \subseteq \mathrm{SN}_{\mathsf{LSC}}$, which is Claim II below. We also need Claim I as an auxiliary result.

- **Claim I.** If $t \in X$, then for any subterm s of t we have that $s \in X$.
  Proof of Claim I. Let s be a subterm of t. Note that any unreachable subterm u of s is also an unreachable subterm in t, so we have that $u \in \mathrm{SN}_{\mathsf{LSC}}$. We are left to show that $s^\diamond \in \mathrm{SN}_\beta$. We consider two cases:
  - **Reachable.** Suppose that s is reachable. Then $s^\diamond$ occurs as a subterm of $t^\diamond$, and $t^\diamond \in \mathrm{SN}_\beta$ by hypothesis, so $s^\diamond \in \mathrm{SN}_\beta$.
  - **Unreachable.** Suppose that s is unreachable. The hypothesis that $t \in X$ implies that $s \in \mathrm{SN}_{\mathsf{LSC}}$. As a consequence, we have that $s^\diamond \in \mathrm{SN}_\beta$, since by Simulation (Coro. 2.78) an infinite sequence

    $$s^\diamond \to_\beta u_1 \to_\beta u_2 \to_\beta \dots$$

    results in an infinite sequence

    $$s \twoheadrightarrow_{\mathtt{ls},\mathtt{gc}} s^\diamond \to^+_{\mathsf{LSC}} u_1 \to^+_{\mathsf{LSC}} u_2 \to^+_{\mathsf{LSC}} \cdots$$

    contradicting $s \in \mathrm{SN}_{\mathsf{LSC}}$.
- **Claim II.** If $t \in X$ then $t \in \mathrm{SN}_{\mathsf{LSC}}$.
  Proof of Claim II. The proof proceeds by induction on the size of the term t. Since the

λ-calculus and the LSC are finitely branching, by Prop. 2.17 we may define the following notions of *depth*:

$$\begin{array}{rcll} \mathsf{depth}_{\beta}(t) & \stackrel{\text{def}}{=} & \max\{|\rho| \mid \rho : t \to_{\beta} s\} & \text{if } t \in \mathrm{SN}_{\beta} \\ \mathsf{depth}_{\mathsf{R(ls,gc)}}(t) & \stackrel{\text{def}}{=} & \max\{|\rho| \mid \rho : t \twoheadrightarrow_{\mathsf{R(ls,gc)}} s\} & \text{if } t \text{ is any LSC term} \\ \mathsf{depth}_{\mathsf{U}}(t) & \stackrel{\text{def}}{=} & \max\{|\rho| \mid \rho : t \twoheadrightarrow_{\mathsf{U}} s\} & \text{if } t \text{ is an LSC term whose} \\ & & & \text{unreachable subterms are in } \mathrm{SN}_{\mathsf{LSC}} \end{array}$$

Note that $\mathsf{depth}_{\mathsf{R(ls,gc)}}(t)$ is well-defined because $\to_{\mathtt{ls,gc}}$ is SN (Lem. 2.79) so in particular $\to_{\mathsf{R(ls,gc)}}$ is SN. The *measure* of t is written $\#(t)$ and defined as the triple:

$$\#(t) \stackrel{\text{def}}{=} (\mathsf{depth}_{\beta}(t^{\diamond}), \mathsf{depth}_{\mathsf{R(ls,gc)}}(t), \mathsf{depth}_{\mathsf{U}}(t))$$

It can then be shown that if $t \to_{\mathsf{LSC}} s$ then $\#(t) > \#(s)$ where $(>)$ is the lexicographic order. We consider three cases:

1. **Reachable** `db` **step.** Suppose that the step is of the form $t = \mathtt{C}\langle(\lambda x.u)\mathtt{L}\, r\rangle \to_{\mathtt{db}} \mathtt{C}\langle u[x\backslash r]\mathtt{L}\rangle = s$ and that it is reachable.

   First, we argue that $s \in X$. Note that $t^{\diamond} \to_{\beta}{}^{+} s^{\diamond}$ in at least one step, as can be checked by induction on t. Since $t^{\diamond} \in \mathrm{SN}_{\beta}$ then $s^{\diamond} \in \mathrm{SN}_{\beta}$. Moreover, consider an unreachable subterm of s, and let us check $s \in \mathrm{SN}_{\mathsf{LSC}}$. The unreachable subterms of s are the same ones as for t, except perhaps for r and its subterms. But r is smaller in size than t, and by Claim I $r \in X$, so by *i.h.* we have that $r \in \mathrm{SN}_{\mathsf{LSC}}$.

   Second, let us show that the measure decreases. We have already noted that $t^{\diamond} \to_{\beta}{}^{+} s^{\diamond}$, so $\mathsf{depth}_{\beta}(t^{\diamond}) > \mathsf{depth}_{\beta}(s^{\diamond})$ and the first component decreases.

2. **Reachable** `ls` **or** `gc` **step.** Suppose that the step is of the form $t \to_{\mathsf{R(ls,gc)}} s$ and that it is reachable.

   First, let us show that $s \in X$. Observe that $t^{\diamond} = s^{\diamond}$ so given that $t^{\diamond} \in \mathrm{SN}_{\beta}$, also $s^{\diamond} \in \mathrm{SN}_{\beta}$. Moreover, let us check that the unreachable subterms of s are in $\mathrm{SN}_{\mathsf{LSC}}$. If the step is a `gc`, it is immediate. If the step is a `ls` step then $t = \mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash u]\rangle \to_{\mathtt{ls}} \mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash u]\rangle = s$. The unreachable subterms of s are the same ones as in t, except perhaps for u and its subterms. But u is smaller in size than t, and by Claim I $u \in X$, so by *i.h.* we have that $u \in \mathrm{SN}_{\mathsf{LSC}}$.

   Second, let us show that the measure decreases. Since $t \to_{\mathsf{R(ls,gc)}} s$ we have that $\mathsf{depth}_{\mathsf{R(ls,gc)}}(t) > \mathsf{depth}_{\mathsf{R(ls,gc)}}(s)$ so the second component decreases. Moreover $t^{\diamond} = s^{\diamond}$ so $\mathsf{depth}_{\beta}(t^{\diamond}) = \mathsf{depth}_{\beta}(s^{\diamond})$, *i.e.* the first component does not change.

3. **Unreachable step.** Suppose that the step is unreachable, *i.e.* of the form $t \to_{\mathsf{U}} s$.

   First, note that $s \in X$ since $t^{\diamond} = s^{\diamond}$ and the reachable subterms of s are the same ones as in t.

   Second, let us show that the measure decreases. Given that $t \to_{\mathsf{U}} s$, we have that $\mathsf{depth}_{\mathsf{U}}(t) > \mathsf{depth}_{\mathsf{U}}(s)$ so the third component decreases. Note that $t^{\diamond} = s^{\diamond}$ so $\mathsf{depth}_{\beta}(t^{\diamond}) = \mathsf{depth}_{\beta}(s^{\diamond})$, *i.e.* the first component does not change. Moreover, $\mathsf{depth}_{\mathsf{R(ls,gc)}}(t) = \mathsf{depth}_{\mathsf{R(ls,gc)}}(s)$, *i.e.* the second component does not change.

□

Proposition 2.81 (Confluence)**.** *The LSC is confluent.*

Proof. We do not give a full proof here, but a few pointers:

- A proof of confluence for the structural λ-calculus—closely related with the LSC—may be found in [8].
- Confluence for the LSC is straightforward using *interpretation methods* [72]. A proof of a stronger property, *meta-confluence*, for the LSC can be found in [52].
- A proof that the LSC is an orthogonal axiomatic rewriting system may be found in [5]. Recall that orthogonal axiomatic rewriting systems enjoy the stronger property of algebraic confluence (Coro. 2.56).
- In Chapter 6 we will reconstruct a proof that the LSC is an orthogonal axiomatic rewriting system, using a labeled calculus.

□

Chapter 3

Distilling Abstract Machines

3.1 Introduction

The λ-calculus is a fine model of computation from the point of view of *computability*—it is Turing-complete. It is however not so clear whether the λ-calculus is a fine model of computation from the point of view of *computational complexity*. By this we mean the amount of resources that the program must consume to be able to run. There are many kinds of computational resources. In this chapter we are interested exclusively in the *time* complexity required to evaluate λ-terms. Time is a most fundamental computational resource, in that other kinds of resources, such as the amount of *space* (memory) that a program uses, or the amount of *energy* (*e.g.* battery) that it consumes, can usually be bounded proportionally by the running time of the program.

As mentioned in the introduction, van Emde Boas' Invariance Thesis stipulates that *reasonable* models of sequential computation should simulate each other with polynomial overhead in time [140]. For example, "traditional" (or "established") models of computation such as Turing machines and random-access machines are known to simulate each other with polynomial overhead. Is the λ-calculus a reasonable time cost model of sequential computation, with respect to the established models? That is, can a sequence of n consecutive β-reduction steps $t_0 \to_\beta t_1 \ldots \to_\beta t_n$ be simulated in a Turing machine with at most a polynomial number of steps in n?

As an illustration of why this question is subtle, note that there are families of λ-terms whose sizes grow exponentially as a function of the number of β-reduction steps. For instance, recall the following families of terms $(t_n)_{n\in\mathbb{N}}$ and $(s_n)_{n\in\mathbb{N}}$ from Section 1.1.2:

$$\begin{array}{rcl@{\qquad\qquad}rcl} t_0 & \stackrel{\text{def}}{=} & y & s_0 & \stackrel{\text{def}}{=} & y \\ t_{n+1} & \stackrel{\text{def}}{=} & (\lambda x.xx)t_n & s_{n+1} & \stackrel{\text{def}}{=} & s_n s_n \end{array}$$

These terms are such that the size of t_n is $\Theta(n)$ and the size of s_n is $\Theta(2^n)$, but t_n reduces to s_n in $\Theta(n)$ steps. Suppose that one represents terms straightforwardly as trees—be it in a Turing machine or in any other established model of sequential computation. With that representation, the amount of memory required to simulate n consecutive β-reduction steps,

starting from t_n, grows exponentially as a function of n. As a necessary consequence, the amount of time required to simulate n consecutive β-reduction steps also grows (at least) exponentially as a function of n.

The above example shows that it is not possible to simulate β-reduction in polynomial time *as long as* terms are represented straightforwardly as trees. The subtle point is that this does not forbid that the λ-calculus may turn out to be a reasonable time cost model if one were to rely on a smarter representation for λ-terms. In summary, regarding the question of whether the λ-calculus is a reasonable time cost model, answering it *positively* would require to conceive a sufficiently smart representation for λ-terms that avoids the exponential blowup in space. Conversely, answering it *negatively* would require to prove that simulating it with polynomial overhead in time is impossible for *any* conceivable encoding of λ-terms.

A noteworthy contribution to the study of this problem has been the work by Accattoli and dal Lago [11], who have shown that leftmost-outermost reduction in the λ-calculus is reasonable, by choosing an appropriate representation for λ-terms that avoids the exponential blowup. In fact, in order to share subterms, the Linear Substitution Calculus (LSC) is used as the primary technical tool. The general question of whether arbitrary β-reduction in the λ-calculus is a reasonable time cost model is currently open, as of the writing of this thesis.

In this chapter, we tackle the question of whether certain *reduction strategies* in explicit substitution calculi are reasonable cost models. For example, in the case of the *call-by-name* reduction strategy for the Linear Substitution Calculus (LSC), the question is whether it is possible to implement the LSC in such a way that n consecutive call-by-name reduction steps can be simulated—in an established model of sequential computation—with at most a polynomial number of steps in n. We answer this question positively for four particular reduction strategies: call-by-name, call-by-value, call-by-need, and strong call-by-name (*i.e.* call-by-name generalized to allow reduction under abstractions).

To be able to study these questions for a given reduction strategy, one needs to provide the following elements:

1. A formal definition of the **reduction strategy** itself.
2. An **implementation** of the reduction strategy.
3. A "**distillation**", *i.e.* a construction showing that the implementation actually implements reduction according to the given strategy.

All of these elements are grouped in an abstract structure that we call a **distillery** (see Def. 3.17). Throughout this chapter we develop a methodology to study distilleries. Besides the particular results on the time complexity of various evaluation strategies, the methodology itself is an important take-home point, for the following reasons:

- Distilleries are *uniform*: abstract machines are consistently seen as implementations of reduction strategies in a single framework—the λ-calculus extended with explicit substitutions. This allows us to understand the working of many existing abstract machines (*e.g.* the Krivine Abstract Machine or Landin's SECD machine) as less *ad hoc*.

- Distilleries are *modular* with respect to various features (*e.g.* local vs. global environments, or split vs. merged stacks)—so formulating variants of abstract machines with different features becomes a relatively mechanical task.
- This very uniform and modular approach can guide the design of future abstract machines to implement other reduction strategies. For instance in the conclusion (Section 8.1) we discuss an abstract machine for the strong call-by-need reduction strategy of Chapter 4.

In the remainder of this section, before diving into the specific details of each strategy, we give a description of the general methodology

First, in this chapter, a **reduction strategy** $\mathbb{S}$ is always an abstract rewriting system over the set Term of terms with explicit substitutions:

$$t ::= x \mid \lambda x.t \mid t\,t \mid t[x\backslash t]$$

and a binary reduction relation $\to_{\mathbb{S}} \subseteq \mathsf{Term} \times \mathsf{Term}$. In order to rigorously define the relation $\to_{\mathbb{S}}$, we use *evaluation contexts*, a technique introduced by Felleisen [55]. The set of $\mathbb{S}$-evaluation contexts is a subset of the set of all possible contexts. The position of the hole in an $\mathbb{S}$-evaluation context indicates where in a term evaluation should focus next, according to the strategy $\mathbb{S}$. For instance, the set of call-by-name evaluation contexts is given by the grammar:

$$\mathtt{H} ::= \Box \mid \mathtt{H}\,t \mid \mathtt{H}[x\backslash t]$$

Hence the following `db` step (underlined):

$$(\underline{(\lambda x.x)(yz)})[y\backslash(\lambda x.x)z][z\backslash w] \to x[x\backslash yz][y\backslash(\lambda x.x)z][z\backslash w]$$

is a step in the call-by-name strategy, as the redex is below the evaluation context $\mathtt{H} := \Box[y\backslash(\lambda x.x)z][z\backslash w]$, whereas the following `db` step:

$$((\lambda x.x)(yz))[y\backslash\underline{(\lambda x.x)z}][z\backslash w] \to ((\lambda x.x)(yz))[y\backslash x[x\backslash z]][z\backslash w]$$

is not a step in the call-by-name strategy, because the context $\mathtt{C} := ((\lambda x.x)(yz))[y\backslash\Box][z\backslash w]$ is not a call-by-name evaluation context.

The reduction relation $\to_{\mathbb{S}}$ for each strategy $\mathbb{S}$ that we study is always defined using a *multiplicative* (`db`-like) reduction rule, and an *exponential* (`ls`-like) reduction rule. The names obey to the fact that `db`-like rules correspond to multiplicative cut-elimination steps in the encoding of explicit substitution calculi using Linear Logic proof-nets and, likewise, `ls`-like rules correspond to exponential cut-elimination steps. The definition of a reduction strategy $\mathbb{S}$ will follow roughly the following template:

Multiplicative reduction rule (`db`-like)

$$\mathtt{E}\langle(\lambda x.t)\mathtt{L}\,s\rangle \quad \to_{\mathbb{S}} \quad \mathtt{E}\langle t[x\backslash s]\mathtt{L}\rangle \qquad \text{if } \mathtt{E} \text{ is an } \mathbb{S}\text{-evaluation context}$$

Exponential reduction rule (`ls`-like)

$$\mathtt{E}_1\langle\mathtt{E}_2\langle\!\langle x\rangle\!\rangle[x\backslash t]\rangle \quad \to_{\mathbb{S}} \quad \mathtt{E}_1\langle\mathtt{E}_2\langle\!\langle t\rangle\!\rangle[x\backslash t]\rangle \quad \text{if } \mathtt{E}_1\langle\mathtt{E}_2\rangle[x\backslash t] \text{ is an } \mathbb{S}\text{-evaluation context}$$

One obvious difference with respect to the LSC is that reduction rules are closed by *evaluation contexts*, rather than by arbitrary contexts. Moreover, each strategy may incorporate slight (or not so slight) variations; for example the call-by-value strategy will require that the argument of a multiplicative step is already an *answer*, *i.e.* a term of the form $(\lambda x.t)\mathtt{L}$.

All the reduction strategies studied in this chapter turn out to be *deterministic*, *i.e.* if $t \to_{\mathbb{S}} s_1$ and $t \to_{\mathbb{S}} s_2$ are steps in a given reduction strategy then $s_1 = s_2$. Moreover, as mentioned, these strategies always select either a `db`-like or a `ls`-like step, and they do not perform garbage collection (there are no `gc`-like rules). As a consequence, the answer obtained as the result of evaluating a term may contain *unreachable* explicit substitutions. The decision to ignore the `gc`-rule in the analysis is justified by the following observations:

1. On one hand, `gc` steps in the explicit substitution calculi that we study do not interfere with other kinds of computation steps. More precisely, `gc` steps can be *postponed*: for every reduction sequence $t \twoheadrightarrow s$ there is a term u such that $t \twoheadrightarrow_{\mathtt{db} \cup \mathtt{ls}} u \twoheadrightarrow_{\mathtt{gc}} s$.

   Formally, the garbage collection rule will be incorporated into an equivalence relation of *structural equivalence* between terms, and we will show that structurally equivalent terms have the same computational behavior.

2. On the other hand, explicit substitution calculi do not allow for cyclic bindings. That is, if a term with explicit substitutions is interpreted as a directed graph in which some subterms are shared, the graph turns out to be acyclic.

   This means that garbage collection of unreachable explicit substitutions may be implemented using the elementary technique known as *reference counting* [82]. Concretely, each explicit substitution $t[x\backslash s]$ may be annotated with an integer $n \geqslant 0$ that counts the free occurrences of x in t. This count must be updated after each reduction step, and the explicit substitution may be reclaimed when the count reaches zero. Moreover, all the implementations that we propose enjoy the *subterm property* (see below), which means that these updates can be done in "constant" time[1].

   As a consequence, incorporating the `gc` rule is not interesting from the point of view of time complexity, and it is left out of the analysis.

Second, **implementations** in this chapter are always defined using *abstract machines*. An abstract machine $\mathbb{M}$ is also an abstract rewriting system, over a set State of states, and rules that define a binary *transition* relation $\rightsquigarrow_{\mathbb{M}} \subseteq \mathsf{State} \times \mathsf{State}$ between states. The concrete definition of the set State varies from machine to machine, but typically a state is a tuple consisting of such elements as:

- A *code*, that is, a term representing the expression that is currently being evaluated. While in reduction strategies we work implicitly modulo α-equivalence, for machines we will *not* do so, as renaming of variables is part of what an abstract machine may have to explicitly do, and different renaming schemes correspond to different approaches to

[1] Actually in time proportional to the size of the starting term.

abstract machines. We use the metavariables $\overline{t}, \overline{s}, \overline{u}, \overline{r}$ for code, that is, terms without explicit substitutions and *not* up to α-equivalence.

- A *stack* $\pi = c_1 :: \ldots :: c_n$, into which the arguments are pushed.
- An *environment* $e = [x_1 \backslash c_1] :: \ldots :: [x_n \backslash c_n]$, which binds variables to their corresponding values.
- A *dump* D, representing a *continuation*, and abstracting the lower-level notion of *call stack*.

Furthermore, the values $c_1, \ldots, c_n$ found inside stacks and environments are sometimes not bare terms, but rather *closures*. A closure is a pair $(\overline{t}, e)$ of a code $\overline{t}$ and an enclosing environment e which should bind all the free variables of $\overline{t}$.

All of the abstract machines that we study in this chapter are deterministic. They are reasonable abstractions of the lower-level constructs that one may implement in standard hardware architectures, such as pointers and stack frames. Moreover, most of the machines that we will define in this chapter (except for the MAD in Section 3.5.6 and the Merged MAD in Section 3.5.7) are to be regarded as *established* models of sequential computation, in the sense that n transitions of an abstract machine can be simulated by Turing-machines in a number of steps polynomial in n.[2]

Third, a **distillation** is given by a decoding function $[\![\cdot]\!] : \mathsf{State} \to \mathsf{Term}$ from the set of states of the machine to the set of terms of the calculus. The decoding functions usually take the *code* $\overline{t}$ in the state of the abstract machine and they leave it verbatim. The remaining components of the state of the abstract machine (stack, environment, etc.) are combined and decoded into an evaluation context $\mathtt{E}$. The whole state of the abstract machine is then decoded as the term $\mathtt{E}\langle\overline{t}\rangle$.

One then aims for a *correctness* result, stating, roughly, that the reduction strategy simulates the abstract machine:

$$\text{if } S \rightsquigarrow_{\mathbb{M}} S' \text{ then } [\![S]\!] \twoheadrightarrow_{\mathbb{S}} [\![S']\!] \tag{3.1}$$

Note that this is not a novel idea. In fact, it is well-known that abstract machines can be seen as implementations of evaluation strategies in calculi of explicit substitutions (see at least [43, 73, 28, 103, 42]).

However, there is a difficulty that must be overcome in calculi with explicit substitutions at a distance. At first sight, reduction strategies and abstract machines compute in quite different ways. Some machine transitions, the *principal transitions*, correspond to computations and can easily be mapped to either multiplicative or exponential steps. For example, in Krivine's abstract machine [97] (KAM), the following principal transition:

term	stack	environment		term	stack	environment
$\lambda x.x$	$(y, \epsilon) :: \epsilon$	$[y \backslash (z, \epsilon)] :: \epsilon$	$\rightsquigarrow_{\mathtt{KAM}}$	x	ϵ	$[x \backslash (y, \epsilon)] :: [y \backslash (z, \epsilon)] :: \epsilon$

[2] This is justified by the *invariants* that the machines enjoy. Using these invariants, it can be seen that the abstract machines can be simulated polynomially by random-access machines, and hence also by Turing machines. We shall not give detailed proofs of these facts.

May be easily decoded as a single multiplicative step in the call-by-name reduction strategy:

$$((\lambda x.x)\, y)[x\backslash z] \to_{\mathrm{name}} x[x\backslash y][x\backslash z]$$

But abstract machines also incorporate *search transitions*, which have no direct counterpart as rewriting steps in the calculus. Let us illustrate the difficulty. To evaluate an application, some machines duplicate the environment, associating a copy of the environment to each of the two subterms. For example, in the KAM:

$$\begin{array}{cccccccc} \text{term} & \text{stack} & \text{environment} & & \text{term} & \text{stack} & \text{environment} \\ t\, s & \pi & e & \rightsquigarrow_{\mathtt{KAM}} & t & (s,e) :: \pi & e \end{array} \tag{3.2}$$

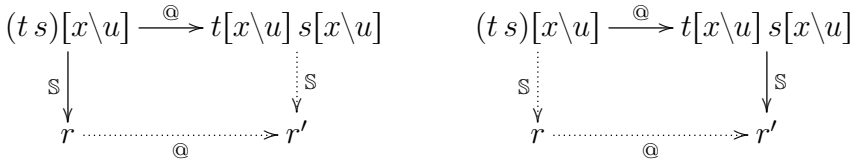
That is, to evaluate an application $t\, s$, one should go on to evaluate the function t, in the stack extended with the closure (s, e).

In the traditional approach to explicit substitutions (not "at a distance"), this corresponds to a rewriting step in the calculus, such as the following $\to_{@}$ rule:

$$(t\, s)[x\backslash u] \to_{@} t[x\backslash u]\, s[x\backslash u] \tag{3.3}$$

However, calculi with explicit substitutions at a distance reject these kinds of rules, and as a consequence the behavior of the machine in (3.2) cannot be simulated by the calculus.

The work in this chapter stems from the key observation that rules like $\to_{@}$ in (3.3)—despite not being at a distance—preserve the behavior of the strategy $\to_{\mathbb{S}}$. The intuitive reason is that the following diagrams commute. As customary, solid arrows in the diagram are universally quantified and dotted arrows are existentially quantified:



These diagrams express the fact that $\to_{@}$ is a *strong bisimulation*. Recall that:

Definition 3.1 (Strong bisimulation)**.** Let $\mathcal{A} = (A, \to_1)$ and $\mathcal{B} = (B, \to_2)$ be abstract rewriting systems. A binary relation $\sim \subseteq A \times B$ is a *strong simulation with respect to* $(\to_1, \to_2)$ if for any objects $a, a' \in A$ and $b \in B$ such that $a \to_1 a'$ and $a \sim b$ there is an object $b' \in B$ such that $b \to_2 b'$ and $a' \sim b'$. Graphically:

$$\begin{array}{ccc} a & \sim & b \\ {\scriptstyle 1}\downarrow & & \vdots\,{\scriptstyle 2} \\ a' & \sim & b' \end{array}$$

If, moreover, the inverse relation $(\sim)^{-1} \stackrel{\text{def}}{=} \{(b, a) \mid a \sim b\} \subseteq B \times A$ is also a strong simulation with respect to $(\to_2, \to_1)$, then $\sim$ is a *strong bisimulation with respect to* $(\to_1, \to_2)$.

In this chapter, we usually have that $\mathcal{A} = \mathcal{B} = (A, \to)$ so $\to_1 = \to_2 = \to$. In this setting, we simply say that $\sim$ is a *strong bisimulation with respect to* $\to$, or just a *strong bisimulation* if $\to$ is clear from the context.

In general, for each corresponding pair $(\mathbb{S}, \mathbb{M})$ of a reduction strategy $\mathbb{S}$ and an abstract machine $\mathbb{M}$ under study, we define an associated binary equivalence relation $\equiv$ of *structural equivalence* between terms. Structural equivalence includes equations to propagate explicit substitutions, such as $(t\, s)[x\backslash u] \equiv t[x\backslash u]\, s[x\backslash u]$, and it turns out to be a strong bisimulation with respect to the reduction strategy $\to_{\mathbb{S}}$.

Note that $\equiv$ being a bisimulation means that the following inclusion between relations holds:

$$(\equiv \to_{\mathbb{S}}) \subseteq (\to_{\mathbb{S}} \equiv)$$

which in turn implies that any sequence of steps:

$$t_0 \ \to_{\mathbb{S}} \equiv \ t_1 \ \to_{\mathbb{S}} \equiv \ t_2 \ldots \ \to_{\mathbb{S}} \equiv \ t_n$$

can always be rearranged as follows (by transitivity of $\equiv$):

$$t_0 \ \to_{\mathbb{S}} \ t'_1 \ \to_{\mathbb{S}} \ t'_2 \ldots \ \to_{\mathbb{S}} \ t'_n \equiv t_n$$

The desired correctness result that we stated in 3.1 is then slightly weakened to allow for propagations of explicit substitutions in the calculus:

$$\text{if } S \rightsquigarrow_{\mathbb{M}} S' \text{ then } [\![S]\!] \twoheadrightarrow_{\mathbb{S}} \equiv [\![S']\!]$$

This means that the reduction strategy simulates the abstract machine, up to propagations of explicit substitutions. Since the reduction strategy and the abstract machine are both deterministic, from such a property we will be able to deduce that the abstract machine simulates the strategy.

Note also that $\equiv$ being a strong bisimulation captures the idea that two structurally equivalent terms are *behaviorally equivalent* with respect to the strategy. In particular if $t \equiv s$ then the number of steps required to normalize t, according to the strategy $\mathbb{S}$, is the same as the number of steps required to normalize s according to $\mathbb{S}$. Consequently, calculi at a distance faithfully represent abstract machines up to propagations of explicit substitutions. The *search* transitions of the abstract machine (such as 3.2) are decoded as structurally equivalent terms (such as the left and right-hand sides of 3.3). Search transitions are thus are somehow forgotten, while principal transitions are retained and simulated as $\to_{\mathbb{S}}$ steps.

Bounding the Time Complexity of Reduction

It is natural to wonder what is lost in forgetting some of the transitions of the abstract machine. We show that *nothing is lost*, at least from a complexity point of view: any time complexity bound for strategies lifts to the corresponding machines, and vice-versa. More precisely, we give a polynomial bound for the number of $\rightsquigarrow_{\mathbb{M}}$-steps required to simulate a sequence of n consecutive $\to_{\mathbb{S}}$-steps starting from an initial pure term t_0. The specific details of the argument depend on the particular abstract machine, but the idea is as follows:

- **Multiplicative steps.** Each multiplicative step $\mathtt{E}\langle \underline{(\lambda x.t)s} \rangle \to_{\mathbb{S}} \mathtt{E}\langle t[x\backslash s] \rangle$, is simulated in the abstract machine with:

  - A number of search transitions, in order to find the underlined redex. The cost of each search transition is constant, and we will argue that there are at most $|t_0|$ such transitions.
  - A principal transition that usually rearranges the stack and creates a binding $[x\backslash s]$ in the environment. The cost of such a transition is constant.

- **Exponential steps.** Each exponential step $\mathtt{E}_1\langle \mathtt{E}_2\langle\!\langle \underline{x} \rangle\!\rangle [x\backslash t]\rangle \to_{\mathbb{S}} \mathtt{E}_1\langle \mathtt{E}_2\langle t\rangle [x\backslash t]\rangle$, is simulated in the abstract machine with:

  - A number of search transitions, in order to find the underlined variable. As before, we will argue that there are at most $|t_0|$ such transitions, each of constant cost.
  - A principal transition that makes a copy of the term t. The cost of such a transition is the cost of copying t, which is $O(|t|)$. *A priori* the size of t could be arbitrarily large, so to give a bound for this cost, it is crucial to prove that the abstract machines verify a number of invariants. One particular invariant, the *subterm property*, states that t is a subterm of the initial term t_0, and it allows us to ensure that the cost of this transition is $O(|t_0|)$.

As a consequence of this analysis, we will obtain *bilinear* bounds. That is, the number of $\rightsquigarrow_{\mathbb{M}}$-steps required to simulate a sequence of n consecutive $\to_{\mathbb{S}}$-steps starting from an initial term t_0 will be bounded by $O(|t_0| \cdot n)$.

Local vs. Global Environments — Explicit Treatment of α-Equivalence

In this chapter we study two kinds of machines: those with many *local environments* and those with just one *global environment*.

The notion of **local environment** is defined mutually inductively with the notion of *closure*:

$$\begin{array}{lrcl} \text{Local environments} & e & ::= & \epsilon \mid [x\backslash c] :: e \\ \text{Closures} & c & ::= & (\overline{t}, e) \end{array}$$

That is, a local environment maps variables to closures, and closures consist of a code $\overline{t}$ in an enclosing local environment e.

In contrast, the **global environment** is *flat*, *i.e.* it maps variables to codes and there is no nesting:

$$\text{Global environments} \quad E \quad ::= \quad \epsilon \mid [x\backslash \overline{t}] :: E$$

machines with global environments will have a single global closure $(\overline{t}, E)$.

To explicitly treat α-equivalence, we work with particular representatives of α-equivalence classes, defined via the notion of support. The *support* $\mathsf{supp}(-)$ of codes, environments, and closures is defined as follows:

- $\mathsf{supp}(\overline{t})$ is the *multiset* of its bound names (*e.g.* $\mathsf{supp}(\lambda x.\lambda y.\lambda x.(zx)) = [x, x, y]$).
- $\mathsf{supp}(e)$ is the multiset of names captured by e (*e.g.* $\mathsf{supp}([x\backslash c_1][y\backslash c_2][x\backslash c_3]) = [x, x, y]$), and similarly for $\mathsf{supp}(E)$.

- $\mathsf{supp}((\bar{t}, e)) \stackrel{\text{def}}{=} \mathsf{supp}(\bar{t}) + \mathsf{supp}(e)$ and $\mathsf{supp}((\bar{t}, E)) \stackrel{\text{def}}{=} \mathsf{supp}(\bar{t}) + \mathsf{supp}(E)$.

A code/environment/closure X is *well-named* if its support $\mathsf{supp}(X)$ is a set, *i.e.* a multiset with no repetitions. Moreover, a closure $(\bar{t}, e)$ (resp. (t, E)) is *closed* if $\mathtt{fv}(\bar{t}) \subseteq \mathsf{supp}(e)$ (resp. $\mathtt{fv}(\bar{t}) \subseteq \mathsf{supp}(E)$).

3.1.1 Our Work

This chapter is the result of collaboration with Beniamino Accattoli and Damiano Mazza, and it is structured as follows. We highlight in boldface what we consider to be the main contributions:

- In Section 3.2 we **present five reduction strategies** (Def. 3.3) using explicit substitutions at a distance. Specifically, the five reduction strategies are: (1) weak call-by-name, (2) weak call-by-value, with left-to-right evaluation, (3) weak call-by-value, with right-to-left evaluation, (4) weak call-by-need, (5) strong call-by-name.

  The first four strategies are easy to define by relying on an appropriate notion of evaluation context. These strategies are well-known from the literature and we do not claim originality, although it should be noted that this is the first presentation that uses explicit substitutions at a distance. In particular, the weak call-by-need strategy is quite simple in contrast with previous formulations [12, 113, 13, 35]—it has two reduction rules, and the grammar of evaluation contexts consists of a single sort with four straightforward productions.

  Strong call-by-name on the other hand requires more care. Our presentation follows a previous work by Accattoli and Dal Lago [11]. In Section 3.2.4 we show that strong call-by-name, defined using evaluation contexts, corresponds to *linear leftmost-outermost reduction* in the LSC [6, 11]—that is at the same time a refinement of leftmost-outermost β-reduction and an extension of linear head reduction to normal form.

  Moreover, we show that all of **these strategies are deterministic** (Prop. 3.11).

- In Section 3.3 we define a notion of *structural equivalence* $\equiv_{\mathbb{S}}$ for each reduction strategy $\mathbb{S}$ defined in Section 3.2. The main technical result is that, for each strategy $\mathbb{S}$, it turns out that **structural equivalence $\equiv_{\mathbb{S}}$ is a strong bisimulation with respect to the strategy $\mathbb{S}$** (Prop. 3.14).

- In Section 3.4 we **introduce the notion of distillery**, an abstract structure used to relate reduction strategies and abstract machines.

- In Section 3.5 we define abstract machines implementing each of the strategies, and we prove that **all of these abstract machines form distilleries for the corresponding reduction strategies**:

Strategy	Abstract Machine	
call-by-name	KAM	Section 3.5.1
	MAM	Section 3.5.2
call-by-value	CEK	Section 3.5.3
	Split CEK	Section 3.5.4
	LAM	Section 3.5.5
call-by-need	MAD	Section 3.5.6
	Merged MAD	Section 3.5.7
	Pointing MAD	Section 3.5.8
strong call-by-name	Strong MAM	Section 3.5.9

- Finally, in Section 3.6 we show that, for each of the abstract machines defined in Section 3.5, the length of an execution in the machine is **bilinearly related** with the length of the reduction sequence starting from the same initial term, in the corresponding reduction strategy.

A note on machines for strong reduction. In this chapter, the only abstract machine for strong reduction that we study—the Strong MAM (Section 3.5.9)—implements strong call-by-name. Machines for other strong strategies, such as *strong call-by-value* and *strong call-by-need*, are more complex—in fact defining the strong reduction strategies is itself a nontrivial task. Abstract machines for *open* call-by-value (*i.e.* allowing the presence of free variables but disallowing evaluation below abstractions) following the spirit of this chapter have been studied by Accattoli and Guerrieri [7]; Grégoire and Leroy [66] also study strong call-by-value, defined by iterating a weak call-by-value strategy. In the following chapter (Chapter 4) we study a strong call-by-need strategy. In Section 8.1 in the Conclusion (Chapter 8), we propose an abstract machine for strong call-by-need evaluation, although we do not study its properties.

3.2 Reduction Strategies

In this section we define five deterministic reduction strategies: call-by-name ($\mathtt{name}$), two variants of call-by-value ($\mathtt{value}^{\mathtt{LR}}$, $\mathtt{value}^{\mathtt{RL}}$), call-by-need ($\mathtt{need}$), and strong call-by-name ($\mathtt{name}^{\mathtt{S}}$). Moreover, in Section 3.2.5 we prove that all of these strategies are deterministic.

Definition 3.2 (Root rewriting rules)**.** As mentioned, the set of *terms* is given as usual for the LSC (*cf.* Def. 2.74) by the grammar $t ::= x \mid \lambda x.t \mid ts \mid t[x\backslash s]$, *values* are given by $\mathtt{v} ::= \lambda x.t$, and *substitution contexts* are given by $\mathtt{L} ::= \Box \mid \mathtt{L}[x\backslash t]$. A term of the form $\mathtt{vL}$ is called an *answer*.

Given a fixed family of *evaluation contexts* ranged over by $\mathtt{E}, \mathtt{E}', \ldots$ we define the following four *root rewriting rules*—two $\mathtt{db}$-like rules and two $\mathtt{ls}$-like rules:

$$\begin{array}{rcl}
(\lambda x.t)\mathtt{L}\, s & \mapsto_{\mathtt{db}} & t[x\backslash s]\mathtt{L} \\
(\lambda x.t)\mathtt{L}\, \mathtt{vL}' & \mapsto_{\mathtt{dbv}} & t[x\backslash \mathtt{vL}']\mathtt{L} \\
\mathtt{E}\langle\!\langle x \rangle\!\rangle[x\backslash s] & \mapsto_{\mathtt{ls}} & \mathtt{E}\langle s \rangle[x\backslash s] \\
\mathtt{E}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}] & \mapsto_{\mathtt{lsv}} & \mathtt{E}\langle \mathtt{v} \rangle[x\backslash \mathtt{v}]\mathtt{L}
\end{array}$$

In the rules suffixed with a "v", the argument of the application/substitution is expected to be an answer. Moreover, we use the notations $\overset{\mathtt{E}}{\mapsto}_{\mathtt{ls}}$ and $\overset{\mathtt{E}}{\mapsto}_{\mathtt{lsv}}$ to specify the family of contexts used by the rules, with $\mathtt{E}$ being the meta-variable ranging over such contexts.

A reduction strategy is specified by a **choice of root rules**, *i.e.* one multiplicative rule (db or dbv) and one exponential rule (ls or lsv), and a **family of evaluation contexts**. The chosen multiplicative (resp. exponential) rule is generically denoted by $\mapsto_{\mathtt{m}}$ (resp. $\mapsto_{\mathtt{e}}$). If $\mathtt{E}$ ranges over a fixed notion of evaluation context, the contextual closures of the root rules are denoted by $\to_{\mathtt{m}} \overset{\text{def}}{=} \mathtt{E}\langle\mapsto_{\mathtt{m}}\rangle$ and $\to_{\mathtt{e}} \overset{\text{def}}{=} \mathtt{E}\langle\mapsto_{\mathtt{e}}\rangle$. The rewriting relation defining the reduction strategy is then $\to \overset{\text{def}}{=} \to_{\mathtt{m}} \cup \to_{\mathtt{e}}$.

Definition 3.3 (The reduction strategies name, $\mathtt{value}^{\mathtt{LR}}$, $\mathtt{value}^{\mathtt{RL}}$, need, $\mathtt{name}^{\mathtt{S}}$)**.** The reduction strategies *call-by-name* (name), *left-to-right call-by-value* ($\mathtt{value}^{\mathtt{LR}}$), *right-to-left call-by-value* ($\mathtt{value}^{\mathtt{RL}}$), *call-by-need* (need), and *strong call-by-name* ($\mathtt{name}^{\mathtt{S}}$), are specified by the following choices of root reduction rules and evaluation contexts:

Strategy	Evaluation contexts	$\mapsto_{\mathtt{m}}$	$\mapsto_{\mathtt{e}}$	$\to_{\mathtt{m}}$	$\to_{\mathtt{e}}$
name	$\mathtt{H} ::= \Box \mid \mathtt{H}\,t \mid \mathtt{H}[x\backslash t]$	$\mapsto_{\mathtt{db}}$	$\overset{\mathtt{H}}{\mapsto}_{\mathtt{ls}}$	$\mathtt{H}\langle\mapsto_{\mathtt{db}}\rangle$	$\mathtt{H}\langle\overset{\mathtt{H}}{\mapsto}_{\mathtt{ls}}\rangle$
$\mathtt{value}^{\mathtt{LR}}$	$\mathtt{V} ::= \Box \mid \mathtt{V}\,t \mid \mathtt{vL}\,\mathtt{V} \mid \mathtt{V}[x\backslash t]$	$\mapsto_{\mathtt{dbv}}$	$\overset{\mathtt{V}}{\mapsto}_{\mathtt{lsv}}$	$\mathtt{V}\langle\mapsto_{\mathtt{dbv}}\rangle$	$\mathtt{V}\langle\overset{\mathtt{V}}{\mapsto}_{\mathtt{lsv}}\rangle$
$\mathtt{value}^{\mathtt{RL}}$	$\mathtt{R} ::= \Box \mid \mathtt{R}\,\mathtt{vL} \mid t\,\mathtt{R} \mid \mathtt{R}[x\backslash t]$	$\mapsto_{\mathtt{dbv}}$	$\overset{\mathtt{R}}{\mapsto}_{\mathtt{lsv}}$	$\mathtt{R}\langle\mapsto_{\mathtt{dbv}}\rangle$	$\mathtt{R}\langle\overset{\mathtt{R}}{\mapsto}_{\mathtt{lsv}}\rangle$
need	$\mathtt{N} ::= \Box \mid \mathtt{N}\,t \mid \mathtt{N}[x\backslash t] \mid \mathtt{N}'\langle x\rangle[x\backslash \mathtt{N}]$	$\mapsto_{\mathtt{db}}$	$\overset{\mathtt{N}}{\mapsto}_{\mathtt{lsv}}$	$\mathtt{N}\langle\mapsto_{\mathtt{db}}\rangle$	$\mathtt{N}\langle\overset{\mathtt{N}}{\mapsto}_{\mathtt{lsv}}\rangle$
$\mathtt{name}^{\mathtt{S}}$	$\mathtt{S} ::= (\mathcal{S}$ contexts, see Def. 3.5)	$\mapsto_{\mathtt{db}}$	$\overset{\mathtt{S}}{\mapsto}_{\mathtt{ls}}$	$\mathtt{S}\langle\mapsto_{\mathtt{db}}\rangle$	$\mathtt{S}\langle\overset{\mathtt{S}}{\mapsto}_{\mathtt{ls}}\rangle$

3.2.1 Call-by-Name

The call-by-name strategy uses the $\mapsto_{\mathtt{db}}$ and $\mapsto_{\mathtt{ls}}$ root reduction rules, *i.e.* it never evaluates arguments. Evaluation contexts $\mathtt{H}$ are sometimes called *head contexts*. Evaluation always focuses on the left-hand side of applications, until the head becomes an answer $\mathtt{vL}$. If there are any arguments remaining, a db-step may be fired. The following is an example of a call-by-name reduction; on each step, the contracted redex is underlined:

$$\begin{array}{rll}
\underline{(\lambda x.x\,x)((\lambda y.y)(\lambda z.f\,z))} & \to_{\mathtt{name}} & (\underline{x}\,x)[x\backslash(\lambda y.y)\,(\lambda z.f\,z)] \\
& \to_{\mathtt{name}} & (\underline{(\lambda y.y)\,(\lambda z.f\,z)}\,x)[x\backslash(\lambda y.y)(\lambda z.f\,z)] \\
& \to_{\mathtt{name}} & (\underline{y}[y\backslash\lambda z.f\,z]\,x)[x\backslash(\lambda y.y)(\lambda z.f\,z)] \\
& \to_{\mathtt{name}} & (\underline{(\lambda z.f\,z)[y\backslash\lambda z.f\,z]\,x})[x\backslash(\lambda y.y)(\lambda z.f\,z)] \\
& \to_{\mathtt{name}} & (f\,z)[z\backslash x][y\backslash\lambda z.f\,z][x\backslash(\lambda y.y)(\lambda z.f\,z)]
\end{array}$$

Observe that call-by-name is a weak reduction strategy, so the result is not a normal form in the LSC. This is not only because there are some gc-redexes—there are also ls-redexes (*e.g.* in the LSC rules are closed by arbitrary contexts so z may be substituted by x).

3.2.2 Call-by-Value

We work with two variants of call-by-value. Both of them use the $\mapsto_{\mathtt{dbv}}$ and $\mapsto_{\mathtt{lsv}}$ root reduction rules, *i.e.* the arguments must always be evaluated before going on. The two variants,

left-to-right call-by-value and *right-to-left call-by-value* differ on the evaluation contexts. Left-to-right call-by-value evaluates the function before evaluating the argument—the production $\mathtt{V} ::= \mathtt{vL}\,\mathtt{V}$ requires that the function is an answer. Right-to-left call-by-value evaluates the argument before evaluating the function—the production $\mathtt{R} ::= \mathtt{R}\,\mathtt{vL}$ requires that the argument is an answer. For example, the following is a left-to-right call-by-value reduction:

$$\begin{array}{rll}
(\lambda x.x\,x)(\underline{(\lambda y.y)(\lambda z.f\ z)}) & \to_{\mathtt{value}^{\mathtt{LR}}} & (\lambda x.x\,x)\,\underline{y}[y\backslash\lambda z.f\ z] \\
& \to_{\mathtt{value}^{\mathtt{LR}}} & \underline{(\lambda x.x\,x)\,(\lambda z.f\ z)[y\backslash\lambda z.f\ z]} \\
& \to_{\mathtt{value}^{\mathtt{LR}}} & (\underline{x}\,x)[x\backslash(\lambda z.f\ z)[y\backslash\lambda z.f\ z]] \\
& \to_{\mathtt{value}^{\mathtt{LR}}} & ((\lambda z.f\ z)\,\underline{x})[x\backslash\lambda z.f\ z][y\backslash\lambda z.f\ z] \\
& \to_{\mathtt{value}^{\mathtt{LR}}} & (\underline{(\lambda z.f\ z)\,(\lambda z.f\ z)})[x\backslash\lambda z.f\ z][y\backslash\lambda z.f\ z] \\
& \to_{\mathtt{value}^{\mathtt{LR}}} & (f\ z)[z\backslash\lambda z.f\ z][x\backslash\lambda z.f\ z][y\backslash\lambda z.f\ z]
\end{array}$$

while the following is a right-to-left call-by-value reduction—it differs from left-to-right call-by-value only in the steps marked with $(\star)$:

$$\begin{array}{rll}
(\lambda x.x\,x)(\underline{(\lambda y.y)(\lambda z.f\ z)}) & \to_{\mathtt{value}^{\mathtt{RL}}} & (\lambda x.x\,x)\,\underline{y}[y\backslash\lambda z.f\ z] \\
& \to_{\mathtt{value}^{\mathtt{RL}}} & \underline{(\lambda x.x\,x)\,(\lambda z.f\ z)[y\backslash\lambda z.f\ z]} \\
& \to_{\mathtt{value}^{\mathtt{RL}}} & (x\,\underline{x})[x\backslash(\lambda z.f\ z)[y\backslash\lambda z.f\ z]] \qquad (\star) \\
& \to_{\mathtt{value}^{\mathtt{RL}}} & (\underline{x}\,(\lambda z.f\ z))[x\backslash\lambda z.f\ z][y\backslash\lambda z.f\ z] \qquad (\star) \\
& \to_{\mathtt{value}^{\mathtt{RL}}} & (\underline{(\lambda z.f\ z)\,(\lambda z.f\ z)})[x\backslash\lambda z.f\ z][y\backslash\lambda z.f\ z] \\
& \to_{\mathtt{value}^{\mathtt{RL}}} & (f\ z)[z\backslash\lambda z.f\ z][x\backslash\lambda z.f\ z][y\backslash\lambda z.f\ z]
\end{array}$$

Both variants of call-by-value are also weak reduction strategies.

3.2.3 Call-by-Need

Call-by-need uses the $\mapsto_{\mathtt{db}}$ root reduction rule. This means that the argument to a function is not evaluated: the formal parameter becomes directly bound to the unevaluated argument. This has the effect of delaying evaluation of the argument until it is *needed*, just as in call-by-name. However, call-by-need uses the $\mapsto_{\mathtt{lsv}}$ root reduction rule, which means that a variable may only be substituted for a value. As a consequence, only values may ever be copied, ensuring that the evaluation of the argument is *shared*.

The most significant change is in the definition of evaluation contexts. These are similar to head contexts in call-by-name, but they include a production $\mathtt{N} ::= \mathtt{N}'\langle x\rangle[x\backslash\mathtt{N}]$. This production means that, when evaluation focuses on a variable and the variable is not an answer yet, evaluation should proceed in the shared argument, *inside* the explicit substitution. The following is an example of a call-by-need reduction:

$$\begin{array}{rll}
\underline{(\lambda x.x\,x)((\lambda y.y)(\lambda z.f\ z))} & \to_{\mathtt{need}} & (\boxed{x}\,x)[x\backslash\underline{(\lambda y.y)(\lambda z.f\ z)}] \\
& \to_{\mathtt{need}} & (\boxed{x}\,x)[x\backslash\underline{y}[y\backslash\lambda z.f\ z]] \\
& \to_{\mathtt{need}} & (\underline{x}\,x)[x\backslash(\lambda z.f\ z)[y\backslash\lambda z.f\ z]] \\
& \to_{\mathtt{need}} & (\underline{(\lambda z.f\ z)\,x})[x\backslash\lambda z.f\ z][y\backslash\lambda z.f\ z] \\
& \to_{\mathtt{need}} & (f\ z)[z\backslash x][x\backslash\lambda z.f\ z][y\backslash\lambda z.f\ z]
\end{array}$$

Note that we underline the redex being contracted. Moreover, a variable inside a box represents the fact that that the variable is the current focus of evaluation and triggers the evaluation of the expression to which it is bound (using the production $\mathtt{N} ::= \mathtt{N}'\langle x\rangle[x\backslash\mathtt{N}]$). Like call-by-name and call-by-value, call-by-need is also a weak reduction strategy (*strong* call-by-need reduction is the topic of Chapters 4 and 5).

3.2.4 Strong Call-by-Name

Strong call-by-name is the only *strong* reduction strategy that we study in this chapter. To complete the definition of strong call-by-name, we still must give a definition for the family of evaluation contexts ($\mathtt{S}, \mathtt{S}', \ldots$). First we need the notion of *left free variables* of a context, *i.e.* the set of variables occurring free at the left of the hole:

Definition 3.4 (Left Free Variables)**.** The set $\mathtt{lfv}(\mathtt{C})$ of *left free variables* of $\mathtt{C}$ is defined by:

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{lfv}(\Box) & \stackrel{\text{def}}{=} & \varnothing & \mathtt{lfv}(t\mathtt{C}) & \stackrel{\text{def}}{=} & \mathtt{fv}(t) \cup \mathtt{lfv}(\mathtt{C}) \\
\mathtt{lfv}(\lambda x.\mathtt{C}) & \stackrel{\text{def}}{=} & \mathtt{lfv}(\mathtt{C})\backslash\{x\} & \mathtt{lfv}(\mathtt{C}[x\backslash t]) & \stackrel{\text{def}}{=} & \mathtt{lfv}(\mathtt{C})\backslash\{x\} \\
\mathtt{lfv}(\mathtt{C}t) & \stackrel{\text{def}}{=} & \mathtt{lfv}(\mathtt{C}) & \mathtt{lfv}(t[x\backslash\mathtt{C}]) & \stackrel{\text{def}}{=} & (\mathtt{fv}(t)\backslash\{x\}) \cup \mathtt{lfv}(\mathtt{C})
\end{array}$$

Definition 3.5 (Strong call-by-name evaluation contexts)**.** A term is *neutral* if it is $\to_{\mathtt{db}\cup\mathtt{ls}}$-normal in the LSC and it is not of the form $(\lambda x.t)\mathtt{L}$. A context $\mathtt{C}$ is a *strong call-by-name evaluation context* if the judgment "$\mathtt{C} \in \mathcal{S}$" can be derived using the following inductive rules:

$$\frac{}{\Box \in \mathcal{S}}\ (\text{AX-}\mathcal{S}) \qquad \frac{\mathtt{C} \in \mathcal{S} \quad \mathtt{C} \neq (\lambda x.\mathtt{C}')\mathtt{L}}{\mathtt{C}\,t \in \mathcal{S}}\ (@\text{L-}\mathcal{S})$$

$$\frac{\mathtt{C} \in \mathcal{S}}{\lambda x.\mathtt{C} \in \mathcal{S}}\ (\lambda\text{-}\mathcal{S}) \qquad \frac{t \text{ is neutral} \quad \mathtt{C} \in \mathcal{S}}{t\,\mathtt{C} \in \mathcal{S}}\ (@\text{R-}\mathcal{S})$$

$$\frac{\mathtt{C} \in \mathcal{S} \quad x \notin \mathtt{lfv}(\mathtt{C})}{\mathtt{C}[x\backslash t] \in \mathcal{S}}\ (\text{ES-}\mathcal{S})$$

Note that neutral terms are such that plugging them into a context cannot create a $\mathtt{db}$ redex. Below, Def. 3.9 gives an alternative definition for strong call-by-name evaluation contexts and Lem. 3.10 shows that these definitions are indeed equivalent.

The strong call-by-name strategy uses the $\mathtt{db}$ and $\mathtt{ls}$ root reduction rules, just as call-by-name. But the set of strong call-by-name evaluation contexts ($\mathtt{S}, \mathtt{S}', \ldots$) *generalize* the set of head contexts used in (weak) call-by-name ($\mathtt{H}, \mathtt{H}', \ldots$). Indeed, it may be easily checked by induction on $\mathtt{H}$ that any head context is also a strong call-by-name evaluation context.

In contrast with weak call-by-name, strong call-by-name performs evaluation below abstractions ($\lambda x.\Box$), as attested by rule λ-$\mathcal{S}$, as long as the abstraction is not applied. Moreover, strong call-by-name performs evaluation on the arguments of applications $t\,\Box$ as long as t is a *neutral term*. Neutral terms should be thought as terms of the form $x\,t_1 \ldots t_n$, sprinkled

with unreachable explicit substitutions (*i.e.* terms whose $\to_{\ell\ \mathsf{gc}}$ -normal form is of the form $x\,t_1 \dots t_n$).

The following is an example of a reduction in strong call-by-name. Observe that the (weak) call-by-name reduction is a *prefix* of the strong call-by-name reduction. The first properly *strong* step is marked with $(\star)$:

$$
\begin{array}{rll}
\underline{(\lambda x.x\,x)((\lambda y.y)(\lambda z.\lambda f.f\,z))} & \to_{\mathtt{name^s}} & (\underline{x}\,x)[x\backslash(\lambda y.y)\,(\lambda z.\lambda f.f\,z)] \\
& \to_{\mathtt{name^s}} & (\underline{(\lambda y.y)\,(\lambda z.\lambda f.f\,z)}\,x)[x\backslash(\lambda y.y)(\lambda z.\lambda f.f\,z)] \\
& \to_{\mathtt{name^s}} & (\underline{y[y\backslash\lambda z.\lambda f.f\,z]}\,x)[x\backslash(\lambda y.y)(\lambda z.\lambda f.f\,z)] \\
& \to_{\mathtt{name^s}} & (\underline{(\lambda z.\lambda f.f\,z)[y\backslash\lambda z.\lambda f.f\,z]\,x})[x\backslash(\lambda y.y)(\lambda z.\lambda f.f\,z)] \\
& \to_{\mathtt{name^s}} & (\lambda f.f\,\underline{z})[z\backslash x][y\backslash\lambda z.\lambda f.f\,z][x\backslash(\lambda y.y)(\lambda z.\lambda f.f\,z)] \\
& \to_{\mathtt{name^s}} & (\lambda f.f\,\underline{x})[z\backslash x][y\backslash\lambda z.\lambda f.f\,z][x\backslash(\lambda y.y)(\lambda z.\lambda f.f\,z)] \quad (\star) \\
& \to_{\mathtt{name^s}} & (\lambda f.f\,(\underline{(\lambda y.y)(\lambda z.\lambda f.f\,z)}))[z\backslash x][y\backslash\lambda z.\lambda f.f\,z][x\backslash(\lambda y.y)(\lambda z.\lambda f.f\,z)] \\
& \to_{\mathtt{name^s}} & (\lambda f.f\,\underline{y[y\backslash\lambda z.\lambda f.f\,z]})[z\backslash x][y\backslash\lambda z.\lambda f.f\,z][x\backslash(\lambda y.y)(\lambda z.\lambda f.f\,z)] \\
& \to_{\mathtt{name^s}} & (\lambda f.f\,(\lambda z.\lambda f.f\,z)[y\backslash\lambda z.\lambda f.f\,z])[z\backslash x][y\backslash\lambda z.\lambda f.f\,z][x\backslash(\lambda y.y)(\lambda z.\lambda f.f\,z)]
\end{array}
$$

Alternative Characterization of Strong Call-by-Name

Reduction according to the strong call-by-name strategy, can be characterized exactly as *linear leftmost-outermost reduction* $\to_{\mathsf{LO}}$. To define $\to_{\mathsf{LO}}$ we need a few previous definitions:

Definition 3.6 (LO order)**.** We write $\mathtt{C} \prec_p t$ if there is a term s such that $\mathtt{C}\langle s\rangle = t$. This is called the *prefix relation*.

The *outside-in order* $\mathtt{C} \prec_O \mathtt{C}'$ between arbitrary contexts $\mathtt{C}, \mathtt{C}'$ is defined by the following rules:

1. *Root*: $\Box \prec_O \mathtt{C}$ for every context $\mathtt{C} \neq \Box$.
2. *Contextual closure*: if $\mathtt{C} \prec_O \mathtt{C}'$ then $\mathtt{C}''\langle\mathtt{C}\rangle \prec_O \mathtt{C}''\langle\mathtt{C}'\rangle$ for any context $\mathtt{C}''$.

Note that $\prec_O$ can be seen as the prefix relation $\prec_p$ on contexts. The *left-to-right order* $\mathtt{C} \prec_L \mathtt{C}'$ is defined by:

1. *Application*: if $\mathtt{C} \prec_p t$ and $\mathtt{C}' \prec_p s$ then $\mathtt{C}\,s \prec_L t\,\mathtt{C}'$.
2. *Substitution*: if $\mathtt{C} \prec_p t$ and $\mathtt{C}' \prec_p s$ then $\mathtt{C}[x\backslash s] \prec_L t[x\backslash\mathtt{C}']$.
3. *Contextual closure*: if $\mathtt{C} \prec_L \mathtt{C}'$ then $\mathtt{C}''\langle\mathtt{C}\rangle \prec_L \mathtt{C}''\langle\mathtt{C}'\rangle$ for any context $\mathtt{C}''$.

Finally, the *left-to-right outside-in order* is defined by $\mathtt{C} \prec_{\mathsf{LO}} \mathtt{C}'$ if $\mathtt{C} \prec_O \mathtt{C}'$ or $\mathtt{C} \prec_L \mathtt{C}'$.

Two examples of the outside-in order are $(\lambda x.\Box)t \prec_O (\lambda x.(\Box[y\backslash s]))t$ and $t[x\backslash\Box] \prec_O t[x\backslash s\mathtt{C}]$, and an example of the left-to-right order is $t[x\backslash\mathtt{C}]s \prec_L t[x\backslash u]\Box$, where the terms t, s, u and the context $\mathtt{C}$ are arbitrary. The following lemma guarantees that it is a total order.

Lemma 3.7 (Totality of $\prec_{\mathsf{LO}}$)**.** *If* $\mathtt{C} \prec_p t$ *and* $\mathtt{C}' \prec_p t$ *then either* $\mathtt{C} \prec_{\mathsf{LO}} \mathtt{C}'$ *or* $\mathtt{C}' \prec_{\mathsf{LO}} \mathtt{C}$ *or* $\mathtt{C} = \mathtt{C}'$.

Proof. Straightforward by induction on t. □

We identify redexes with the context that focuses on the *anchor*. Recall from Def. 2.75 that the anchor of a db-step is the contracted application, and the anchor of a ls-step is the contracted variable. We can now define linear LO reduction, first considered in [6], where it is proved that it is standard and normalizing, and then in [11], extending linear head reduction [115, 47, 3] to normal form.

Definition 3.8 (Linear LO Reduction $\to_{\text{LO}}$)**.** Let t be a term. A redex $\mathtt{C}$ is the *leftmost-outermost* (LO for short) redex of t if $\mathtt{C} \prec_{\mathsf{LO}} \mathtt{C}'$ for every other redex $\mathtt{C}'$ of t. We write $t \to_{\text{LO}} s$ for a step contracting the leftmost-outermost redex.

We now define LO contexts and prove that the position of a linear LO step is always a LO context:

Definition 3.9 (LO Contexts)**.** A context $\mathtt{C}$ is LO if:

1. *Right Application*: whenever $\mathtt{C} = \mathtt{C}'\langle t\, \mathtt{C}''\rangle$ then t is neutral.
2. *Left Application*: whenever $\mathtt{C} = \mathtt{C}'\langle \mathtt{C}'' t\rangle$ then $\mathtt{C}'' \neq \mathtt{L}\langle \lambda x.\mathtt{C}'''\rangle$.
3. *Substitution*: whenever $\mathtt{C} = \mathtt{C}'\langle \mathtt{C}''[x\backslash s]\rangle$ then $x \notin \mathtt{lfv}(\mathtt{C}'')$.

Lemma 3.10 (Characterization of LO contexts)**.**

1. *Let $\mathtt{C}$ be a context. Then $\mathtt{C} \in \mathcal{S}$ if and only if $\mathtt{C}$ is* LO.
2. *Let $t \to s$ by reducing a redex under a context $\mathtt{C}$. Then $\mathtt{C}$ is a $\to_{LO}$ step if and only if $\mathtt{C}$ is* LO.

Proof. The first item is an immediate induction on $\mathtt{C}$. For the second item, we prove each direction of the equivalence. ($\Rightarrow$) There are three cases:

1. *Left application*: if $\mathtt{C} = \mathtt{C}'\langle \mathtt{C}'' t\rangle$ then clearly $\mathtt{C}'' \neq \mathtt{L}\langle \lambda x.\mathtt{C}'''\rangle$, otherwise $\mathtt{C}$ is not the position of the LO redex.
2. *Right Application*: let $\mathtt{C} = \mathtt{C}'\langle u\mathtt{C}''\rangle$, and note u is neutral otherwise $\mathtt{C}$ is not the position of the LO redex.
3. *Substitution*: if $\mathtt{C} = \mathtt{C}'\langle \mathtt{C}''[x\backslash s]\rangle$ then $x \notin \mathtt{lfv}(\mathtt{C}'')$ otherwise there is an exponential redex of position $\prec_{\mathsf{LO}} \mathtt{C}$, which would be absurd.

($\Leftarrow$) Let $\mathtt{C}'$ the position of the step in t and suppose that $\mathtt{C}' \neq \mathtt{C}$. By definition $\mathtt{C}' \prec_{\mathsf{LO}} \mathtt{C}$. We have two cases:

1. $\mathtt{C}' \prec_O \mathtt{C}$. Then necessarily $\mathtt{C}'$ identifies a db-redex and we have $\mathtt{C} = \mathtt{C}'\langle \mathtt{L}\langle \lambda x.\mathtt{C}''\rangle u\rangle$. It follows that $\mathtt{C}$ is not a LO context, because this contradicts the left application clause.
2. $\mathtt{C}' \prec_L \mathtt{C}$. Then there is a decomposition $\mathtt{C} = \mathtt{C}''\langle u\mathtt{C}'''\rangle$ with the hole of $\mathtt{C}'$ falling in u. By hypothesis u is neutral. Then $u = \mathtt{C}_0\langle x\rangle$ and the $\to_{\text{LO}}$ step is a ls-step substituting on x from a substitution in $\mathtt{C}''$, *i.e.* $\mathtt{C}'' = \mathtt{C}^\bullet\langle \mathtt{C}^\circ[x\backslash t]\rangle$ for some contexts $\mathtt{C}^\bullet$ and $\mathtt{C}^\circ$. Then $\mathtt{C} = \mathtt{C}^\bullet\langle \mathtt{C}^\circ\langle u\mathtt{C}'''\rangle[x\backslash t]\rangle$ and $x \in \mathtt{lfv}(\mathtt{C}^\circ\langle u\mathtt{C}'''\rangle)$, which contradicts the substitution clause in the hypothesis that $\mathtt{C}$ is a LO context.

□

3.2.5 Determinism

All the reduction strategies studied in this chapter are deterministic. Recall that the *anchor* of a multiplicative step is the contracted application, and the anchor of an exponential step is the contracted variable. Then:

Proposition 3.11 (Determinism — ♣ Prop. A.1)**.** *The five reduction strategies of Def. 3.3 are deterministic. In each case, if* $\mathtt{E}_1, \mathtt{E}_2$ *are evaluation contexts,* r_1, r_2 *are* anchors*, and* $\mathtt{E}_1\langle r_1\rangle = \mathtt{E}_2\langle r_2\rangle$ *then* $\mathtt{E}_1 = \mathtt{E}_2$ *and* $r_1 = r_2$*. So there is at most one way to reduce a term.*

Proof. See Prop. A.1 in the appendix for the detailed proofs. The proofs for the call-by-name, call-by-value, and call-by-need cases are by induction on the structure of the terms, verifying that there may be at most one redex under an evaluation context. The proof for the strong call-by-name case is easily derived from the fact that strong call-by-name reduction is precisely leftmost-outermost reduction (Lem. 3.10). □

3.3 Structural Equivalences

Each of the five reduction strategies $\mathbb{S} \in \{\mathtt{name}, \mathtt{value}^{\mathtt{LR}}, \mathtt{value}^{\mathtt{RL}}, \mathtt{need}, \mathtt{name}^{\mathtt{S}}\}$ presented so far comes equipped with a corresponding notion of *structural equivalence*, denoted by $\equiv_{\mathbb{S}}$. Structural equivalence allows manipulating explicit substitutions, moving them around in a *computationally irrelevant* way. Technically, this is expressed by the property that structural equivalence is a *strong bisimulation* (*cf.* Def. 3.1).

Certain ways of moving substitutions around are allowed in some strategies and not in other ones. For instance, the equivalence:

$$(t\, s)[x\backslash u] \equiv_{@} t[x\backslash u]\, s[x\backslash u]$$

is sound in call-by-name, *i.e.* the term on the left and the term on the right are in fact *strongly bisimilar* with respect to $\to_{\mathtt{name}}$, whereas it is not sound in call-by-need. The reason is that call-by-need evaluates inside some substitutions (those that hereditarily bind a head variable), so the term on the left may evaluate u at most once, and the term on the right may evaluate it twice. For our purposes in this chapter, it suffices to show that each structural equivalence is a strong bisimulation. A deeper explanation of why some propagations of explicit substitutions are unsound may be found in the translation of these strategies into linear logic proof nets: substitutions may move freely as long as they do not cross the boundaries of *boxes*.

Each structural equivalence is given by choosing some of the following axioms:

Definition 3.12 (Axioms for structural equivalences)**.**

$$\begin{array}{rlll}
(\lambda x.t)[y\backslash s] & \equiv_{\lambda} & \lambda x.t[y\backslash s] & \text{if } x \notin \mathtt{fv}(s) \\
(t\,u)[x\backslash s] & \equiv_{@} & t[x\backslash s]u[x\backslash s] & \\
(t\,u)[x\backslash s] & \equiv_{@\mathtt{l}} & t[x\backslash s]\,u & \text{if } x \notin \mathtt{fv}(u) \\
(t\,u)[x\backslash s] & \equiv_{@\mathtt{r}} & t\,u[x\backslash s] & \text{if } x \notin \mathtt{fv}(t) \\
t[x\backslash s][y\backslash u] & \equiv_{\mathtt{com}} & t[y\backslash u][x\backslash s] & \text{if } y \notin \mathtt{fv}(s) \text{ and } x \notin \mathtt{fv}(u) \\
t[x\backslash s][y\backslash u] & \equiv_{[\cdot]} & t[x\backslash s[y\backslash u]] & \text{if } y \notin \mathtt{fv}(t) \\
t[x\backslash s] & \equiv_{\mathtt{gc}} & t & \text{if } x \notin \mathtt{fv}(t) \\
t[x\backslash s] & \equiv_{\mathtt{dup}} & t_{[y]_x}[x\backslash s][y\backslash s] &
\end{array}$$

In the $\equiv_{\mathtt{dup}}$ rule, $t_{[y]_x}$ denotes a term obtained from t by renaming some (possibly none) occurrences of x as y.

Definition 3.13 (Structural equivalences)**.** For each strategy $\mathbb{S}$, we select a subset of the structural equivalence axioms, and a family of contexts, as follows:

Strategy	Structural equivalence axioms	Family of contexts
$\mathtt{name}$	$\equiv_{@}, \equiv_{\mathtt{com}}, \equiv_{[\cdot]}, \equiv_{\mathtt{gc}}, \equiv_{\mathtt{dup}}$	$\mathtt{H}$
$\mathtt{value}^{\mathtt{LR}}$	$\equiv_{@}, \equiv_{\mathtt{com}}, \equiv_{[\cdot]}, \equiv_{\mathtt{gc}}, \equiv_{\mathtt{dup}}$	$\mathtt{V}$
$\mathtt{value}^{\mathtt{RL}}$	$\equiv_{@}, \equiv_{\mathtt{com}}, \equiv_{[\cdot]}, \equiv_{\mathtt{gc}}, \equiv_{\mathtt{dup}}$	$\mathtt{R}$
$\mathtt{need}$	$\equiv_{@\mathtt{l}}, \equiv_{\mathtt{com}}, \equiv_{[\cdot]}$	$\mathtt{N}$
$\mathtt{name}^{\mathtt{S}}$	$\equiv_{\lambda}, \equiv_{@\mathtt{l}}, \equiv_{@\mathtt{r}}, \equiv_{\mathtt{com}}, \equiv_{[\cdot]}, \equiv_{\mathtt{gc}}, \equiv_{\mathtt{dup}}$	$\mathtt{C}$ (arbitrary contexts)

The corresponding structural equivalence $\equiv_{\mathbb{S}}$ is defined as the reflexive, symmetric, transitive, and contextual closure of the axioms, under the specified family of contexts.

Note that the structural equivalences for call-by-name and call-by-value use the same axioms but closed under their respective notions of evaluation context. The structural equivalence for strong call-by-name is closed under arbitrary contexts. For example:

$$\begin{array}{rlll}
((\lambda x.x)y)[x\backslash x'][y\backslash y'] & \equiv_{\mathtt{value}^{\mathtt{LR}}} & ((\lambda x.x)y)[y\backslash y'][x\backslash x'] & (\text{by } \equiv_{\mathtt{com}}) \\
& \equiv_{\mathtt{value}^{\mathtt{LR}}} & ((\lambda x.x)[y\backslash y']\,y[y\backslash y'])[x\backslash x'] & (\text{by } \equiv_{@}) \\
& \equiv_{\mathtt{value}^{\mathtt{LR}}} & ((\lambda x.x)\,y[y\backslash y'])[x\backslash x'] & (\text{by } \equiv_{\mathtt{gc}})
\end{array}$$

and:

$$\begin{array}{rlll}
(\lambda x.y\,y)[y\backslash z] & \equiv_{\mathtt{name}^{\mathtt{S}}} & \lambda x.(y\,y)[y\backslash z] & (\text{by } \equiv_{\lambda}) \\
& \equiv_{\mathtt{name}^{\mathtt{S}}} & \lambda x.(y_1\,y_2)[y_1\backslash z][y_2\backslash z] & (\text{by } \equiv_{\mathtt{dup}}) \\
& \equiv_{\mathtt{name}^{\mathtt{S}}} & \lambda x.(y_1[y_1\backslash z]\,y_2)[y_2\backslash z] & (\text{by } \equiv_{@\mathtt{l}}) \\
& \equiv_{\mathtt{name}^{\mathtt{S}}} & \lambda x.y_1[y_1\backslash z]\,y_2[y_2\backslash z] & (\text{by } \equiv_{@\mathtt{r}})
\end{array}$$

Let $\to_{\mathtt{m}}$ (resp. $\to_{\mathtt{e}}$) denote the multiplicative (resp. exponential) reduction relation of any of the strategies $\mathbb{S}$ defined in Def. 3.3, and let $\equiv_{\mathbb{S}}$ be the structural equivalence relation of $\mathbb{S}$. The key result is the following:

Proposition 3.14 (Structural equivalence is a strong bisimulation — ♣ Prop. A.5)**.** *Let* $\mathtt{x} \in \{\mathtt{m}, \mathtt{e}\}$*. If* $t \equiv_{\mathbb{S}} t' \to_{\mathtt{x}} s$ *then there exists* s' *such that* $t \to_{\mathtt{x}} s' \equiv_{\mathbb{S}} s$*.*

Proof. See the appendix. The proofs are long, by exhaustive case analysis on all the possible diagrams that can be formed by overlapping an instance of a reduction step and an instance of an axiom of the structural equivalence (*i.e.* "critical pairs").

For instance, in call-by-need, one possible diagram involves an overlap between an exponential ($\mathtt{lsv}$) step at the top, and an instance of the structural equivalence axiom $\equiv_{@\mathtt{l}}$. Note that, on the right-hand side, the $\equiv_{@\mathtt{l}}$ axiom must be used many times in order to be able to close the diagram:

$$\begin{array}{ccc} \mathtt{N}\langle x\rangle[x\backslash \mathtt{vL}]\, t & \xrightarrow{\quad\mathtt{lsv}\quad} & \mathtt{N}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\mathtt{L}\, t \\ \equiv_{@\mathtt{l}} & & \equiv^{*}_{@\mathtt{l}} \\ (\mathtt{N}\langle x\rangle\, t)[x\backslash \mathtt{vL}] & \dashrightarrow^{\mathtt{lsv}} & (\mathtt{N}\langle \mathtt{v}\rangle\, t)[x\backslash \mathtt{v}]\mathtt{L} \end{array}$$

□

An essential property of strong bisimulations is that they can be postponed. In fact, it is immediate to prove the following for any of the five strategies $\mathbb{S}$ defined in Def. 3.3:

Lemma 3.15 (Postponement of structural equivalence)**.** *Let* $t\ (\to_{\mathtt{m}} \cup \to_{\mathtt{e}} \cup \equiv)^{*}\ s$. *Then* $t\ (\to_{\mathtt{m}} \cup \to_{\mathtt{e}})^{*} \equiv s$ *and the number of multiplicative and exponential steps in the two reduction sequences is exactly the same.*

In the simulation theorems for machines with a global environment we will also use the following commutation property between substitutions and evaluation contexts via the structural equivalence of every evaluation scheme, proved by an easy induction on the actual definition of evaluation contexts.

Lemma 3.16 (Explicit substitutions commute with evaluation contexts, up to $\equiv$)**.** *Let* $\mathtt{E}$ *denote an evaluation context for a strategy* $\mathbb{S}$. *If* $x \notin \mathtt{fv}(\mathtt{E})$ *and* $\mathtt{E}$ *does not bind any of the free variables of* s, *then* $\mathtt{E}\langle t\rangle[x\backslash s] \equiv_{\mathbb{S}} \mathtt{E}\langle t[x\backslash s]\rangle$.

3.4 Distilleries

This section presents an abstract, high-level view of the relationship between abstract machines and explicit substitution calculi, via the following notion:

Definition 3.17 (Distillery)**.** A *distillery* $\mathbb{D} = (\mathbb{M}, \mathbb{S}, \equiv, [\![\cdot]\!])$ is given by:

1. An *abstract machine* $\mathbb{M}$, given by:
   - 1.1 A deterministic reduction relation $\rightsquigarrow_{\mathbb{M}}$ on a set of states $\mathsf{State} = \{S_1, S_2, \ldots\}$.
   - 1.2 A distinguished class of states deemed *initial*, in bijection with closed λ-terms and from which one obtains the *reachable* states by applying $\rightsquigarrow_{\mathbb{M}}{}^{*}$.
   - 1.3 A partition of the transitions defining the relation $\rightsquigarrow_{\mathbb{M}}$:
     - 1.3.1 *Search* transitions, noted $\rightsquigarrow_{\mathtt{s}}$.
     - 1.3.2 *Principal* transitions, in turn partitioned into:

1.3.2.1 *Multiplicative* transitions, denoted by $\rightsquigarrow_{\mathtt{m}}$.

1.3.2.2 *Exponential* transitions, denoted by $\rightsquigarrow_{\mathtt{e}}$.

2. A deterministic *reduction strategy* $\mathbb{S}$ given by a pair $(\to_{\mathtt{m}}, \to_{\mathtt{e}})$ of rewriting relations on terms with explicit substitutions.

3. A *structural equivalence* $\equiv$ on terms with explicit substitutions, such that $\equiv$ is a strong bisimulation with respect to $\to_{\mathtt{m}}$ and $\to_{\mathtt{e}}$.

4. A *distillation* $[\![\cdot]\!]$, *i.e.* a decoding function from states to terms, such that, on reachable states:

   4.1 *Search*: $S \rightsquigarrow_{\mathtt{s}} S'$ implies $[\![S]\!] \equiv [\![S']\!]$.

   4.2 *Multiplicative*: $S \rightsquigarrow_{\mathtt{m}} S'$ implies $[\![S]\!] \to_{\mathtt{m}} \equiv [\![S']\!]$.

   4.3 *Exponential*: $S \rightsquigarrow_{\mathtt{e}} S'$ implies $[\![S]\!] \to_{\mathtt{e}} \equiv [\![S']\!]$.

Given a distillery, the following *simulation* result holds abstractly. We write $|\rho|$ for the number of steps in an execution $\rho : S \rightsquigarrow_{\mathbb{M}}{}^{*} S'$ of the machine, and $|\pi|$ for the number of steps in a derivation $\pi : t \to_{\mathbb{S}}^{*} t'$ of the strategy. Similarly, we write $|\rho|_m$ (resp. $|\pi|_m$), $|\rho|_e$ (resp. $|\pi|_e$), and $|\rho|_p$ (resp. $|\pi|_p$) for the number of multiplicative, exponential, and principal steps (*i.e.* multiplicative or exponential) in an execution of the machine (resp. in a derivation $\pi : t \to_{\mathbb{S}}^{*} t'$ of the strategy). Then:

Proposition 3.18 (Simulation)**.** *Let $\mathbb{D}$ be a distillery. Then for every execution $\rho : S \rightsquigarrow_{\mathbb{M}}{}^{*} S'$ there is a derivation $\pi : [\![S]\!] \to^{*} \equiv [\![S']\!]$ such that $|\rho|_m = |\pi|_m$, $|\rho|_e = |\pi|_e$, and $|\rho|_p = |\pi|$.*

Proof. By induction on $|\rho|$ and by the properties of the decoding, it follows that there is a derivation $\xi : [\![S]\!](\to\equiv)^{*}[\![S']\!]$ such that the number $|\rho|_p = |\xi|$. The witness π for the statement is obtained by applying the postponement of strong bisimulations (Lem. 3.15) to ξ. □

3.4.1 Reflective Distilleries

Given a distillery, one would also expect that reduction in the strategy is reflected in the machine. This result in fact requires two additional abstract properties.

Definition 3.19 (Reflective distillery)**.** A distillery is *reflective* when:

1. *Termination:* search transitions $\rightsquigarrow_{\mathtt{s}}$ *terminate* on reachable states. Hence, by determinism, every state S has a unique *search normal form* $\mathtt{nf}_{\mathtt{s}}(S)$.

2. *Progress:* if S is reachable, $\mathtt{nf}_{\mathtt{s}}(S) = S$ and $[\![S]\!] \to_{\mathtt{x}} t$ with $\mathtt{x} \in \{\mathtt{m}, \mathtt{e}\}$, then there exists S' such that $S \rightsquigarrow_{\mathtt{x}} S'$.

Then, we may prove the following reflection of steps in full generality:

Lemma 3.20 (Reflection)**.** *Let $\mathbb{D}$ be a reflective distillery, let S be a reachable state, and let $\mathtt{x} \in \{\mathtt{m}, \mathtt{e}\}$. Then, $[\![S]\!] \to_{\mathtt{x}} t$ implies that there exists a state S' such that $\mathtt{nf}_{\mathtt{s}}(S) \rightsquigarrow_{\mathtt{x}} S'$ and $[\![S']\!] \equiv t$.*

In other words, every rewriting step on the calculus can be also performed on the machine, up to search transitions.

Proof. The proof is by induction on the number n of transitions leading from S to $\mathtt{nf}_{\mathtt{s}}(S)$.

- *Base case* $n = 0$: by the progress property, we have $S \to_{\mathtt{x}'} S'$ for some state S' and $\mathtt{x}' \in \{\mathtt{m}, \mathtt{e}\}$. By Prop. 3.18, we have $[\![S]\!] \to_{\mathtt{x}'} s \equiv [\![S']\!]$ and we may conclude because $\mathtt{x}' = \mathtt{x}$ and $s = t$ by determinism of the calculus (Prop. 3.11).
- *Inductive case* $n > 0$: by hypothesis, we have $S \rightsquigarrow_{\mathtt{s}} S''$. By Thm. 3.18, $[\![S]\!] \equiv [\![S'']\!]$. The hypothesis and the strong bisimulation property (Prop. 3.14) then give us $[\![S'']\!] \to_{\mathtt{x}} s \equiv t$. But the induction hypothesis holds for S'', giving us a state S' such that $\mathtt{nf}_{\mathtt{s}}(S'') \rightsquigarrow_{\mathtt{x}} S'$ and $[\![S']\!] \equiv s \equiv t$. We may now conclude because $\mathtt{nf}_{\mathtt{s}}(S) = \mathtt{nf}_{\mathtt{s}}(S'')$.

□

The preceding lemma can then be easily extended to a reverse simulation result:

Proposition 3.21 (Reverse simulation)**.** *Let $\mathbb{D}$ be a reflective distillery and let S be an initial state. Given a derivation $\pi : [\![S]\!] \to^* t$ there is an execution $\rho : S \rightsquigarrow_{\mathbb{M}}^* S'$ such that $t \equiv [\![S']\!]$ and $|\rho|_m = |\pi|_m$, $|\rho|_e = |\pi|_e$, and $|\rho|_p = |\pi|$.*

Proof. By induction on the length of π, using Prop. 3.20. □

3.5 Abstract Machines

In this section we introduce *abstract machines* and *distillations*, and we prove that they form *reflective distilleries* with respect to the strategies of Section 3.2. For each machine we prove: (1) that the decoding is in fact a distillation, and (2) the progress property. For the moment we assume the termination property, whose proof is delayed to the quantitative study of Section 3.6, where we prove stronger results, giving explicit bounds.

3.5.1 Call-by-Name: the KAM

The Krivine Abstract Machine (KAM), originally introduced by Jean-Louis Krivine [97], is the first machine studied in this chapter. Note however that Krivine's presentation of the KAM uses de Bruijn indices, whereas we use variable names.

Definition 3.22 (Krivine Abstract Machine)**.** A KAM *state* $(S, S', S'', \ldots)$ is a pair (c, π), where c is a closure and π is a *stack* of closures:

$$\pi \quad ::= \quad \epsilon \mid c :: \pi \qquad\qquad S \quad ::= \quad (c, \pi)$$

For readability, we use the notation $\bar{t} \mid e \mid \pi$ for a state (c, π) where $c = (\bar{t}, e)$. The transitions of the KAM then are:

$$\begin{array}{rcccccc} \overline{t}\overline{s} & \mid & e & \mid & \pi & \rightsquigarrow_{\mathtt{s}} & \overline{t} \mid e \mid (\overline{s}, e) :: \pi \\ \lambda x.\overline{t} & \mid & e & \mid & c :: \pi & \rightsquigarrow_{\mathtt{m}} & \overline{t} \mid [x\backslash c] :: e \mid \pi \\ x & \mid & e & \mid & \pi & \rightsquigarrow_{\mathtt{e}} & \overline{t} \mid e' \mid \pi \end{array}$$

where $\rightsquigarrow_{\mathtt{e}}$ takes place only if $e = e_1 :: [x\backslash(\overline{t}, e')] :: e_2$.

A key point of our study is that environments and stacks can be readily understood as contexts, through the following decoding, which satisfies the properties stated in the following lemma:

Definition 3.23 (KAM decoding)**.**

$$\begin{array}{rcl@{\qquad\qquad}rcl} [\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box & [\![[x\backslash c] :: e]\!] & \stackrel{\text{def}}{=} & [\![e]\!]\langle\Box[x\backslash[\![c]\!]]\rangle \\ [\![(\overline{t}, e)]\!] & \stackrel{\text{def}}{=} & [\![e]\!]\langle\overline{t}\rangle & [\![c :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle\Box[\![c]\!]\rangle \\ [\![\overline{t} \mid e \mid \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle[\![e]\!]\langle\overline{t}\rangle\rangle \end{array}$$

Lemma 3.24 (Contextual decoding)**.** *Let e be an environment and π be a stack of the KAM. Then$[\![e]\!]$ is a substitution context, and both $[\![\pi]\!]$ and $[\![\pi]\!]\langle[\![e]\!]\rangle$ are call-by-name evaluation contexts.*

Proof. Straightforward by induction on e and π. □

Next, we state the dynamic invariants of the machine. Recall that a code/environment/-closure X is *well-named* if its support $\mathsf{supp}(X)$ has no repetitions, *i.e.* bindings do not shadow existing names.

Lemma 3.25 (KAM invariants)**.** *Let $S = \overline{s} \mid e \mid \pi$ be a KAM reachable state whose initial code $\overline{t}$ is well-named. Then:*

1. Closure*: every closure in S is closed.*
2. Subterm*: any code in S is a literal subterm of $\overline{t}$.*
3. Name*: any closure c in S is well-named and its names are names of $\overline{t}$ (*i.e. $\mathsf{supp}(c) \subseteq \mathtt{fv}(\overline{t})$*).*
4. Environment Size*: the length of any environment in S is bound by $|\overline{t}|$.*

Proof. It is routine to check that the invariant is preserved, by direct inspection of the machine transitions. □

Abstract Considerations on Concrete Implementations. The *name* invariant is the abstract property that allows to avoid both α-equivalence and name generation in KAM executions. Note that, by definition of well-named closure, there cannot be repetitions in the support of an environment. Then the length of any environment in any reachable state is bound by the number of distinct names in the initial code $\overline{t}$, *i.e.* with $|\overline{t}|$. This fact is important, as the static bound on the size of environments guarantees that $\rightsquigarrow_{\mathtt{e}}$ and $\rightsquigarrow_{\mathtt{s}}$—the transitions looking-up and copying environments—can be implemented (independently of the chosen

concrete representation of terms) in at worst linear time in $|t|$, so that an execution ρ can be implemented in $O(|\rho| \cdot |t|)$. The same will hold for every machine with local environments. In fact, we may turn this into a definition: an abstract machine is *reasonable* if its implementation enjoys the above bilinear bound. In this way, the length of an execution of a reasonable machine provides an accurate estimate of its implementation cost.

The previous considerations are based on the name and environment size invariants. The closure invariant is used in the progress part of the next theorem, and the subterm invariant is used in the complexity analysis of Section 3.6, subsuming the termination condition of reflective distilleries.

Theorem 3.26 (KAM distillation)**.** $(\mathrm{KAM}, \mathtt{name}, \equiv, [\![\cdot]\!])$ *is a reflective distillery.*

Proof.

1. *Properties of the decoding:*

   1.1 *Search.* Let $\overline{t}\overline{s} \mid e \mid \pi \rightsquigarrow_{\mathtt{s}} \overline{t} \mid e \mid (\overline{s}, e) :: \pi$. Then:

   $$\begin{array}{rcl} [\![\overline{t}\overline{s} \mid e \mid \pi]\!] & = & [\![\pi]\!]\langle [\![e]\!]\langle \overline{t}\overline{s}\rangle\rangle \\ & \equiv^{*}_{@} & [\![\pi]\!]\langle [\![e]\!]\langle \overline{t}\rangle [\![e]\!]\langle \overline{s}\rangle\rangle \;=\; [\![\overline{t} \mid e \mid (\overline{s}, e) :: \pi]\!] \end{array}$$

   1.2 *Multiplicative.* Let $\lambda x.\overline{t} \mid e \mid c :: \pi \rightsquigarrow_{\mathtt{m}} \overline{t} \mid [x\backslash c] :: e \mid \pi$. Then:

   $$\begin{array}{rcl} [\![\lambda x.\overline{t} \mid e \mid c :: \pi]\!] & = & [\![\pi]\!]\langle [\![e]\!]\langle \lambda x.\overline{t}\rangle [\![c]\!]\rangle \\ & \to_{\mathtt{m}} & [\![\pi]\!]\langle [\![e]\!]\langle \overline{t}[x\backslash [\![c]\!]]\rangle\rangle \\ & = & [\![\overline{t} \mid [x\backslash c] :: e \mid \pi]\!] \end{array}$$

   The rewriting step can be applied because by contextual decoding (Lem. 3.24) it takes place in an evaluation context.

   1.3 *Exponential.* Let $x \mid e' :: [x\backslash(\overline{t}, e)] :: e'' \mid \pi \rightsquigarrow_{\mathtt{e}} \overline{t} \mid e \mid \pi$. Then:

   $$\begin{array}{rcl} [\![x \mid e' :: [x\backslash(\overline{t}, e)] :: e'' \mid \pi]\!] & = & [\![\pi]\!]\langle [\![e'']\!]\langle [\![e']\!]\langle x\rangle [x\backslash [\![e]\!]\langle \overline{t}\rangle]\rangle\rangle \\ & \to_{\mathtt{e}} & [\![\pi]\!]\langle [\![e'']\!]\langle [\![e']\!]\langle [\![e]\!]\langle \overline{t}\rangle\rangle [x\backslash [\![e]\!]\langle \overline{t}\rangle]\rangle\rangle \\ & \equiv^{*}_{\mathtt{gc}} & [\![\pi]\!]\langle [\![e]\!]\langle \overline{t}\rangle\rangle \\ & = & [\![\overline{t} \mid e \mid \pi]\!] \end{array}$$

   Note that $e''\langle e'\langle e\langle \overline{t}\rangle\rangle [x\backslash e\langle \overline{t}\rangle]\rangle \equiv^{*}_{\mathtt{gc}} e\langle \overline{t}\rangle$ holds because $e\langle \overline{t}\rangle$ is closed by item 1 of Lem. 3.25, and so all the substitutions around it can be garbage collected.

2. *Termination:* Given by (forthcoming) Thm. 3.72.
   Note: future proofs of distillation theorems will omit termination.

3. *Progress:* Let $S = \overline{t} \mid e \mid \pi$ be a commutative normal form such that $[\![S]\!] \to s$. If $\overline{t}$ is

   3.1 *an application* $\overline{s}\overline{u}$. Then a $\rightsquigarrow_{\mathtt{s}}$ transition applies and S is not a commutative normal form, impossible.

   3.2 *an abstraction* $\lambda x.\overline{s}$: if $\pi = \epsilon$ then $[\![S]\!] = [\![e]\!]\langle \lambda x.\overline{s}\rangle$, which is $\to$-normal, impossible. Hence, a $\rightsquigarrow_{\mathtt{m}}$ transition applies.

3.3 *a variable* x: by point 1 of Lem. 3.25.1, we must have $e = e' :: [x\backslash c] :: e''$, so a $\rightsquigarrow_{\mathtt{e}}$ transition applies.

□

3.5.2 Call-by-Name with Global Environment: the MAM

The LSC suggests the design of a simpler version of the KAM, that we call the *Milner Abstract Machine* (MAM), that avoids the concept of closure. At the language level, the idea is that, by repeatedly applying the axioms $\equiv_{\mathtt{dup}}$ and $\equiv_{@}$ of the structural equivalence, explicit substitutions can be brought *outside*. At the machine level, the local environments in the closures are replaced by just one global environment that closes the code and the stack, as well as the global environment itself.

Naively turning to a global environment breaks the well-named invariant of the machine. This point is addressed using an α-renaming and name generation in the variable (or exponential) transition, *i.e.* when substitution takes place.

Definition 3.27 (Milner Abstract Machine)**.** The MAM employs global environments E. Stacks are lists of codes, *i.e.* $\pi ::= \epsilon \mid \bar{t} :: \pi$. A state of is a triple $S = (\bar{t}, \pi, E)$. The transitions of the MAM are:

$$\begin{array}{rcccl}
\bar{t}\bar{s} & \mid \pi \mid & E & \rightsquigarrow_{\mathtt{s}} & \bar{t} \mid \bar{s} :: \pi \mid E \\
\lambda x.\bar{t} \mid \bar{s} :: \pi \mid & E & \rightsquigarrow_{\mathtt{m}} & \bar{t} \mid \pi \mid & [x\backslash\bar{s}] :: E \\
x & \mid \pi \mid & E & \rightsquigarrow_{\mathtt{e}} & \bar{t}^{\alpha} \mid \pi \mid E
\end{array}$$

where $\rightsquigarrow_{\mathtt{e}}$ takes place only if $E = E''\langle E'[x\backslash\bar{t}]\rangle$ and $\bar{t}^{\alpha}$ is a well-named code α-equivalent to $\bar{t}$ and such that any bound name in $\bar{t}^{\alpha}$ is fresh with respect to those in π and E.

Definition 3.28 (MAM decoding)**.** The decoding of a MAM state $\bar{t} \mid \pi \mid E$ is similar to the decoding of a KAM state, but the stack and the environment context are applied in reverse order:

$$\begin{array}{rclcrcl}
[\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box & \qquad & [\![[x\backslash\bar{t}] :: E]\!] & \stackrel{\text{def}}{=} & [\![E]\!]\langle\Box[x\backslash\bar{t}]\rangle \\
[\![\bar{t} :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle\Box\bar{t}\rangle & \qquad & [\![\bar{t} \mid \pi \mid E]\!] & \stackrel{\text{def}}{=} & [\![E]\!]\langle[\![\pi]\!]\langle\bar{t}\rangle\rangle
\end{array}$$

To every MAM state $\bar{t} \mid \pi \mid E$ we associate the pair $([\![\pi]\!]\langle\bar{t}\rangle, E)$, and call it the *global closure* of the state. Note that $[\![\pi]\!]\langle\bar{t}\rangle$ now is a code, *i.e.* it does not contain explicit substitutions.

Lemma 3.29 (Contextual decoding)**.** *Let E be a global environment and π be a stack of the MAM. Then $[\![E]\!]$ is a substitution context, and both $[\![\pi]\!]$ and $[\![\pi]\!]\langle[\![E]\!]\rangle$ are evaluation contexts.*

Proof. Straightforward by induction on E and π. □

For the dynamic invariants we need a different notion of closed closure.

Definition 3.30. Given a global environment E and a code $\bar{t}$, we define by mutual induction two predicates E *is closed* and $(\bar{t}, E)$ *is closed* as follows:

$$\begin{array}{rcl}
 & & \epsilon \text{ is closed} \\
(\bar{t}, E) \text{ is closed} & \Longrightarrow & [x\backslash t] :: E \text{ is closed} \\
\mathtt{fv}(\bar{t}) \subseteq \mathsf{supp}(E) \wedge E \text{ is closed} & \Longrightarrow & (\bar{t}, E) \text{ is closed}
\end{array}$$

The dynamic invariants are:

Lemma 3.31 (MAM invariants)**.** *Let $S = \overline{s} \mid \pi \mid E$ be a MAM state reached by an execution ρ of initial well-named code $\overline{t}$. Then:*

1. Global Closure: *the global closure $(\llbracket\pi\rrbracket\langle\overline{t}\rangle, E)$ of S is closed;*
2. Subterm: *any code in S is a literal subterm of $\overline{t}$;*
3. Names: *the global closure of S is well-named;*
4. Environment Size: *the length of the global environment in S is bound by $|\rho|_m$.*

Proof. Straightforward by inspection of the machine transitions. □

Abstract Considerations on Concrete Implementations. Note the new environment size invariant. The bound now depends on the length of the execution ρ, not on the size of the initial term $\overline{t}$. If one implements $\rightsquigarrow_{\mathtt{e}}$ looking for x in E sequentially, then each $\rightsquigarrow_{\mathtt{e}}$ transition has cost $O(|\rho|_m)$, and the cost of implementing ρ is easily seen to become quadratic in $|\rho|$. Therefore—at first sight—the MAM is not a reasonable abstract machine. However, the MAM is meant to be implemented using a representation of codes pointers for variables, so that looking for x in E takes constant time. Then the global environment, even if formalized as a list, should rather be considered as a store.

The name invariant is what guarantees that variables can indeed be taken as pointers, as there is no name clash. Note that the cost of a $\rightsquigarrow_{\mathtt{e}}$ transition is not constant, as the renaming operation actually makes $\rightsquigarrow_{\mathtt{e}}$ linear in $|t|$ (by the subterm invariant). So, assuming a pointer-based representation, ρ can be implemented in time $O(|\rho| \cdot |\overline{t}|)$, as for local machines. In other words, the MAM is a reasonable abstract machine.

Theorem 3.32 (MAM distillation)**.** $(\text{MAM}, \mathtt{name}, \equiv, \llbracket\cdot\rrbracket)$ *is a reflective distillery. In particular, on a reachable state S we have:*

1. Search*: if $S \rightsquigarrow_{\mathtt{s}} S'$ then $\llbracket S\rrbracket = \llbracket S'\rrbracket$;*
2. Multiplicative*: if $S \rightsquigarrow_{\mathtt{m}} S'$ then $\llbracket S\rrbracket \to_{\mathtt{m}} \equiv \llbracket S'\rrbracket$;*
3. Exponential*: if $S \rightsquigarrow_{\mathtt{e}} S'$ then $\llbracket S\rrbracket \to_{\mathtt{e}} =_\alpha \llbracket S'\rrbracket$.*

Proof. Properties of the decoding (*progress* is as for the KAM):

1. *Search.* In contrast to the KAM, $\rightsquigarrow_{\mathtt{s}}$ gives a true identity:

$$\llbracket\overline{t}\overline{s} \mid \pi \mid E\rrbracket \;=\; \llbracket E\rrbracket\langle\llbracket\pi\rrbracket\langle\overline{t}\overline{s}\rangle\rangle \;=\; \llbracket\overline{t} \mid \overline{s} :: \pi \mid E\rrbracket$$

2. *Multiplicative.* Since substitutions and evaluation contexts commute via $\equiv$ (Lem. 3.16), $\rightsquigarrow_{\mathtt{m}}$ maps to:

$$\begin{array}{rcll} [\![\lambda x.\bar{t} \mid \bar{s} :: \pi \mid E]\!] & = & [\![E]\!]\langle [\![\pi]\!]\langle(\lambda x.\bar{t})\bar{s}\rangle\rangle & \to_{\mathtt{m}} \\ & & [\![E]\!]\langle [\![\pi]\!]\langle \bar{t}[x\backslash\bar{s}]\rangle\rangle & \equiv_{Lem.3.16} \\ & & [\![E]\!]\langle [\![\pi]\!]\langle \bar{t}\rangle[x\backslash\bar{s}]\rangle & = \\ & & [\![\bar{t} \mid \pi \mid [x\backslash\bar{s}] :: E]\!] & \end{array}$$

3. *Exponential.* The erasure of part of the environment of the KAM is replaced by an explicit use of α-equivalence:

$$\begin{array}{rcll} [\![x \mid \pi \mid E :: [x\backslash\bar{s}] :: E']\!] & = & [\![E']\!]\langle [\![E]\!]\langle [\![\pi]\!]\langle x\rangle\rangle[x\backslash\bar{s}]\rangle & \to_{\mathtt{e}} \\ & & [\![E']\!]\langle [\![E]\!]\langle [\![\pi]\!]\langle \bar{s}\rangle\rangle[x\backslash\bar{s}]\rangle & =_{\alpha} \\ & & [\![E']\!]\langle [\![E]\!]\langle [\![\pi]\!]\langle \bar{s}^{\alpha}\rangle\rangle[x\backslash\bar{s}]\rangle & = \\ & & [\![\bar{s}^{\alpha} \mid \pi \mid E :: [x\backslash\bar{s}] :: E']\!] & \end{array}$$

□

3.5.3 Left-to-Right Call-by-Value: the CEK

In this section we present an adaptation to call-by-value of the KAM, namely Felleisen and Friedman's CEK machine [56] (without control operators), implementing left-to-right call-by-value.

States of the CEK have the same shape of those of the KAM, *i.e.* they are given by a closure plus a stack. The difference is that they use *call-by-value stacks*, whose elements are labeled either as *arguments* or *functions*, so that the machine knows whether the code currently being evaluated is a function that must be applied to a yet unevaluated argument on top of the stack or the argument to the already evaluated function on top of the stack.

Definition 3.33 (CEK Machine)**.** Stacks are defined as follows:

$$\pi \quad ::= \quad \epsilon \mid \mathbf{f}(c) :: \pi \mid \mathbf{a}(c) :: \pi$$

A state is a triple $S = (\bar{t}, e, \pi)$. The transitions of the CEK are:

$$\begin{array}{llllll} \bar{t}\bar{s} \mid e \mid & \pi & \rightsquigarrow_{\mathtt{s}_1} & \bar{t} \mid & e & \mid \mathbf{a}(\bar{s}, e) :: \pi \\ \bar{\mathtt{v}} \mid e \mid & \mathbf{a}(\bar{s}, e') :: \pi & \rightsquigarrow_{\mathtt{s}_2} & \bar{s} \mid & e' & \mid \mathbf{f}(\bar{\mathtt{v}}, e) :: \pi \\ \bar{\mathtt{v}} \mid e \mid \mathbf{f}(\lambda x.\bar{t}, e') :: \pi & & \rightsquigarrow_{\mathtt{m}} & \bar{t} \mid [x\backslash(\bar{\mathtt{v}}, e)] :: e' \mid & & \pi \\ x \mid e \mid & \pi & \rightsquigarrow_{\mathtt{e}} & \bar{t} \mid & e' & \mid \pi \end{array}$$

where $\rightsquigarrow_{\mathtt{e}}$ takes place only if $e = e'' :: [x\backslash(\bar{t}, e')] :: e'''$.

Definition 3.34 (CEK decoding)**.** Stacks are decoded as follows:

$$\begin{array}{rcl} [\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box \\ [\![\mathbf{f}(c) :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle [\![c]\!]\Box\rangle \\ [\![\mathbf{a}(c) :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle \Box[\![c]\!]\rangle \end{array}$$

States of the machine are decoded exactly as for the KAM, *i.e.* $[\![\bar{t} \mid e \mid \pi]\!] \stackrel{\text{def}}{=} [\![\pi]\!]\langle [\![e]\!]\langle \bar{t}\rangle\rangle$.

While one can still statically prove that environments decode to substitution contexts, to prove that $[\![\pi]\!]$ and $[\![\pi]\!]\langle[\![e]\!]\rangle$ are evaluation contexts we need the dynamic invariants of the machine.

Lemma 3.35 (CEK invariants)**.** *Let $S = \overline{s} \mid e \mid \pi$ be a CEK reachable state whose initial code $\overline{t}$ is well-named. Then:*

1. Closure*: every closure in S is closed;*
2. Subterm*: any code in S is a literal subterm of $\overline{t}$;*
3. Value*: any code in e is a value and, for every element of π of the form $\mathbf{f}(\overline{s}, e')$, $\overline{s}$ is a value;*
4. Contextual Decoding*: $[\![\pi]\!]$ and $[\![\pi]\!]\langle[\![e]\!]\rangle$ are left-to-right call-by-value evaluation contexts;*
5. Name*: any closure c in S is well-named and its names are names of $\overline{t}$ (*i.e. $\mathsf{supp}(c) \subseteq \mathtt{fv}(\overline{t})$*);*
6. Environment Size*: the length of any environment in S is bound by $|\overline{t}|$.*

Proof. Straightforward by inspection of the machine transitions. □

Theorem 3.36 (CEK distillation)**.** $(\text{CEK}, \mathtt{value}^{\mathtt{LR}}, \equiv, [\![\cdot]\!])$ *is a reflective distillery. In particular, on a reachable state S we have:*

1. Search 1*: if $S \rightsquigarrow_{\mathtt{s}_1} S'$ then $[\![S]\!] \equiv [\![S']\!]$;*
2. Search 2*: if $S \rightsquigarrow_{\mathtt{s}_2} S'$ then $[\![S]\!] = [\![S']\!]$.*
3. Multiplicative*: if $S \rightsquigarrow_{\mathtt{m}} S'$ then $[\![S]\!] \to_{\mathtt{m}} [\![S']\!]$;*
4. Exponential*: if $S \rightsquigarrow_{\mathtt{e}} S'$ then $[\![S]\!] \to_{\mathtt{e}} \equiv [\![S']\!]$;*

Proof. Properties of the decoding: in the following cases, evaluation will always takes place under a context that by Lem. 3.35.4 will be a left-to-right call-by-value evaluation context, and similarly structural equivalence will alway be used in a weak context, as it should be.

1. *Search 1.* We have $\overline{t}\overline{s} \mid e \mid \pi \rightsquigarrow_{\mathtt{s}_1} \overline{t} \mid e \mid \mathbf{a}(\overline{s}, e) :: \pi$:

$$\begin{array}{rcll} [\![\overline{t}\overline{s} \mid e \mid \pi]\!] & = & [\![\pi]\!]\langle[\![e]\!]\langle\overline{t}\overline{s}\rangle\rangle & \equiv^{*}_{@} \\ & & [\![\pi]\!]\langle[\![e]\!]\langle\overline{t}\rangle[\![e]\!]\langle\overline{s}\rangle\rangle & = \quad [\![\overline{t} \mid e \mid \mathbf{a}(\overline{s}, e) :: \pi]\!] \end{array}$$

2. *Search 2.* We have $\overline{\mathtt{v}} \mid e \mid \mathbf{a}(\overline{s}, e') :: \pi \rightsquigarrow_{\mathtt{s}_2} \overline{s} \mid e' \mid \mathbf{f}(\overline{\mathtt{v}}, e) :: \pi$, and:

$$\begin{array}{rcll} [\![\overline{\mathtt{v}} \mid e \mid \mathbf{a}(\overline{s}, e') :: \pi]\!] & = & [\![\pi]\!]\langle[\![e]\!]\langle\overline{\mathtt{v}}\rangle[\![e']\!]\langle\overline{s}\rangle\rangle & = \\ & & [\![\overline{s} \mid e' \mid \mathbf{f}(\overline{\mathtt{v}}, e) :: \pi]\!] \end{array}$$

3. *Multiplicative.* We have $\overline{\mathtt{v}} \mid e \mid \mathbf{f}(\lambda x.\overline{t}, e') :: \pi \rightsquigarrow_{\mathtt{m}} \overline{s} \mid [x\backslash(\overline{\mathtt{v}}, e)] :: e' \mid \pi$, and:

$$\begin{array}{rcll} [\![\overline{\mathtt{v}} \mid e \mid \mathbf{f}(\lambda x.\bar{t}, e') :: \pi]\!] & = & [\![\pi]\!]\langle [\![e']\!]\langle \lambda x.\bar{t}\rangle [\![e]\!]\langle \overline{\mathtt{v}}\rangle\rangle & \to_{\mathtt{m}} \\ & & [\![\pi]\!]\langle [\![e']\!]\langle \bar{t}[x\backslash [\![e]\!]\langle \overline{\mathtt{v}}\rangle]\rangle\rangle & = \\ & & [\![\bar{t} \mid [x\backslash(\overline{\mathtt{v}}, e)] :: e' \mid \pi]\!] & \end{array}$$

4. *Exponential.* Let $e = e'' :: [x\backslash(\bar{t}, e')] :: e'''$. We have $x \mid e \mid \pi \rightsquigarrow_{\mathtt{e}} \bar{t} \mid e' \mid \pi$, and:

$$\begin{array}{rcll} [\![x \mid e \mid \pi]\!] & = & \pi\langle e\langle x\rangle\rangle & = \\ & & [\![\pi]\!]\langle [\![e''']\!]\langle [\![e'']\!]\langle x\rangle [x\backslash [\![e']\!]\langle \bar{t}\rangle]\rangle\rangle & \to_{\mathtt{e}} \\ & & [\![\pi]\!]\langle [\![e''']\!]\langle [\![e']\!]\langle [\![e'']\!]\langle \bar{t}\rangle [x\backslash \bar{t}]\rangle\rangle\rangle & \equiv^{*}_{\mathtt{gc}} \\ & & [\![\pi]\!]\langle [\![e']\!]\langle \bar{t}\rangle\rangle & = \quad [\![\bar{t} \mid e' \mid \pi]\!] \end{array}$$

   We can apply $\to_{\mathtt{e}}$ since by Lem. 3.35.3, $\bar{t}$ is a value. We also use that by Lem. 3.35.1, $[\![e']\!]\langle \bar{t}\rangle$ is a closed term to ensure that $[\![e'']\!]$ and $[\![e''']\!]$ can be garbage collected.

Progress. Let $S = \bar{t} \mid e \mid \pi$ be a commutative normal form such that $[\![S]\!] \to s$. If $\bar{t}$ is

- *an application* $\overline{su}$. Then a $\rightsquigarrow_{\mathtt{s}_1}$ transition applies and S is not a commutative normal form, absurd;
- *an abstraction* $\overline{\mathtt{v}}$: by hypothesis, π cannot be of the form $\mathbf{a}(c) :: \pi'$. Suppose it is equal to ϵ. We would then have $[\![S]\!] = [\![e]\!]\langle \overline{\mathtt{v}}\rangle$, which is a call-by-value normal form, because $[\![e]\!]$ is a substitution context. This would contradict our hypothesis, so π must be of the form $\mathbf{f}(\overline{s}, e') :: \pi'$. By point 3 of Lem. 3.35, $\overline{s}$ is an abstraction, hence a $\rightsquigarrow_{\mathtt{m}}$ transition applies;
- *a variable* x: by point 1 of Lem. 3.35, e must be of the form $e' :: [x\backslash c] :: e''$, so a $\rightsquigarrow_{\mathtt{e}}$ transition applies.

□

3.5.4 Left-to-Right Call-by-Value: the Split CEK

For the CEK machine we proved that the stack, that collects both arguments and functions, decodes to an evaluation context (Lem. 3.35.4). In this section we study another left-to-right call-by-value machine, deemed *Split CEK* (SCEK), which has two stacks: one for arguments and one for functions. Both decode to evaluation contexts.

Note that the evaluation contexts $\mathtt{V}$ for the calculus $\mathtt{value}^{\mathtt{LR}}$:

$$\mathtt{V} ::= \Box \mid \mathtt{V}t \mid \mathtt{vL}\,\mathtt{V} \mid \mathtt{V}[x\backslash t]$$

have two cases for the application. Essentially, when dealing with $\mathtt{V}t$ the machine puts t in a stack for arguments (identical to the stack of the KAM), while in the case $\mathtt{vL}\,\mathtt{V}$ the machine puts the closure (corresponding to) $\mathtt{vL}$ in a stack for functions, called *dump*. Actually, together with the closure it stores the current argument stack.

Thus, an entry of the function stack is a pair (c, π), where c is a closure $(\overline{\mathtt{v}}, e)$, and the three components $\overline{\mathtt{v}}$, e, and π together correspond to the evaluation context $[\![\pi]\!]\langle [\![e]\!]\langle \overline{\mathtt{v}}\Box\rangle\rangle$.

Whenever the code is an abstraction $\overline{\mathtt{v}}$ and the argument stack π is non-empty (*i.e.* $\pi = c :: \pi'$), the machine saves the active closure, given by current code $\overline{\mathtt{v}}$ and environment e, and the remainder of the stack π' by pushing a new entry $((\overline{\mathtt{v}}, e), \pi')$ on the dump, and then starts evaluating the first closure c of the stack. In terms of the concrete implementation, each element of the dump corresponds roughly to a *stack frame* or *activation record*.

Definition 3.37 (SCEK Machine)**.** Stacks are defined as in the KAM. The syntax for dumps is given by:

$$D ::= \epsilon \mid (c, \pi) :: D$$

States are 4-uples $(\overline{t}, e, \pi, D)$. The transitions of the SCEK are:

$$\begin{array}{llllll}
\overline{t}\overline{s} \mid & e & \mid \pi & \mid D & \rightsquigarrow_{\mathtt{s}_1} & \overline{t} \mid e \mid (\overline{s}, e) :: \pi \mid D \\
\overline{\mathtt{v}} \mid & e & \mid (\overline{t}, e') :: \pi & \mid D & \rightsquigarrow_{\mathtt{s}_2} & \overline{t} \mid e' \mid \epsilon \mid ((\overline{\mathtt{v}}, e), \pi) :: D \\
\overline{\mathtt{v}} \mid & e & \mid \epsilon & \mid ((\lambda x.\overline{t}, e'), \pi) :: D & \rightsquigarrow_{\mathtt{m}} & \overline{t} \mid [x \backslash (\overline{\mathtt{v}}, e)] :: e' \mid \pi \mid D \\
x \mid & e :: [x \backslash (\overline{\mathtt{v}}, e')] :: e'' \mid \pi & \mid D & & \rightsquigarrow_{\mathtt{e}} & \overline{\mathtt{v}} \mid e' \mid \pi \mid D
\end{array}$$

Definition 3.38 (SCEK decoding)**.** The decoding of terms, environments, closures, and stacks is as for the KAM. Every dump decodes to a context according to:

$$[\![\epsilon]\!] \stackrel{\text{def}}{=} \Box \qquad [\![((\overline{\mathtt{v}}, e), \pi) :: D]\!] \stackrel{\text{def}}{=} [\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\langle \overline{\mathtt{v}}\Box \rangle\rangle\rangle$$

The decoding of states is defined as $[\![\overline{t} \mid e \mid \pi \mid D]\!] \stackrel{\text{def}}{=} [\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\langle \overline{t} \rangle\rangle\rangle$.

The SCEK machine is closely related with Landin's SECD machine [99], which also incorporates a notion of dump. In [48], Danvy studies the SECD machine, and shows that the SECD implements right-to-left call-by-value (and not left-to-right call-by-value as the SCEK). Our main point here is illustrating that "splitting the stack" into an argument stack plus a dump is a general transformation.

Lemma 3.39 (SCEK invariants)**.** *Let $S = \overline{s} \mid e \mid \pi \mid D$ be a SCEK reachable state whose initial code $\overline{t}$ is well-named. Then:*

1. Closure: *every closure in S is closed;*
2. Subterm: *any code in S is a literal subterm of $\overline{t}$;*
3. Value: *the code of any closure in the dump or in any environment in S is a value;*
4. Contextual Decoding*: $[\![D]\!]$, $[\![D]\!]\langle [\![\pi]\!]\rangle$, and $[\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\rangle\rangle$ are left-to-right call-by-value evaluation contexts.*
5. Name*: any closure c in S is well-named and its names are names of $\overline{t}$ (*i.e. $\mathsf{supp}(c) \subseteq \mathtt{fv}(\overline{t})$*).*
6. Environment Size*: the length of any environment in S is bound by $|\overline{t}|$.*

Proof. Straightforward by inspection of the machine transitions. □

Theorem 3.40 (SCEK distillation)**.** $(\text{SCEK}, \mathtt{value}^{\mathtt{LR}}, \equiv, [\![\cdot]\!])$ *is a reflective distillery. In particular, on a reachable state S we have:*

1. Search 1*: if* $S \rightsquigarrow_{\mathtt{s}_1} S'$ *then* $[\![S]\!] \equiv [\![S']\!]$*;*
2. Search 2*: if* $S \rightsquigarrow_{\mathtt{s}_2} S'$ *then* $[\![S]\!] \equiv [\![S']\!]$*;*
3. Multiplicative*: if* $S \rightsquigarrow_{\mathtt{m}} S'$ *then* $[\![S]\!] \to_{\mathtt{m}} [\![S']\!]$*;*
4. Exponential*: if* $S \rightsquigarrow_{\mathtt{e}} S'$ *then* $[\![S]\!] \to_{\mathtt{e}} \equiv [\![S']\!]$.

Proof. Properties of the decoding:

1. *Search 1.* We have $\overline{t}\,\overline{s} \mid e \mid \pi \mid D \rightsquigarrow_{\mathtt{s}_1} \overline{t} \mid e \mid (\overline{s}, e) :: \pi \mid D$, and:

$$\begin{array}{rcll} [\![\overline{t}\,\overline{s} \mid e \mid \pi \mid D]\!] & = & [\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\langle \overline{t}\,\overline{s}\rangle\rangle\rangle & \equiv^*_{@} \\ & & [\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\langle \overline{t}\rangle\, [\![e]\!]\langle \overline{s}\rangle\rangle\rangle & = \\ & & [\![\overline{t} \mid e \mid (\overline{s}, e) :: \pi \mid D]\!] & \end{array}$$

2. *Search 2.* We have $\overline{\mathtt{v}} \mid e \mid (\overline{t}, e') :: \pi \mid D \rightsquigarrow_{\mathtt{s}_2} \overline{t} \mid e' \mid \epsilon \mid ((\overline{\mathtt{v}}, e), \pi) :: D$, and:

$$\begin{array}{rcll} [\![\overline{\mathtt{v}} \mid e \mid (\overline{t}, e') :: \pi \mid D]\!] & = & [\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\langle \mathtt{v}\rangle\, [\![e']\!]\langle \overline{t}\rangle\rangle\rangle & \equiv^*_{\mathtt{gc}} \\ & & [\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\langle \mathtt{v}\rangle\, [\![e]\!]\langle [\![e']\!]\langle \overline{t}\rangle\rangle\rangle\rangle & \equiv^*_{@} \\ & & [\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\langle \mathtt{v}\, [\![e']\!]\langle \overline{t}\rangle\rangle\rangle\rangle & = \\ & & [\![\overline{t} \mid e' \mid \epsilon \mid ((\overline{\mathtt{v}}, e), \pi) :: D]\!] & \end{array}$$

3. *Multiplicative.* We have $\overline{\mathtt{v}} \mid e \mid \epsilon \mid ((\lambda x.\overline{t}, e'), \pi) :: D \rightsquigarrow_{\mathtt{m}} \overline{t} \mid [x\backslash(\overline{\mathtt{v}}, e)] :: e' \mid \pi \mid D$, and:

$$\begin{array}{rcl} [\![\overline{\mathtt{v}} \mid e \mid \epsilon \mid ((\lambda x.\overline{t}, e'), \pi) :: D]\!] & = & \\ [\![D]\!]\langle [\![\pi]\!]\langle [\![e']\!]\langle (\lambda x.t)\, [\![e]\!]\langle \mathtt{v}\rangle\rangle\rangle\rangle & \rightsquigarrow_{\mathtt{m}} & \\ [\![D]\!]\langle [\![\pi]\!]\langle [\![e']\!]\langle t[x\backslash [\![e]\!]\langle \mathtt{v}\rangle]\rangle\rangle\rangle & = & \\ [\![\overline{t} \mid [x\backslash(\overline{\mathtt{v}}, e)] :: e' \mid \pi \mid D]\!] & & \end{array}$$

4. *Exponential.* We have $x \mid e_1 :: [x\backslash(\overline{\mathtt{v}}, e)] :: e_2 \mid \pi \mid D \rightsquigarrow_{\mathtt{e}} \overline{\mathtt{v}} \mid e \mid \pi \mid D$, and:

$$\begin{array}{rcl} [\![x \mid e_1 :: [x\backslash(\overline{\mathtt{v}}, e)] :: e_2 \mid \pi \mid D]\!] & = & \\ [\![D]\!]\langle [\![\pi]\!]\langle [\![e_2]\!]\langle [\![e_1]\!]\langle x\rangle [x\backslash [\![e]\!]\langle \overline{\mathtt{v}}\rangle]\rangle\rangle\rangle & \rightsquigarrow_{\mathtt{e}} & \\ [\![D]\!]\langle [\![\pi]\!]\langle [\![e_2]\!]\langle [\![e]\!]\langle [\![e_1]\!]\langle \overline{\mathtt{v}}\rangle [x\backslash \overline{\mathtt{v}}]\rangle\rangle\rangle\rangle & \equiv^*_{\mathtt{gc}} & \\ [\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\langle \overline{\mathtt{v}}\rangle\rangle\rangle & = & \\ [\![\overline{\mathtt{v}} \mid e \mid \pi \mid D]\!] & & \end{array}$$

   We use the fact that $[\![e]\!]\langle\overline{\mathtt{v}}\rangle$ is closed by Lem. 3.39.1 to ensure that $[\![e_1]\!]$, $[\![e_2]\!]$, and $[x\backslash\overline{\mathtt{v}}]$ can be garbage collected.

Progress. Let $S = \overline{t} \mid e \mid \pi$ be a commutative normal form such that $[\![S]\!] \to s$. If $\overline{t}$ is

- *an application* $\overline{su}$. Then a $\rightsquigarrow_{\mathtt{s}_1}$ transition applies and S is not a commutative normal form, absurd.

- *an abstraction* $\mathtt{v}$. The decoding $[\![S]\!] = [\![D]\!]\langle[\![\pi]\!]\langle[\![e]\!]\langle\overline{\mathtt{v}}\rangle\rangle\rangle$ must have a multiplicative redex, because it must have a redex and $\overline{\mathtt{v}}$ is not a variable. So $\overline{\mathtt{v}}$ is applied to something, *i.e.* there must be at least one application node in $[\![D]\!]\langle[\![\pi]\!]\rangle$. Moreover, the stack π must be empty, otherwise there would be an administrative $\rightsquigarrow_{\mathtt{s}_2}$ transition, contradicting the hypothesis. So D is not empty. Let $D = ((\overline{s}, e'), \pi') :: D'$. By point 3 of Lem. 3.39, $\overline{s}$ must be a value, and a $\rightsquigarrow_{\mathtt{m}}$ transition applies.
- *a variable* x. By point 1 of Lem. 3.39, x must be bound by e, so $e = e_1 :: [x\backslash(\overline{s}, e')] :: e_2$ and a $\rightsquigarrow_{\mathtt{e}}$ transition applies.

□

3.5.5 Right-to-Left Call-by-Value: the LAM

In this section we present another adaptation to call-by-value of the KAM, a machine deemed *Leroy Abstract Machine* (LAM), implementing right-to-left call-by-value. The LAM owes its name to Leroy's ZINC machine [106], that implements right-to-left call-by-value evaluation. We introduce a new name because the ZINC is a quite more sophisticated machine than the LAM: it has a separate set of instructions to which terms are compiled, it handles arithmetic expressions, and it avoids needless closure creations in a way that it is not captured by the LAM. The LAM can be seen as a minor variation of the CEK; we present it mostly to stress the modularity of our contextual approach. We omit all the proofs because they are minimal variations on those for the CEK.

Definition 3.41 (Leroy Abstract Machine)**.** Stacks and states are like those for the CEK. The transitions of the LAM are:

$$\begin{array}{rcccccc}
\overline{t}\overline{s} & | e | & \pi & \rightsquigarrow_{\mathtt{s}_1} & \overline{s} | & e & | \mathbf{f}(\overline{t}, e) :: \pi \\
\overline{\mathtt{v}} & | e | \mathbf{f}(\overline{t}, e') :: \pi & & \rightsquigarrow_{\mathtt{s}_2} & \overline{t} | & e' & | \mathbf{a}(\overline{\mathtt{v}}, e) :: \pi \\
\lambda x.\overline{t} & | e | & \mathbf{a}(c) :: \pi & \rightsquigarrow_{\mathtt{m}} & \overline{t} | & [x\backslash c] :: e | & \pi \\
x & | e | & \pi & \rightsquigarrow_{\mathtt{e}} & \overline{t} | & e' & | \pi
\end{array}$$

where $\rightsquigarrow_{\mathtt{e}}$ takes place only if $e = e'' :: [x\backslash(\overline{t}, e')] :: e'''$.

Lemma 3.42 (LAM invariants)**.** *Let $S = \overline{s} \mid e \mid \pi$ be a LAM reachable state whose initial code $\overline{t}$ is well-named. Then:*

1. Closure*: every closure in S is closed;*
2. Subterm*: any code in S is a literal subterm of $\overline{t}$;*
3. Value*: any code in e is a value and, for every element of π of the form $\mathbf{a}(\overline{s}, e')$, $\overline{s}$ is a value;*
4. Contexts Decoding*: $[\![\pi]\!]$ and $[\![\pi]\!]\langle[\![e]\!]\rangle$ are right-to-left call-by-value evaluation contexts;*
5. Name*: any closure c in S is well-named and its names are names of $\overline{t}$ (*i.e. $\mathsf{supp}(c) \subseteq \mathtt{fv}(\overline{t})$*);*
6. Environment Size*: the length of any environment in S is bound by $|\overline{t}|$.*

Proof. Straightforward by inspection of the machine transitions. □

Theorem 3.43 (LAM distillation)**.** $(\text{LAM}, \texttt{value}^{\texttt{RL}}, \equiv, [\![\cdot]\!])$ *is a reflective distillery. In particular, on a reachable state* S *we have:*

1. Search 1*: if* $S \rightsquigarrow_{\texttt{s}_1} S'$ *then* $[\![S]\!] \equiv [\![S']\!]$*;*
2. Search 2*: if* $S \rightsquigarrow_{\texttt{s}_2} S'$ *then* $[\![S]\!] = [\![S']\!]$*.*
3. Multiplicative*: if* $S \rightsquigarrow_{\texttt{m}} S'$ *then* $[\![S]\!] \to_{\texttt{m}} [\![S']\!]$*;*
4. Exponential*: if* $S \rightsquigarrow_{\texttt{e}} S'$ *then* $[\![S]\!] \to_{\texttt{e}} \equiv [\![S']\!]$*;*

Proof. Similar to the CEK distillation (Thm. 3.36). □

3.5.6 Call-by-Need: the MAD

In this section we introduce a new abstract machine for call-by-need, deemed *Milner Abstract machine by-neeD* (MAD). The MAD arises very naturally as a reformulation of the need strategy (Def. 3.3) in the framework of distilleries. The motivations behind the introduction of a new machine are:

1. *Simplicity*: the MAD is arguably simpler than previous call-by-need machines known in the literature, in particular its distillation is very natural.
2. *Factorizing the Distillation of the Lazy KAM and of the SAM*: the study of the MAD will be followed by two sections showing how to tweak the MAD in order to obtain (simplifications of) two call-by-need machines in the literature, Cregut's Lazy KAM and Sestoft's machine (here called SAM). Expressing the Lazy KAM and the SAM as modifications of the MAD helps understanding their design, their distillation (that would otherwise look very technical), and their relationship.

The MAD uses the global environment approach of the MAM to implement memoization and the dump-like approach of the SCEK to evaluate inside explicit substitutions.

Definition 3.44 (Milner Abstract Machine by Need)**.** Terms, environments and stacks are defined as for the KAM. Dumps (D) are defined by:

$$D ::= \epsilon \mid (E, x, \pi) :: D$$

Transitions are given by:

$$\begin{array}{cccccccccc}
\bar{t}\bar{s} & | & \pi & | & D & | & E & \rightsquigarrow_{\texttt{s}_1} & \bar{t} & | \overline{s} :: \pi | & D & | & E \\
\lambda x.\bar{t} & | & \overline{s} :: \pi & | & D & | & E & \rightsquigarrow_{\texttt{m}} & \bar{t} & | \quad \pi \quad | & D & | & [x\backslash\overline{s}] :: E \\
x & | & \pi & | & D & | & E_1 :: [x\backslash\overline{t}] :: E_2 & \rightsquigarrow_{\texttt{s}_2} & \bar{t} & | \quad \epsilon \quad | & (E_1, x, \pi) :: D & | & E_2 \\
\overline{\texttt{v}} & | & \epsilon & | & (E_1, x, \pi) :: D & | & E_2 & \rightsquigarrow_{\texttt{e}} & \overline{\texttt{v}}^{\alpha} & | \quad \pi \quad | & D & | & E_1 :: [x\backslash\overline{\texttt{v}}] :: E_2
\end{array}$$

Definition 3.45 (MAD decoding)**.** The decoding of terms, environments, and stacks is defined as for the MAM. The decoding of dumps is given by:

$$[\![\epsilon]\!] \stackrel{\text{def}}{=} \Box \quad [\![(E, x, \pi) :: D]\!] \stackrel{\text{def}}{=} [\![E]\!]\langle [\![D]\!]\langle [\![\pi]\!]\langle x\rangle\rangle\rangle[x\backslash\Box]$$

The decoding of states is defined by $[\![\bar{t} \mid \pi \mid D \mid E]\!] := [\![E]\!]\langle [\![D]\!]\langle [\![\pi]\!]\langle\bar{t}\rangle\rangle\rangle$.

Note that when the code is a variable, a *search* transition should take place. The idea is that whenever the code is a variable x and the environment has the form $E_1 :: [x\backslash\bar{t}] :: E_2$, the machine should jump to evaluate $\bar{t}$, saving the prefix of the environment E_1, the variable x on which it will substitute the result of evaluating $\bar{t}$, and the stack π. This in fact corresponds to hereditarily weak head evaluation.

Lemma 3.46 (Contextual Decoding)**.** *Let D, π, and E be a dump, a stack, and a global environment of the MAD, respectively. Then $[\![D]\!]$, $[\![D]\!]\langle [\![\pi]\!]\rangle$, $[\![E]\!]\langle [\![D]\!]\rangle$, and $[\![E]\!]\langle [\![D]\!]\langle [\![\pi]\!]\rangle\rangle$ are call-by-need evaluation contexts.*

Proof. Straightforward by induction on D, and respectively on E and π, using the fact that if $\mathtt{N}$ is a call-by-need evaluation context then $\mathtt{N}[x\backslash t]$ and $\mathtt{N}\langle\Box\, t\rangle$ are also call-by-need evaluation contexts. □

The notion of *closed closure* is defined exactly as for the MAM. Given a state $S = \bar{t} \mid \pi \mid D \mid E_0$ with $D = (E_1, x_1, \pi_1) :: \ldots :: (E_n, x_n, \pi_n)$, its closures are $([\![\pi]\!]\langle\bar{t}\rangle, E_0)$ and, for each $i \in \{1, \ldots, n\}$:

$$([\![\pi_i]\!]\langle x_i\rangle, E_i :: [x_i\backslash[\![\pi_{i-1}]\!]\langle x_{i-1}\rangle] :: \ldots :: [x_1\backslash[\![\pi]\!]\langle\bar{t}\rangle] :: E_0)$$

The dynamic invariants are:

Lemma 3.47 (MAD invariants)**.** *Let $S = \bar{t} \mid \pi \mid D \mid E_0$ be a MAD reachable state whose initial code $\bar{t}$ is well-named, and such that $D = (E_1, x_1, \pi_1) :: \ldots :: (E_n, x_n, \pi_n)$. Then:*

1. Global Closure: *the closures of S are closed;*
2. Subterm: *any code in S is a literal subterm of $\bar{t}$;*
3. Names: *the closures of S are well-named.*

For the properties of the decoding function recall that the structural congruence for call-by-need ($\equiv_{\mathtt{Need}}$) is defined as the least equivalence including the axioms $\equiv_{@\mathtt{l}}$, $\equiv_{\mathtt{com}}$, and $\equiv_{[\cdot]}$.

Theorem 3.48 (MAD distillation)**.** $(\text{MAD}, \mathtt{need}, \equiv_{\mathtt{Need}}, [\![\cdot]\!])$ *is a reflective distillery. In particular, on a reachable state S we have:*

1. Search 1*: if $S \rightsquigarrow_{\mathtt{s}_1} S'$ then $[\![S]\!] = [\![S']\!]$;*
2. Search 2*: if $S \rightsquigarrow_{\mathtt{s}_2} S'$ then $[\![S]\!] = [\![S']\!]$;*
3. Multiplicative*: if $S \rightsquigarrow_{\mathtt{m}} S'$ then $[\![S]\!] \to_{\mathtt{m}} \equiv_{\mathtt{Need}} [\![S']\!]$;*

4. Exponential*: if* $S \rightsquigarrow_{\mathtt{e}} S'$ *then* $[\![S]\!] \to_{\mathtt{e}} =_\alpha [\![S']\!]$.

Proof.

1. *Search 1.*

$$[\![\overline{t}\,\overline{s} \mid \pi \mid D \mid E]\!] = [\![E]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle\overline{t}\,\overline{s}\rangle\rangle\rangle = [\![\overline{t} \mid \overline{s} :: \pi \mid D \mid E]\!]$$

2. *Search 2*:

$$\begin{array}{rcl} [\![x \mid \pi \mid D \mid E_1 :: [x\backslash\overline{t}] :: E_2]\!] & = & [\![E_2]\!]\langle[\![E_1]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle x\rangle\rangle\rangle[x\backslash\overline{t}]\rangle \\ & = & [\![\overline{t} \mid \epsilon \mid (E_1, x, \pi) :: D \mid E_2]\!] \end{array}$$

3. *Multiplicative.*

$$\begin{array}{rcll} [\![\lambda x.\overline{t} \mid \overline{s} :: \pi \mid D \mid E]\!] & = & [\![E]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle(\lambda x.\overline{t})\,\overline{s}\rangle\rangle\rangle & \to_{\mathtt{m}} \\ & & [\![E]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle\overline{t}[x\backslash\overline{s}]\rangle\rangle\rangle & \equiv_{\mathtt{Need}\ \text{Lem. 3.16}} \\ & & [\![E]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle\overline{t}\rangle\rangle[x\backslash\overline{s}]\rangle & = \\ & & [\![\overline{t} \mid \pi \mid D \mid [x\backslash\overline{s}] :: E]\!] & \end{array}$$

   Note that to apply Lem. 3.16 we use the global closure invariant, as $\overline{s}$, being on the stack, is closed by E and so $[\![D]\!]$ does not capture its free variables.

4. *Exponential.*

$$\begin{array}{rcl} [\![\overline{\mathtt{v}} \mid \epsilon \mid (E_1, x, \pi) :: D \mid E_2]\!] & = & [\![E_2]\!]\langle[\![E_1]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle x\rangle\rangle\rangle[x\backslash\overline{\mathtt{v}}]\rangle \\ & \to_{\mathtt{e}} & [\![E_2]\!]\langle[\![E_1]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle\overline{\mathtt{v}}\rangle\rangle\rangle[x\backslash\overline{\mathtt{v}}]\rangle \\ & =_\alpha & [\![E_2]\!]\langle[\![E_1]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle\overline{\mathtt{v}}^\alpha\rangle\rangle\rangle[x\backslash\overline{\mathtt{v}}]\rangle \\ & = & [\![\overline{\mathtt{v}}^\alpha \mid \pi \mid D \mid E_1 :: [x\backslash\overline{\mathtt{v}}] :: E_2]\!] \end{array}$$

Progress. Let $S = \overline{t} \mid \pi \mid D \mid E$ be a commutative normal form such that $[\![S]\!] \to s$. If $\overline{t}$ is

1. *an application* $\overline{su}$. Then a $\rightsquigarrow_{\mathtt{s}_1}$ transition applies and S is not a commutative normal form, impossible;

2. *an abstraction* $\mathtt{v}$. The decoding $[\![S]\!]$ is of the form $[\![E]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle\mathtt{v}\rangle\rangle\rangle$. The stack π and the dump D cannot both be empty, since then $[\![S]\!] = [\![E]\!]\langle\mathtt{v}\rangle$ would be normal. So either the stack is empty and a $\rightsquigarrow_{\mathtt{e}}$ transition applies, or the stack is not empty and a $\rightsquigarrow_{\mathtt{m}}$ transition applies;

3. *a variable* x. By Lem. 3.47.1 it must be bound by E, so a $\rightsquigarrow_{\mathtt{s}_2}$ transition applies, and S is not a commutative normal form, impossible.

□

Abstract Considerations on Concrete Implementations. Consider transition $\rightsquigarrow_{\mathtt{s}_2}$. Note that the saving of the prefix E_1 in the dump forces to have E implemented as a list, and so to go through E sequentially. This fact goes against the intuition that E is a store (rather than a list), and makes the MAD an unreasonable abstract machine (see the analogous considerations for the KAM and for the MAM). To solve this point, in the following sections we present the Pointing MAD, a variant of the MAD (akin to Sestoft's machine for call-by-need [?]) that avoids saving E_1 in a dump entry, and restoring the store view of the global environment. The detour is justified as follows:

1. the Pointing MAD is more involved;
2. for the complexity analysis of distillation it is easier to reason on the MAD;

Note that the issue about concrete implementations is orthogonal to the complexity analysis of the distillation process.

3.5.7 Call-by-Need: the Merged MAD

Splitting the stack of the CEK machine in two we obtained a simpler form of the SECD machine. In this section we apply to the MAD the reverse transformation. The result is a machine, deemed *Merged MAD*, having only one stack and that can be seen as a simpler version of Crégut's lazy KAM [42] (but we are rather inspired by Danvy and Zerny's presentation in [49]).

To distinguish the two kinds of objects on the stack we use a marker, as for the CEK and the LAM. Formally:

Definition 3.49 (Merged MAD)**.** Terms and environments are defined as for the MAM. The syntax for stacks is:

$$\pi ::= \epsilon \mid \mathbf{a}(\bar{t}) :: \pi \mid \mathbf{h}(E, x) :: \pi$$

where $\mathbf{a}(\bar{t})$ denotes a term to be used as an argument (as for the CEK) and $\mathbf{h}(E, x, \pi)$ is morally an entry of the dump of the MAD, where however there is no need to save the current stack. The transitions are:

$$\begin{array}{rcccccc}
\bar{t}\bar{s} & \pi & E & \rightsquigarrow_{\mathtt{s}_1} & \bar{t} & \mathbf{a}(\bar{s}) :: \pi & E \\
\lambda x.\bar{t} & \mathbf{a}(\bar{s}) :: \pi & E & \rightsquigarrow_{\mathtt{m}} & \bar{t} & \pi & [x\backslash\bar{s}] :: E \\
x & \pi & E_1 :: [x\backslash\bar{t}] :: E_2 & \rightsquigarrow_{\mathtt{s}_2} & \bar{t} & \mathbf{h}(E_1, x) :: \pi & E_2 \\
\overline{\mathtt{v}} & \mathbf{h}(E_1, x) :: \pi & E_2 & \rightsquigarrow_{\mathtt{e}} & \overline{\mathtt{v}}^{\alpha} & \pi & E_1 :: [x\backslash\overline{\mathtt{v}}] :: E_2
\end{array}$$

Definition 3.50 (Merged MAD decoding)**.** The decoding is defined as follows

$$\begin{array}{rcl}
[\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box \\
[\![[x\backslash\bar{t}] :: E]\!] & \stackrel{\text{def}}{=} & [\![E]\!]\langle\Box[x\backslash\bar{t}]\rangle \\
[\![\mathbf{h}(E, x) :: \pi]\!] & \stackrel{\text{def}}{=} & [\![E]\!]\langle[\![\pi]\!]\langle x\rangle\rangle[x\backslash\Box] \\
[\![\mathbf{a}(\bar{t}) :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle\Box\bar{t}\rangle \\
[\![\bar{t} \mid \pi \mid E]\!] & \stackrel{\text{def}}{=} & [\![E]\!]\langle[\![\pi]\!]\langle\bar{t}\rangle\rangle
\end{array}$$

Lemma 3.51 (Contextual Decoding). *Let π and E be a stack and a global environment of the Merged MAD. Then $[\![\pi]\!]$ and $[\![E]\!]\langle[\![\pi]\!]\rangle$ are call-by-need evaluation contexts.*

The dynamic invariants of the Merged MAD are exactly the same of the MAD, with respect to an analogous set of closures associated to a state (whose exact definition is omitted). The proof of the following theorem—almost identical to that of the MAD—is omitted.

Theorem 3.52 (Merged MAD Distillation). $(\text{Merged MAD}, \mathtt{need}, \equiv_{\mathtt{Need}}, [\![\cdot]\!])$ *is a reflective distillery. In particular, on a reachable state S we have:*

1. Search 1*: if $S \rightsquigarrow_{\mathtt{s}_1} S'$ then $[\![S]\!] = [\![S']\!]$;*
2. Search 2*: if $S \rightsquigarrow_{\mathtt{s}_2} S'$ then $[\![S]\!] = [\![S']\!]$;*
3. Multiplicative*: if $S \rightsquigarrow_{\mathtt{m}} S'$ then $[\![S]\!] \to_{\mathtt{m}} \equiv_{\mathtt{Need}} [\![S']\!]$;*
4. Exponential*: if $S \rightsquigarrow_{\mathtt{e}} S'$ then $[\![S]\!] \to_{\mathtt{e}} =_\alpha [\![S']\!]$.*

3.5.8 Call-by-Need: the Pointing MAD

In the MAD, the global environment is divided between the environment of the machine and the entries of the dump. On the one hand, this choice makes the decoding very natural. On the other hand, one would like to keep the global environment in just one place, to validate the intuition that it is a store rather than a list, and let the dump only collect variables and stacks. This is what we do here, exploiting the fact that variable names can be taken as pointers (see the *abstract considerations* in Sec. 3.5.2 and Sec. 3.5.6).

The new machine can be seen as a simpler version of *Sestoft's Abstract Machine* [**?**], here called SAM. It uses a new dummy constant $\square$ for the substitutions whose variable is in the dump.

Definition 3.53 (The Pointing MAD). Dumps and environments are defined as follows:

$$\begin{array}{rcl} D & ::= & \epsilon \mid (x,\pi) :: D \\ E & ::= & \epsilon \mid [x\backslash\overline{t}] :: E \mid [x\backslash\square] :: E \end{array}$$

Transitions are given by:

$$\begin{array}{cccccccccc}
\overline{t}\overline{s} & \pi & D & E & \rightsquigarrow_{\mathtt{s}_1} & \overline{t} & \overline{s} :: \pi & D & E \\
\lambda x.\overline{t} & \overline{s} :: \pi & \epsilon & E & \rightsquigarrow_{\mathtt{m}_1} & \overline{t} & \pi & \epsilon & [x\backslash\overline{s}] :: E \\
\lambda x.\overline{t} & \overline{s} :: \pi & (y,\pi') :: D & E_1 :: [y\backslash\square] :: E_2 & \rightsquigarrow_{\mathtt{m}_2} & \overline{t} & \pi & (y,\pi') :: D & E_1 :: [y\backslash\square] :: [x\backslash\overline{s}] :: E_2 \\
x & \pi & D & E_1 :: [x\backslash\overline{t}] :: E_2 & \rightsquigarrow_{\mathtt{s}_2} & \overline{t} & \epsilon & (x,\pi) :: D & E_1 :: [x\backslash\square] :: E_2 \\
\overline{\mathtt{v}} & \epsilon & (x,\pi) :: D & E_1 :: [x\backslash\square] :: E_2 & \rightsquigarrow_{\mathtt{e}} & \overline{\mathtt{v}}^{\alpha} & \pi & D & E_1 :: [x\backslash\overline{\mathtt{v}}] :: E_2
\end{array}$$

Note that there are two multiplicative transitions, that are both distilled as multiplicative steps, depending on the content of the dump. A substitution of the form $[x\backslash\square]$ is called *dumped*, and in such a situation we also say that x is dumped.

Note also that the variables of the entries in D appear in reverse order with respect to the corresponding substitutions in E. We will show that fact is an invariant, called *compatibility*.

Definition 3.54 (Compatibility $E \propto D$)**.** Compatibility $E \propto D$ between environments and dumps is defined by

1. $\epsilon \propto \epsilon$;
2. $E :: [x \backslash \overline{t}] \propto D$ if $E \propto D$;
3. $E :: [x \backslash \square] \propto (x, \pi) :: D$ if $E \propto D$.

Note that in a compatible pair the environment is always at least as long as the dump.

Definition 3.55 (Pointing MAD decoding)**.** A compatible pair $E \propto D$ decodes to a context as follows:

$$\begin{array}{rcl} [\![(E, \epsilon)]\!] & \stackrel{\text{def}}{=} & [\![E]\!] \\ [\![(E :: [x \backslash \square], (x, \pi) :: D)]\!] & \stackrel{\text{def}}{=} & [\![(E, D)]\!] \langle [\![\pi]\!] \langle x \rangle \rangle [x \backslash \square] \\ [\![(E :: [x \backslash \overline{t}], (y, \pi) :: D)]\!] & \stackrel{\text{def}}{=} & [\![(E, (y, \pi) :: D)]\!] [x \backslash \overline{t}] \end{array}$$

The decoding of a state is defined as $[\![\overline{t} \mid \pi \mid D \mid E]\!] := [\![(E, D)]\!] \langle [\![\pi]\!] \langle \overline{t} \rangle \rangle$ provided that E and D are compatible.

The analysis of the Pointing MAD is based on a complex invariant that includes compatibility plus a generalization of the global closure invariant. We need an auxiliary definition:

Definition 3.56 (Slice of an environment)**.** Given an environment E, we define its *slice* $E\!\upharpoonright$ as the sequence of substitutions after the rightmost dumped substitution. Formally:

$$\begin{array}{rcl} \epsilon\!\upharpoonright & := & \epsilon \\ (E :: [x \backslash \overline{t}])\!\upharpoonright & := & E\!\upharpoonright :: [x \backslash \overline{t}] \\ (E :: [x \backslash \square])\!\upharpoonright & := & \epsilon \end{array}$$

Moreover, if an environment E is of the form $E_1 :: [x \backslash \square] :: E_2$, we define $E\!\upharpoonright_x := E_1\!\upharpoonright :: [x \backslash \square] :: E_2$.

The notion of closed closure with global environment (Sec. 3.5.2) is extended to dummy constants $\square$ as expected.

Lemma 3.57 (Pointing MAD invariants — ♣ Lem. A.13)**.** *Let $S = \overline{t} \mid E \mid \pi \mid D$ be a Pointing MAD reachable state whose initial code $\overline{t}$ is well-named. Then:*

1. Subterm: *any code in S is a literal subterm of $\overline{t}$;*
2. Names: *the global closure of S is well-named;*
3. Dump-Environment Compatibility:
   - 3.1 *$([\![\pi]\!] \langle \overline{t} \rangle, E\!\upharpoonright)$ is closed;*
   - 3.2 *for every pair (x, π') in D, $([\![\pi']\!] \langle x \rangle, E\!\upharpoonright_x)$ is closed;*

3.3 $E \propto D$ *holds.*

4. Contextual Decoding: $[\![(E, D)]\!]$ *is a call-by-need evaluation context.*

Proof. See Lem. A.13 in the appendix. □

Theorem 3.58 (Pointing MAD distillation)**.** $(\text{Pointing MAD}, \mathtt{need}, \equiv_{\mathtt{Need}}, [\![\cdot]\!])$ *is a reflective distillery. In particular, on a reachable state* S *we have:*

1. Search*: if* $S \rightsquigarrow_{\mathtt{s}_1} S'$ *or* $S \rightsquigarrow_{\mathtt{s}_2} S'$ *then* $[\![S]\!] = [\![S']\!]$*;*
2. Multiplicative*: if* $S \rightsquigarrow_{\mathtt{m}_1} S'$ *or* $S \rightsquigarrow_{\mathtt{m}_2} S'$ *then* $[\![S]\!] \to_{\mathtt{m}} \equiv_{\mathtt{Need}} [\![S']\!]$*;*
3. Exponential*: if* $S \rightsquigarrow_{\mathtt{e}} S'$ *then* $[\![S]\!] \to_{\mathtt{e}} =_{\alpha} [\![S']\!]$*;*

Proof. Properties of the decoding:

1. *Search 1.* We have:

$$[\![\overline{t}\,\overline{s} \mid \pi \mid D \mid E]\!] \;=\; [\![(E, D)]\!]\langle [\![\pi]\!]\langle \overline{t}\,\overline{s}\rangle\rangle \;=\; [\![\overline{t} \mid \overline{s} :: \pi \mid D \mid E]\!]$$

2. *Search 2.* Note that E_2 has no dumped substitutions, since $E_1 :: [x\backslash\Box] :: E_2 \propto (x, \pi) :: D$. Then:

$$\begin{array}{ll} [\![x \mid \pi \mid D \mid E_1 :: [x\backslash\overline{t}] :: E_2]\!] & = \\ [\![E_2]\!]\langle [\![(E_1, D)]\!]\langle [\![\pi]\!]\langle x\rangle\rangle[x\backslash\overline{t}]\rangle & = \\ [\![\overline{t} \mid \epsilon \mid (x, \pi) :: D \mid E_1 :: [x\backslash\Box] :: E_2]\!] & \end{array}$$

3. *Multiplicative 1, empty dump.*

$$\begin{array}{rll} [\![\lambda x.\overline{t} \mid \overline{s} :: \pi \mid \epsilon \mid E]\!] \;=\; & [\![E]\!]\langle [\![\pi]\!]\langle (\lambda x.\overline{t})\,\overline{s}\rangle\rangle & \to_{\mathtt{m}} \\ & [\![E]\!]\langle [\![\pi]\!]\langle \overline{t}[x\backslash\overline{s}]\rangle\rangle & \equiv^{*}_{@1} \text{ Lem. 3.16} \\ & [\![E]\!]\langle [\![\pi]\!]\langle \overline{t}\rangle[x\backslash\overline{s}]\rangle & = \\ & [\![\overline{t} \mid \pi \mid \epsilon \mid [x\backslash\overline{s}] :: E]\!] & \end{array}$$

4. *Multiplicative 2, non-empty dump.*

$$\begin{array}{ll} [\![\lambda x.\overline{t} \mid \overline{s} :: \pi \mid (y, \pi') :: D \mid E_1 :: [y\backslash\Box] :: E_2]\!] & = \\ [\![E_2]\!]\langle [\![(E_1, D)]\!]\langle [\![\pi']\!]\langle y\rangle\rangle[y\backslash[\![\pi]\!]\langle (\lambda x.\overline{t})\,\overline{s}\rangle]\rangle & \to_{\mathtt{m}} \\ [\![E_2]\!]\langle [\![(E_1, D)]\!]\langle [\![\pi']\!]\langle y\rangle\rangle[y\backslash[\![\pi]\!]\langle \overline{t}[x\backslash\overline{s}]\rangle]\rangle & \equiv_{\mathtt{Need}} \text{ Lem. 3.16} \\ [\![E_2]\!]\langle [\![(E_1, D)]\!]\langle [\![\pi']\!]\langle y\rangle\rangle[y\backslash[\![\pi]\!]\langle \overline{t}\rangle][x\backslash\overline{s}]\rangle & = \\ [\![\overline{t} \mid \pi \mid (y, \pi') :: D \mid E_1 :: [y\backslash\Box] :: [x\backslash\overline{s}] :: E_2]\!] & \end{array}$$

5. *Exponential.*

$$\begin{array}{ll} [\![\overline{\mathtt{v}} \mid \epsilon \mid (x, \pi) :: D \mid E_1 :: [x\backslash\Box] :: E_2]\!] & = \\ [\![E_2]\!]\langle [\![(E_1, D)]\!]\langle [\![\pi]\!]\langle x\rangle\rangle[x\backslash\mathtt{v}]\rangle & \to_{\mathtt{e}} \\ [\![E_2]\!]\langle [\![(E_1, D)]\!]\langle [\![\pi]\!]\langle \mathtt{v}\rangle\rangle[x\backslash\mathtt{v}]\rangle & =_{\alpha} \\ [\![E_2]\!]\langle [\![(E_1, D)]\!]\langle [\![\pi]\!]\langle \mathtt{v}^{\alpha}\rangle\rangle[x\backslash\mathtt{v}]\rangle & = \\ [\![\overline{\mathtt{v}}^{\alpha} \mid \pi \mid D \mid E_1 :: [x\backslash\overline{\mathtt{v}}] :: E_2]\!] & \end{array}$$

Progress. Let $S = \overline{t} \mid \pi \mid D \mid E$ be a commutative normal form such that $[\![S]\!] \to s$. If $\overline{t}$ is

- *an application* $\overline{su}$. Then a $\rightsquigarrow_{\mathtt{s}_1}$ transition applies and S is not a commutative normal form, absurd.
- *a variable* x. By the machine invariant, x must be bound by $E\!\uparrow$. So $E = E_1 :: [x\backslash\overline{s}] :: E_2$, a $\rightsquigarrow_{\mathtt{s}_2}$ transition applies, and S is not a commutative normal form, absurd.
- *an abstraction* $\overline{\mathtt{v}}$. Two cases:
  - *The stack π is empty.* The dump D cannot be empty, since if $D = \epsilon$ we have that $[\![S]\!] = [\![e]\!]\langle\overline{\mathtt{v}}\rangle$ is normal. So $D = (x, \pi') :: D'$. By compatibility, $E = E_1 :: [x\backslash\Box] :: E_2$ and a $\rightsquigarrow_{\mathtt{e}}$ transition applies;
  - *The stack π is non-empty.* If the dump D is empty, the first case of $\rightsquigarrow_{\mathtt{m}}$ applies. If $D = (x, \pi') :: D'$, by compatibility $E = E_1 :: [x\backslash\Box] :: E_2$ and the second case of $\rightsquigarrow_{\mathtt{m}}$ applies. □

3.5.9 Strong Call-by-Name: the Strong MAM

The machine introduced in this section implements strong call-by-name, and may therefore be seen as a strong version of the MAM.

We know that the MAM performs *weak* head reduction, whose reduction contexts are (informally) of the form $\Box\, t_1 \dots t_n$. This justifies the presence of the stack $\pi = t_1 :: \dots :: t_n$, which collects the list of arguments. It is immediate to extend the MAM so that it performs full head reduction, *i.e.*, so that the head redex is reduced even if it is under an abstraction. Since head contexts are now of the form $\lambda x_1.\dots\lambda x_m.\Box t_1 \dots t_n$, we simply add a stack of abstractions $\Lambda = x_m :: \dots :: x_1$ and augment the machine with the following transition:

$$\begin{array}{c|c|c|c} \text{Abs} & \text{Code} & \text{Stack} & \text{Env} \\ \Lambda & \lambda x.\overline{t} & \epsilon & E \end{array} \quad \rightsquigarrow_{\mathtt{s}_2} \quad \begin{array}{c|c|c|c} \text{Abs} & \text{Code} & \text{Stack} & \text{Env} \\ x :: \Lambda & \overline{t} & \epsilon & E \end{array} \tag{3.4}$$

The other transitions do not affect the abstraction stack Λ.

Strong call-by-name reduction is nothing but iterated head reduction. Strong call-by-name evaluation contexts, which we formally introduced in Def. 3.5, when restricted to the pure λ-calculus (without explicit substitutions) are either of the form $\lambda x_1.\dots\lambda x_m.\Box\, t_1 \dots t_n$ as before, or of the form $\lambda x_1.\dots\lambda x_m.s\,\mathtt{C}\,t_1 \dots t_n$, where s is a neutral term and $\mathtt{C}$ is, inductively, a strong call-by-name evaluation context. As a consequence strong call-by-name evaluation contexts may be represented by stacks of triples of the form (Λ, s, π), where s is a neutral term. These stacks of triples will be called *dumps*.

The states of the machine for strong call-by-name reduction are as above but augmented with a dump and a *phase* φ, indicating whether we are executing head reduction ($\Downarrow$) or whether we are backtracking to find the starting point of the next iteration ($\Uparrow$). Besides the transitions

of the MAM, which do not touch the dump and are always in the $\Downarrow$ phase, and the transition (3.4) above, we add the following transitions:

Abs	Code	Stack	Env	Dump	Ph		Abs	Code	Stack	Env	Dump	Ph
Λ	x	π	E	D	$\Downarrow$	$\rightsquigarrow_{s_3}$	Λ	x	π	E	D	$\Uparrow$
											if $E(x) = \bot$	
$x :: \Lambda$	$\bar{t}$	ϵ	E	D	$\Uparrow$	$\rightsquigarrow_{s_5}$	Λ	$\lambda x.\bar{t}$	ϵ	E	D	$\Uparrow$
ϵ	$\bar{s}$	ϵ	E	$(\Lambda, \bar{t}, \pi) :: D$	$\Uparrow$	$\rightsquigarrow_{s_7}$	Λ	$\bar{t}\bar{s}$	π	E	D	$\Uparrow$
Λ	$\bar{t}$	$\bar{s} :: \pi$	E	D	$\Uparrow$	$\rightsquigarrow_{s_6}$	ϵ	$\bar{s}$	ϵ	E	$(\Lambda, \bar{t}, \pi) :: D$	$\Downarrow$

where $E(x) = \bot$ means that the variable x is undefined in the environment E.

In the actual machine that we define next, we merge the dump D and the abstraction stack Λ into a structure F that we call a *frame*, as to reduce the number of machine components. The analysis will however somewhat reintroduce the distinction between dump and abstraction stack. In the sequel, the reader should bear in mind that a state of the Strong MAM introduced below corresponds to a state of the machine just discussed according to the following correspondence:[3]

Discussed Machine:

Abs	Code	Stack	Env	Dump	Ph
Λ_0	$\bar{t}$	π	E	$(\Lambda_1, \bar{t}_1, \pi_1) :: \cdots :: (\Lambda_n, \bar{t}_n, \pi_n)$	φ

$\updownarrow$

Strong MAM:

Frame	Code	Stack	Env	Ph
$\Lambda_0 :: (\bar{t}_1, \pi_1) :: \Lambda_1 :: \cdots :: (\bar{t}_n, \pi_n) :: \Lambda_n$	$\bar{t}$	π	E	φ

We turn to the formal definition of the machine:

Definition 3.59 (The Strong MAM)**.** The sets of *stacks*, *environments*, *frames*, and *phases* are defined as follows:

Frames	F	$::=$	$\epsilon \mid (\bar{t}, \pi) :: F \mid x :: F$	Stacks	π	$::=$	$\epsilon \mid \bar{t} :: \pi$
Environments	E	$::=$	$\epsilon \mid [x \backslash \bar{t}] :: E \mid \rhd x :: E \mid x \lhd :: E$	Phases	φ	$::=$	$\Downarrow \mid \Uparrow$

States of the machine are 5-uples $(F, \bar{t}, \pi, E, \varphi)$. Transitions are given by:

F	$\bar{t}\bar{s}$	π	E	$\Downarrow$	$\rightsquigarrow_{\Downarrow s_1}$	F	$\bar{t}$	$\bar{s} :: \pi$	E	$\Downarrow$		
F	$\lambda x.\bar{t}$	$\bar{s} :: \pi$	E	$\Downarrow$	$\rightsquigarrow_{\mathtt{m}}$	F	$\bar{t}$	π	$[x \backslash \bar{s}] :: E$	$\Downarrow$		
F	$\lambda x.\bar{t}$	ϵ	E	$\Downarrow$	$\rightsquigarrow_{\Downarrow s_2}$	$x :: F$	$\bar{t}$	ϵ	$\rhd x :: E$	$\Downarrow$		
F	x	π	E	$\Downarrow$	$\rightsquigarrow_{\mathtt{e}}$	F	$\bar{t}^{\alpha}$	π	E	$\Downarrow$		
											if $E(x) = \bar{t}$	
F	x	π	E	$\Downarrow$	$\rightsquigarrow_{\Downarrow s_3}$	F	x	π	E	$\Uparrow$		
											if $E(x) = \rhd$	
$x :: F$	$\bar{t}$	ϵ	E	$\Uparrow$	$\rightsquigarrow_{\Uparrow s_4}$	F	$\lambda x.\bar{t}$	ϵ	$x \lhd :: E$	$\Uparrow$		
$(\bar{t}, \pi) :: F$	$\bar{s}$	ϵ	E	$\Uparrow$	$\rightsquigarrow_{\Uparrow s_5}$	F	$\bar{t}\bar{s}$	π	E	$\Uparrow$		
F	$\bar{t}$	$\bar{s} :: \pi$	E	$\Uparrow$	$\rightsquigarrow_{\Uparrow s_6}$	$(\bar{t}, \pi) :: F$	$\bar{s}$	ϵ	E	$\Downarrow$		

[3] Modulo the presence of markers of the form $x \lhd$ and $\rhd x$ in the environment, which are needed for bookkeeping purposes and were omitted here.

A few comments on the machine follow.

Scope Markers. The two transitions to evaluate and backtrack on abstractions, $\rightsquigarrow_{\Downarrow_{\mathtt{s}_2}}$ and $\rightsquigarrow_{\Uparrow_{\mathtt{s}_4}}$, add markers to delimit subenvironments associated to scopes. The marker $\rhd x$ is introduced when the machine starts evaluating under an abstraction λx, while $x\lhd$ marks the end of such a subenvironment. Note that the markers are not inspected by the machine. They are in fact needed only for the analysis, as they structure the frame and the environment of a reachable state into *weak* and *trunk* parts, allowing a simple decoding towards terms with explicit substitutions. The following notions of ordinary frames (F), weak frames (F_{w}), and trunk frames (F_{t}), and the following notions of well-formed environments (E), weak environments (E_{w}), and trunk environments (E_{t}) are used in the analysis of the machine:

Definition 3.60 (Auxiliary notions of frames and environments)**.**

Ordinary, Weak, and Trunk Frames			Well-Formed, Weak, and Trunk Environments		
F	$::=$	$F_{\mathrm{w}} \mid F_{\mathrm{t}} \mid F_{\mathrm{w}} :: F_{\mathrm{t}}$	E	$::=$	$E_{\mathrm{w}} \mid E_{\mathrm{t}} \mid E_{\mathrm{w}} :: E_{\mathrm{t}}$
F_{w}	$::=$	$\epsilon \mid (\bar{t}, \pi) :: F$	E_{w}	$::=$	$\epsilon \mid [x\backslash\bar{t}] :: E_{\mathrm{w}} \mid x\lhd :: E_{\mathrm{w}} :: \rhd x :: E'_{\mathrm{w}}$
F_{t}	$::=$	$\epsilon \mid x :: F$	E_{t}	$::=$	$\epsilon \mid \rhd x :: E$

Weak and Trunk Frames. A frame F may be uniquely decomposed as $F = F_{\mathrm{w}} :: F_{\mathrm{t}}$, where $F_{\mathrm{w}} = (\bar{t}_1, \pi_1) :: \cdots :: (\bar{t}_n, \pi_n)$ (with $n \geqslant 0$) is a *weak frame*, *i.e.* where there are no abstracted variables, and F_{t} is a *trunk frame*, *i.e.* not of the form $(\bar{t}, \pi) :: F'$ —it must either start with a variable entry or be empty. Note that here "::" denotes the concatenation of frames. We denote by $\Lambda(F)$ the set of variables abstracted in F, *i.e.* the set of x such that $F = F' :: x :: F''$.

Weak and Trunk Environments. Similarly to the frame, the environment of a reachable state has a weak/trunk structure. In contrast to frames, however, not every environment can be seen this way, but only the well-formed ones. In fact, reachable environments will be shown to be well-formed as part of the invariant of the machine. A weak environment E_{w} does not contain any open scope, *i.e.* whenever in E_{w} there is a scope opener marker ($\rhd x$) then one can also find the scope closer marker ($x\lhd$), and (globally) the closed scopes of E_{w} are well-parenthesized. A trunk environment E_{t} may instead also contain open scopes that have no closing marker in E_{t} (but not unmatched closing markers $x\lhd$).

Accessing Environments and Meta-level Garbage Collection. Fragments of the form $x\lhd :: E_{\mathrm{w}} :: \rhd x$ within an environment will essentially be ignored—this is how a simple form of garbage collection is encapsulated at the meta-level in the decoding. In particular, for a well-formed environment E we define $E(x)$ as:

$$\begin{array}{rcl@{\qquad}rcl} \epsilon(x) & := & \bot & (y\lhd :: E_{\mathrm{w}} :: \rhd y :: E)(x) & := & E(x) \\ ([x\backslash\bar{t}] :: E)(x) & := & \bar{t} & (\rhd x :: E)(x) & := & \rhd \\ ([y\backslash\bar{t}] :: E)(x) & := & E(x) & (\rhd y :: E)(x) & := & E(x) \end{array}$$

We write $\Lambda(E)$ to denote the set of variables bound to $\rhd$ by an environment E, *i.e.* those variables whose scope is not closed with $\lhd$.

Lemma 3.61 (Weak environments contain only closed scopes)**.** *If E_{w} is a weak environment then $\Lambda(E_{\mathrm{w}}) = \varnothing$.*

Abstract Considerations on Concrete Implementations. Variables are meant to be implemented as memory locations, so that the environment is simply a store, and accessing it takes constant time on a random-access machine. In particular, both the list structure of environments and the scope markers are used to define the decoding (*i.e.* for the analysis), but are not meant to be part of the actual implementation.

Compatibility. In the Strong MAM, both the frame and the environment record information about the abstractions in which evaluation is currently taking place. Clearly, such information has to be coherent, otherwise the decoding of a state becomes impossible. The following compatibility predicate captures the correlation between the structure of the frame and that of the environment.

Definition 3.62 (Compatibility $F \propto E$)**.** Compatibility $F \propto E$ between frames and environments is defined by

1. *Base*: $\epsilon \propto \epsilon$.
2. *Weak extension*: $(F_{\mathrm{w}} :: F_{\mathrm{t}}) \propto (E_{\mathrm{w}} :: E_{\mathrm{t}})$ if $F_{\mathrm{t}} \propto E_{\mathrm{t}}$.
3. *Abstraction*: $(x :: F) \propto (\rhd x :: E)$ if $F \propto E$.

Lemma 3.63 (Properties of compatibility)**.**

1. Well-Formed Environments*: if F and E are compatible then E is well-formed.*
2. Factorization*: every compatible pair $F \propto E$ can be written as $(F_{\mathrm{w}} :: F_{\mathrm{t}}) \propto (E_{\mathrm{w}} :: E_{\mathrm{t}})$ in such a way that F_{t} is of the form $F_{\mathrm{t}} = x :: F'$ if and only if E_{t} is of the form $E_{\mathrm{t}} = \rhd x :: E'$.*
3. Open Scopes Match*: $\Lambda(F) = \Lambda(E)$.*
4. Compatibility and Weak Structures Commute*: for all F_{w} and E_{w}, $F \propto E$ if and only if $(F_{\mathrm{w}} :: F) \propto (E_{\mathrm{w}} :: E)$.*

Proof. The first three items are by induction on the definition of compatible pair. Item 1. is straightforward. The base case is immediate for items 2. and 3. Let us check the two inductive cases:

1. *Weak extension*:
   - 1.1 *Factorization*: the decomposition is immediate, and the correspondence about the first variable name follows from the *i.h.*.
   - 1.2 *Open Scopes Match*: by *i.h.*, $\Lambda(F_{\mathrm{t}}) = \Lambda(E_{\mathrm{t}})$. By Lem. 3.61, $\Lambda(E_{\mathrm{w}}) = \varnothing$, and by definition $\Lambda(F_{\mathrm{w}}) = \varnothing$. Then $\Lambda(F) = \Lambda(F_{\mathrm{w}}) \cup \Lambda(F_{\mathrm{t}}) = \Lambda(F_{\mathrm{t}}) = \Lambda(E_{\mathrm{t}}) = \Lambda(E_{\mathrm{w}}) \cup \Lambda(E_{\mathrm{t}}) = \Lambda(E)$.

2. *Abstraction*

   2.1 *Factorization*: by definition $x :: F$ and $\rhd x :: E$ are a trunk frame F_{t} and a trunk environment E_{t}, respectively. given that :: is overloaded with composition, and weak trunk and environments can be empty we have $F_{\mathrm{t}} =:: F_{\mathrm{t}}$, and similarly for E_{t}, proving the decomposition property. The correspondence about the first variable name is evident.

   2.2 *Open Scopes*: $\Lambda(x :: F) = \{x\} \cup \Lambda(F) =_{\textit{i.h.}} \{x\} \cup \Lambda(E) = \Lambda(x :: E)$.

Finally, item 4. is a corollary of item 2.
Compatibility and Weak Structures Commute.

1. $\Rightarrow$) By Factorization, $F = F'_{\mathrm{w}} :: F_{\mathrm{t}}$ and $E = E'_{\mathrm{w}} :: E_{\mathrm{t}}$. By definition of compatibility, if $F \propto E$ is derivable then $F_{\mathrm{t}} \propto E_{\mathrm{t}}$ is also derivable. Now $F_{\mathrm{w}} :: F'_{\mathrm{w}}$ and $E_{\mathrm{w}} :: E'_{\mathrm{w}}$ are weak structures and so by the weak extension rule $F_{\mathrm{w}} :: F = F_{\mathrm{w}} :: F'_{\mathrm{w}} :: F_{\mathrm{t}} \propto E_{\mathrm{w}} :: E'_{\mathrm{w}} :: E_{\mathrm{t}} = E_{\mathrm{w}} :: E$.

2. $\Leftarrow$) By definition of compatibility, if $F_{\mathrm{w}} :: F = F_{\mathrm{w}} :: F'_{\mathrm{w}} :: F_{\mathrm{t}} \propto E_{\mathrm{w}} :: E'_{\mathrm{w}} :: E_{\mathrm{t}} = E_{\mathrm{w}} :: E$ is derivable then $F_{\mathrm{t}} \propto E_{\mathrm{t}}$ is also derivable, and $F = F'_{\mathrm{w}} :: F_{\mathrm{t}} \propto = E'_{\mathrm{w}} :: E_{\mathrm{t}} = E$ by applying the weak extension rule.

□

As for the previous abstract machines, we state and prove a set of dynamic invariants that hold in all reachable states:

Lemma 3.64 (Strong MAM invariants — ♣ Lem. A.14)**.** *Let $S = \varphi \mid F \mid \overline{s} \mid \pi \mid E$ be a state reachable from an initial term $\overline{t}_0$. Then:*

1. Compatibility: *F and E are compatible,* i.e. $F \propto E$.

2. Normal Form:

   2.1 Backtracking Code*: if $\varphi = \Uparrow$, then $\overline{s}$ is normal, and if π is non-empty, then $\overline{s}$ is neutral.*

   2.2 Frame*: if $F = F' :: (\overline{u}, \pi') :: F''$, then $\overline{u}$ is neutral.*

3. Backtracking Free Variables:

   3.1 Backtracking Code*: if $\varphi = \Uparrow$ then $\mathtt{fv}(\overline{s}) \subseteq \Lambda(F)$.*

   3.2 Pairs in the Frame*: if $F = F' :: (\overline{u}, \pi') :: F''$ then $\mathtt{fv}(\overline{u}) \subseteq \Lambda(F'')$.*

4. Name:

   4.1 Substitutions*: if $E = E' :: [x\backslash\overline{t}] :: E''$ then x is fresh with respect to $\overline{t}$ and E''.*

   4.2 Markers*: if $E = E' :: \rhd x :: E''$ and $F = F' :: x :: F''$ then x is fresh with respect to E'' and F'', and $E'(y) = \bot$ for any free variable y in F''.*

4.3 Abstractions*: if* $\mathbf{a}x\bar{t}$ *is a subterm of* F*,* $\bar{s}$*,* π*, or* E *then* x *may occur only in* $\bar{t}$ *and in the closed subenvironment* $x\lhd :: E_{\mathrm{w}} :: \rhd x$ *of* E*, if it exists.*

5. Closure:

5.1 Environment*: if* $E = E' :: [x\backslash\bar{t}] :: E''$ *then* $E''(y) \neq \bot$ *for all* $y \in \mathtt{fv}(\bar{t})$*.*

5.2 Code, Stack, and Frame*:* $E(x) \neq \bot$ *for any free variable* x *in* $\bar{s}$ *and in any code of* π *and* F*.*

Proof. See Section A.1.4 in the appendix. □

The definition of the decoding relies on the notion of compatible pair.

Definition 3.65 (Strong MAM decoding)**.** Let $S = (F, \bar{t}, \pi, E, \varphi)$ be a state such that $F \propto E$ is a compatible pair. Then S decodes to a state context $\mathtt{C}_S$ and a term $[\![S]\!]$ as follows:

- Weak environments:

$$\begin{array}{rcl} [\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box \\ [\![[x\backslash\bar{s}] :: E_{\mathrm{w}}]\!] & \stackrel{\text{def}}{=} & [\![E_{\mathrm{w}}]\!]\langle\Box[x\backslash\bar{s}]\rangle \\ [\![x\lhd :: E_{\mathrm{w}} :: \rhd x :: E'_{\mathrm{w}}]\!] & \stackrel{\text{def}}{=} & [\![E'_{\mathrm{w}}]\!] \end{array}$$

- Compatible pairs:

$$\begin{array}{rcl} [\![(\epsilon, \epsilon)]\!] & \stackrel{\text{def}}{=} & \Box \\ [\![((F_{\mathrm{w}} :: F_{\mathrm{t}}), (E_{\mathrm{w}} :: E_{\mathrm{t}}))]\!] & \stackrel{\text{def}}{=} & [\![(F_{\mathrm{t}}, E_{\mathrm{t}})]\!]\langle[\![E_{\mathrm{w}}]\!]\langle[\![F_{\mathrm{w}}]\!]\rangle\rangle \\ [\![((x :: F), (\rhd x :: E))]\!] & \stackrel{\text{def}}{=} & [\![(F, E)]\!]\langle\lambda x.\Box\rangle \end{array}$$

- Weak frames:

$$\begin{array}{rcl} [\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box \\ [\![(\bar{s}, \pi) :: F_{\mathrm{w}}]\!] & \stackrel{\text{def}}{=} & [\![F_{\mathrm{w}}]\!]\langle[\![\pi]\!]\langle\bar{s}\Box\rangle\rangle \end{array}$$

- Stacks:

$$\begin{array}{rcl} [\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box \\ [\![\bar{s} :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle\Box\bar{s}\rangle \end{array}$$

- States:

$$\begin{array}{rcl} \mathtt{C}_S & \stackrel{\text{def}}{=} & [\![(F, E)]\!]\langle[\![\pi]\!]\rangle \\ [\![S]\!] & \stackrel{\text{def}}{=} & \mathtt{C}_S\langle\bar{t}\rangle \end{array}$$

The following lemmas sum up the properties of the decoding.

Lemma 3.66 (Closed scopes disappear)**.** *Let* $F \propto E$ *be a compatible pair. Then* $[\![(F, (x\lhd :: E_{\mathrm{w}} :: \rhd x :: E))]\!] = [\![(F, E)]\!]$*.*

Proof. Essentially it follows from $[\![x\lhd :: E_{\mathrm{w}} :: \rhd x :: E]\!] = [\![E]\!]$. Precisely, by Lem. 3.63 F and E have, respectively, the forms $F_{\mathrm{w}} :: F_{\mathrm{t}}$ and $E'_{\mathrm{w}} :: E_{\mathrm{t}}$. Now:

$$\begin{array}{rcl} [\![(F, (x\lhd :: E_{\mathrm{w}} :: \rhd x :: E))]\!] & = & [\![((F_{\mathrm{w}} :: F_{\mathrm{t}}), (x\lhd :: E_{\mathrm{w}} :: \rhd x :: E'_{\mathrm{w}} :: E_{\mathrm{t}}))]\!] \\ & = & [\![(F_{\mathrm{t}}, E_{\mathrm{t}})]\!]\langle [\![x\lhd :: E_{\mathrm{w}} :: \rhd x :: E'_{\mathrm{w}}]\!]\langle [\![F_{\mathrm{w}}]\!]\rangle\rangle \\ & = & [\![(F_{\mathrm{t}}, E_{\mathrm{t}})]\!]\langle [\![E'_{\mathrm{w}}]\!]\langle [\![F_{\mathrm{w}}]\!]\rangle\rangle \\ & = & [\![((F_{\mathrm{w}} :: F_{\mathrm{t}}), (E'_{\mathrm{w}} :: E_{\mathrm{t}}))]\!] \\ & = & [\![(F, E)]\!] \end{array}$$

□

Lemma 3.67 (LO decoding invariant — ♣ Lem. A.16)**.** *Let $S = \langle \varphi \mid F \mid \overline{s} \mid \pi \mid E\rangle$ be a reachable state. Then $[\![(F, E)]\!]$ and $\mathtt{C}_S$ are* LO *contexts.*

Proof. See Section A.1.5 in the appendix. □

Lemma 3.68 (Decoding and structural equivalence $\equiv$)**.**

1. Stacks and substitutions commute*: if x does not occur free in π then $[\![\pi]\!]\langle t[x\backslash s]\rangle \equiv [\![\pi]\!]\langle t\rangle[x\backslash s]$;*
2. Compatible pairs absorb substitutions*: if x does not occur free in F then $[\![(F, E)]\!]\langle t[x\backslash s]\rangle \equiv [\![(F, ([x\backslash s] :: E))]\!]\langle t\rangle$.*

Proof. Straightforward by induction on π and the derivation of $F \propto E$. □

Theorem 3.69 (Strong MAM distillation)**.** *$(\text{Strong MAM}, \to_{LO}, \equiv, [\![\cdot]\!])$ is a reflective distillery. In particular:*

1. Search 1, 2, 3, 5, 6*: if $S \rightsquigarrow_{\mathtt{s}_{1,2,3,5,6}} S'$ then $[\![S]\!] = [\![S']\!]$.*
2. Search 4*: if $S \rightsquigarrow_{\mathtt{s}_4} S'$ then $[\![S]\!] \equiv_{\mathtt{gc}} [\![S']\!]$;*
3. Multiplicative*: if $S \rightsquigarrow_{\mathtt{m}} S'$ then $[\![S]\!] \to_{\mathtt{db}} \equiv_{\mathtt{name^s}} [\![S']\!]$;*
4. Exponential*: if $S \rightsquigarrow_{\mathtt{e}} S'$ then $[\![S]\!] \to_{\mathtt{ls}} [\![S']\!]$, duplicating the same subterm.*

Proof.

Properties of the decoding: Determinism of the machine follows by the name invariant (Lem. 3.64), and that of the strategy follows from the totality of the LO order (Lem. 3.7). We analyze only the interesting cases (ignoring transitions that are decoded to simple equalities).

- *Multiplicative: i.e.* $S = (F, \lambda x.\overline{t}, \overline{s} :: \pi, E, \Downarrow) \rightsquigarrow_{\mathtt{m}} (F, \overline{t}, \pi, [x\backslash\overline{s}] :: E, \Downarrow) = S'$. Note that $\mathtt{C}_{S'} = [\![(F, E)]\!]\langle [\![\pi]\!]\rangle$ is LO by the LO decoding invariant (Lem. A.16).

Moreover by the closure invariant (Lem. 3.64) x does not occur in F nor π, justifying the use of Lem. 3.68 in:

$$\begin{array}{lll}
[\![(F, \lambda x.\bar{t}, \bar{s} :: \pi, E, \Downarrow)]\!] & = & [\![(F, E)]\!]\langle [\![\bar{s} :: \pi]\!]\langle \lambda x.\bar{t}\rangle\rangle \\
& = & [\![(F, E)]\!]\langle [\![\pi]\!]\langle (\lambda x.\bar{t})\bar{s}\rangle\rangle \\
& \to_{\mathtt{db}} & [\![(F, E)]\!]\langle [\![\pi]\!]\langle \bar{t}[x\backslash\bar{s}]\rangle\rangle \\
& \equiv_{Lem.\ 3.68} & [\![(F, E)]\!]\langle [\![\pi]\!]\langle \bar{t}\rangle[x\backslash\bar{s}]\rangle \\
& \equiv_{Lem.\ 3.68} & [\![(F, ([x\backslash\bar{s}] :: E))]\!]\langle [\![\pi]\!]\langle \bar{t}\rangle\rangle \\
& = & [\![(F, \bar{t}, \pi, [x\backslash\bar{s}] :: E, \Downarrow)]\!]
\end{array}$$

– *Exponential:* $S = (F, x, \pi, E, \Downarrow) \rightsquigarrow_{\mathtt{e}} (F, \bar{t}^{\alpha}, \pi, E, \Downarrow) = S'$ with $E(x) = \bar{t}$. As before, $\mathtt{C}_S$ is LO by Lem. A.16. Moreover, $E(x) = \bar{t}$ guarantees that E, and thus $\mathtt{C}_S$, have a substitution binding x to $\bar{t}$. Finally, $\mathtt{C}_S = \mathtt{C}_{S'}$. Then

$$[\![S]\!] \;=\; \mathtt{C}_S\langle x\rangle \;\to_{\mathtt{ls}}\; \mathtt{C}_S\langle \bar{t}^{\alpha}\rangle \;=\; [\![S']\!]$$

– *Search 4:* $S = (x :: F, \bar{t}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_4} (F, \lambda x.\bar{t}, \epsilon, x\lhd :: E, \Uparrow) = S'$. By Lem. 3.64 $x :: F \propto E$, and by Lem. 3.63 $E = E_{\mathrm{w}} :: \rhd x :: E'$. Then

$$[\![((x :: F), E)]\!] \;=\; [\![((x :: F), (E_{\mathrm{w}} :: \rhd x :: E'))]\!] \;=\; [\![((x :: F), (\rhd x :: E'))]\!]\langle [\![E_{\mathrm{w}}]\!]\rangle$$

Moreover, being in a backtracking phase ($\Uparrow$) and so the backtracking closure invariant (Lem. 3.64) and the open scopes matching property (Lem. 3.63) give $\mathtt{fv}(\bar{t}) \subseteq_{Lem.\ 3.64} \Lambda(F) =_{Lem.\ 3.63} \Lambda(E_{\mathrm{w}} :: \rhd x :: E') =_{Lem.\ 3.61} \Lambda(\rhd x :: E')$, *i.e.* $[\![E_{\mathrm{w}}]\!]$ does not bind any variable in $\mathtt{fv}(\bar{t})$. Then $[\![E_{\mathrm{w}}]\!]\langle \bar{t}\rangle \equiv^{*}_{\mathtt{gc}} \bar{t}$, and

$$\begin{array}{lll}
[\![(x :: F, \bar{t}, \epsilon, E, \Uparrow)]\!] & = & [\![((x :: F), E)]\!]\langle \bar{t}\rangle \\
& = & [\![((x :: F), (E_{\mathrm{w}} :: \rhd x :: E'))]\!]\langle \bar{t}\rangle \\
& = & [\![((x :: F), (\rhd x :: E'))]\!]\langle [\![E_{\mathrm{w}}]\!]\langle \bar{t}\rangle\rangle \\
& \equiv^{*}_{\mathtt{gc}} & [\![((x :: F), (\rhd x :: E'))]\!]\langle \bar{t}\rangle \\
& = & [\![(F, E')]\!]\langle \mathbf{a}x\bar{t}\rangle \\
& =_{Lem.\ 3.66} & [\![(F, (x\lhd :: E_{\mathrm{w}} :: \rhd x :: E'))]\!]\langle \lambda x.\bar{t}\rangle \\
& = & [\![(F, (x\lhd :: E))]\!]\langle \lambda x.\bar{t}\rangle \\
& = & [\![(F, \lambda x.\bar{t}, \epsilon, x\lhd :: E, \Uparrow)]\!]
\end{array}$$

Progress:

1. *the machine cannot get stuck during the evaluation phase*: for applications and abstractions it is evident and for variables one among $\rightsquigarrow_{\mathtt{e}}$ and $\rightsquigarrow_{\Downarrow \mathtt{s}_3}$ always applies, because of the closure invariant (Lem. 3.64).
2. *final states have the form* $(\epsilon, t, \epsilon, E, \Uparrow)$, because
   2.1 by the previous consideration they are in a backtracking phase,
   2.2 if the stack is non-empty then $\rightsquigarrow_{\Uparrow \mathtt{s}_6}$ applies,
   2.3 otherwise if the frame is not empty then either $\rightsquigarrow_{\Uparrow \mathtt{s}_4}$ or $\rightsquigarrow_{\Uparrow \mathtt{s}_5}$ applies.

3. *final states decode to normal terms*: a final state $S = (\epsilon, t, \epsilon, E, \Uparrow)$ decodes to $[\![S]\!] = [\![E]\!]\langle t \rangle$ which is normal and closed by the normal form invariant and backtracking free variables invariant (Lem. 3.64).

□

3.6 Complexity Analysis

In this section we show that the length of an execution $\rho : S \leadsto_{\mathbb{M}}^{*} S'$ in each of the abstract machines can be bounded linearly by the length of the distilled derivation $[\![S]\!] \twoheadrightarrow_{\mathbb{S}} \equiv [\![S']\!]$, up to a factor $|\bar{t}|$ proportional to the size of the initial code $\bar{t}$.

Recall that principal (*i.e.* multiplicative and exponential) transitions are decoded as exactly one step in the reduction strategy, while non-principal (*i.e.* search) transitions are decoded as zero steps in the strategy. Hence, in order to obtain a bound for the length of the distilled derivation it suffices to bound the number of search steps $|\rho|_{\mathtt{s}}$ in an execution ρ in terms of:

1. the number of principal steps $|\rho|_{\neg \mathtt{s}}$,
2. the size $|\bar{t}|$ of the initial code $\bar{t}$.

The analysis only concerns the machines, but via the distillation theorems it expresses the length of the machine executions as a linear function of the length of the distilled derivations in the strategy. For every distillery, we will prove that the relationship is linear in both parameters, namely $|\rho|_{\mathtt{s}} \in O((|\bar{t}| + 1) \cdot |\rho|_{\neg \mathtt{s}})$ holds.

Definition 3.70. Let $\mathbb{M}$ be a distilled abstract machine and $\rho : S \leadsto_{\mathbb{M}}^{*} S'$ be an execution of initial code $\bar{t}$. The machine $\mathbb{M}$ is:

1. **Locally linear** if whenever $S' \leadsto_{\mathtt{s}}^{k} S''$ then $k \in O(|\bar{t}|)$.
2. **Globally bilinear** if $|\rho|_{\mathtt{s}} \in O((|\bar{t}| + 1) \cdot |\rho|_{\neg \mathtt{s}})$.

The following result ensures that local linearity is a sufficient condition for global bilinearity.

Proposition 3.71 (Locally Linear ⇒ Globally Bilinear)**.** *Let $\mathbb{M}$ be a locally linear distilled abstract machine, and ρ an execution of initial code $\bar{t}$. Then $\mathbb{M}$ is globally bilinear.*

Proof. The execution ρ can be written uniquely as $\leadsto_{\mathtt{s}}^{k_1} \leadsto_{\neg \mathtt{s}}^{h_1} \dots \leadsto_{\mathtt{s}}^{k_m} \leadsto_{\neg \mathtt{s}}^{h_m}$. By hypothesis $k_i = O(|\bar{t}|)$ for every $i \in \{1, \dots, m\}$. From $m \leqslant |\rho|_{\neg \mathtt{s}}$ follows that $|\rho|_{\mathtt{s}} = O(|\bar{t}| \cdot |\rho|_{\neg \mathtt{s}})$. We conclude with $|\rho| = |\rho|_{\neg \mathtt{s}} + |\rho|_{\mathtt{s}} = |\rho|_{\neg \mathtt{s}} + O(|\bar{t}| \cdot |\rho|_{\neg \mathtt{s}}) = O((|\bar{t}| + 1) \cdot |\rho|_{\neg \mathtt{s}})$. □

3.6.1 Call-by-name and call-by-value

Call-by-name and call-by-value machines are easily seen to be locally linear, and thus globally bilinear.

Theorem 3.72 (Bilinearity for call-by-name and call-by-value)**.** *The distilleries for the KAM, MAM, CEK, SCEK, and LAM are locally linear, and so also globally bilinear.*

Proof.

1. *KAM/MAM.* Immediate: $\rightsquigarrow_{\mathtt{s}}$ reduces the size of the code, that is bounded by $|\overline{t}|$ by the subterm invariant (Lem. 3.25/Lem. 3.31).

2. *CEK.* Consider the following measure for states:

$$\#(\overline{s} \mid e \mid \pi) := \begin{cases} |\overline{s}| + |\overline{u}| & \text{if } \pi = \mathbf{a}(\overline{u}, e') :: \pi' \\ |\overline{s}| & \text{otherwise} \end{cases}$$

   By direct inspection of the rules, it can be seen that both $\rightsquigarrow_{\mathtt{s}_1}$ and $\rightsquigarrow_{\mathtt{s}_2}$ transitions decrease the value of $\#$ for CEK states, and so the relation $\rightsquigarrow_{\mathtt{s}_1} \cup \rightsquigarrow_{\mathtt{s}_2}$ terminates (on reachable states). Moreover, both $|\overline{s}|$ and $|\overline{u}|$ are bounded by $|\overline{t}|$ by the subterm invariant (Lem. 3.35), and so $k \leqslant 2 \cdot |\overline{t}| = O(|\overline{t}|)$.

3. *SCEK.* As for the CEK, using the corresponding subterm invariant (Lem. 3.39) and the following measure:

$$\#(\overline{s} \mid e \mid \pi \mid D) := \begin{cases} |\overline{s}| + |\overline{u}| & \text{if } \pi = (\overline{u}, e') :: \pi' \\ |\overline{s}| & \text{otherwise} \end{cases}$$

4. *LAM.* As for the CEK, using the corresponding subterm invariant (Lem. 3.42) and the following measure:

$$\#(\overline{s} \mid e \mid \pi) := \begin{cases} |\overline{s}| + |\overline{u}| & \text{if } \pi = \mathbf{f}(\overline{u}, e') :: \pi' \\ |\overline{s}| & \text{otherwise} \end{cases}$$

□

3.6.2 Call-by-need

Call-by-need machines are not locally linear, because a sequence of $\rightsquigarrow_{\mathtt{s}_2}$ steps can be as long as the global environment E, that is not bound by $|\overline{t}|$ but only by the number $|\rho|_{\neg\mathtt{s}}$ of preceding principal transitions (as for the MAM). Adapting the previous reasoning to this other bound would only show that globally $|\rho|_{\mathtt{s}}$ is quadratic in $|\rho|_{\neg\mathtt{s}}$, not linear. However, being locally linear is not a necessary condition for global bilinearity. In fact, call-by-need machines are globally bilinear. The key observation is that $|\rho|_{s_2}$ is not only locally but also globally bound by $|\rho|_p$, as the next lemma formalizes.

We treat the MAD. The reasoning for the Merged/Pointing MAD is analogous. Define $|\epsilon| := 0$ and $|(E, x, \pi) :: D| := 1 + |D|$.

Lemma 3.73. *Let $S = \overline{t} \mid \pi \mid D \mid E$ be a MAD state, reached by the execution ρ. Then:*

1. $|\rho|_{s_2} = |\rho|_e + |D|$
2. $|E| + |D| \leqslant |\rho|_m$
3. $|\rho|_{s_2} \leqslant |\rho|_e + |\rho|_m = |\rho|_p$

Proof.

1. Immediate, as $\rightsquigarrow_{\mathtt{s}_2}$ is the only transition that pushes elements on D and $\rightsquigarrow_{\mathtt{e}}$ is the only transition that pops them.
2. The only rule that produces substitutions is $\rightsquigarrow_{\mathtt{m}}$. Note that 1) $\rightsquigarrow_{\mathtt{s}_2}$ and $\rightsquigarrow_{\mathtt{e}}$ preserve the global number of substitutions in a state; 2) E and D are made out of substitutions, if one considers every entry (E, x, π) of the dump as a substitution on x (and so the statement follows); 3) the inequality is given by the fact that an entry of the dump includes an environment (counting for many substitutions).
3. Substitute item 2 in item 1.

□

Theorem 3.74 (Bilinearity for call-by-need)**.** *The distillery for the MAD is globally bilinear.*

Proof. Let ρ be an execution of initial code $\overline{t}$. Define $\rightarrow_{\neg s_1} := \rightsquigarrow_{\mathtt{e}} \cup \rightsquigarrow_{\mathtt{m}} \cup \rightsquigarrow_{\mathtt{s}_2}$ and write $|\rho|_{\neg s_1}$ to stand for the number of its steps in ρ. We estimate $\rightsquigarrow_{\mathtt{s}} := \rightsquigarrow_{\mathtt{s}_1} \cup \rightsquigarrow_{\mathtt{s}_2}$ by studying its components separately. For $\rightsquigarrow_{\mathtt{s}_2}$, Lem. 3.73.3 proves $|\rho|_{s_2} \leqslant |\rho|_p = O(|\rho|_p)$. For $\rightsquigarrow_{\mathtt{s}_1}$, as for the KAM, the length of a maximal $\rightsquigarrow_{\mathtt{s}_1}$ subsequence of ρ is bounded by $|\overline{t}|$. The number of $\rightsquigarrow_{\mathtt{s}_1}$ maximal subsequences of ρ is bounded by $|\rho|_{\neg s_1}$, that by Lem. 3.73.3 is linear in $O(|\rho|_p)$. Then $|\rho|_{s_1} = O(|\overline{t}| \cdot |\rho|_{\neg \mathtt{s}})$. Summing up,

$$|\rho|_{s_2} + |\rho|_{s_1} = O(|\rho|_p) + O(|\overline{t}| \cdot |\rho|_{\neg \mathtt{s}}) = O((|\overline{t}| + 1) \cdot |\rho|_{\neg \mathtt{s}})$$

□

3.6.3 Strong call-by-name

The complexity analysis of the strong MAM requires a further invariant, bounding the size of the duplicated subterms. In this subsection, we say that $\overline{s}$ is a *subterm* of $\overline{t}$ if it does so *up to variable names*, both free and bound. More precisely: define t^- as t in which all variables (including those appearing in binders) are replaced by a fixed symbol $*$. Then, we will consider s to be a subterm of t whenever s^- is a subterm of t^- in the usual sense. The key property ensured by this definition is that the size $|\overline{s}|$ of $\overline{s}$ is bounded by $|\overline{t}|$.

Lemma 3.75 (Subterm invariant)**.** *Let ρ be an execution from an initial code $\overline{t}$. Every code duplicated along ρ using $\rightsquigarrow_{\mathtt{e}}$ is a subterm of $\overline{t}$.*

Proof. Straightforward by inspection of the machine transitions. □

The following invariant provides a new proof of the subterm property of linear LO reduction (first proved in [11]):

Lemma 3.76 (Subterm Property for $\to_{\mathsf{LO}}$)**.** *Let π be a $\to_{LO}$-derivation from an initial term t. Every term duplicated along π using a $\to_{\mathtt{ls}}$ is a subterm of t.*

Proof. Easy by the subterm invariant (Lem. 3.64) via the case of an exponential transition of the distillation theorem (Thm. 3.69). □

Finally, the following theorem ensures that the strong MAM is globally bilinear. Let us stress that, despite the simplicity of the reasoning, the analysis is subtle as the length of backtracking phases can be bound only *globally* by the previous work done on evaluation phases.

Theorem 3.77 (Bilinearity for strong call-by-name)**.** *The distillery for the strong MAM is globally bilinear. More precisely, given an execution $\rho : S \rightsquigarrow_{\mathbb{M}}^{*} S'$ from an initial state of code t then:*

1. Search evaluation steps are bilinear*: $|\rho|_{\Downarrow \mathtt{s}} \leqslant (1 + |\rho|_e) \cdot |t|$.*
2. Search evaluation bounds backtracking*: $|\rho|_{\Uparrow \mathtt{s}} \leqslant 2 \cdot |\rho|_{\Downarrow \mathtt{s}}$.*
3. Search steps are bilinear*: $|\rho|_s \leqslant 3 \cdot (1 + |\rho|_e) \cdot |t|$.*

Proof.

1. We prove a slightly stronger statement, namely $|\rho|_{\Downarrow \mathtt{s}} + |\rho|_m \leqslant (1 + |\rho|_e) \cdot |t|$, by means of the following notion of size for stacks/frames/states:

$$\begin{array}{rclcrcl} |\epsilon| & := & 0 & & |x :: F| & := & |F| \\ |\bar{t} :: \pi| & := & |\bar{t}| + |\pi| & & |(\bar{t}, \pi) :: F| & := & |\pi| + |F| \\ |(F, \bar{t}, \pi, E, \Downarrow)| & := & |F| + |\pi| + |\bar{t}| & & |(F, \bar{t}, \pi, E, \Uparrow)| & := & |F| + |\pi| \end{array}$$

   By direct inspection of the rules of the machine it can be checked that:

   - *Exponentials increase the size*: if $S \rightsquigarrow_{\mathtt{e}} S'$ is an exponential transition, then $|S'| \leqslant |S| + |t|$ where $|t|$ is the size of the initial term; this is a consequence of the fact that exponential steps retrieve a piece of code from the environment, which is a subterm of the initial term by Lem. 3.75;
   - *Non-Exponential evaluation transitions decrease the size*: if $S \rightsquigarrow_a S'$ with $a \in \{\mathtt{m}, \Downarrow \mathtt{s}_1, \Downarrow \mathtt{s}_2, \Downarrow \mathtt{s}_3\}$ then $|S'| < |S|$;
   - *Backtracking transitions do not change the size*: if $S \rightsquigarrow_a S'$ with $a \in \{\Uparrow \mathtt{s}_4, \Uparrow \mathtt{s}_5, \Uparrow \mathtt{s}_6\}$ then $|S'| = |S|$.

   Then a straightforward induction on $|\rho|$ shows that

$$|S'| \leqslant |S| + |\rho|_e \cdot |t| - |\rho|_{\Downarrow \mathtt{s}} - |\rho|_m$$

   i.e. that $|\rho|_{\Downarrow \mathtt{s}} + |\rho|_m \leqslant |S| + |\rho|_e \cdot |t| - |S'|$.

Now note that $|\cdot|$ is always non-negative and that since S is initial we have $|S| = |t|$. We can then conclude with

$$\begin{array}{rcl} |\rho|_{\Downarrow \mathtt{s}} + |\rho|_m & \leqslant & |S| + |\rho|_e \cdot |t| - |S'| \\ & \leqslant & |S| + |\rho|_e \cdot |t| \quad = \quad |t| + |\rho|_e \cdot |t| \quad = \quad (1 + |\rho|_e) \cdot |t| \end{array}$$

2. We have to estimate $|\rho|_{\Uparrow \mathtt{s}} = |\rho|_{\Uparrow \mathtt{s}_4} + |\rho|_{\Uparrow \mathtt{s}_5} + |\rho|_{\Uparrow \mathtt{s}_6}$. Note that:

   2.1 $|\rho|_{\Uparrow \mathtt{s}_4} \leqslant |\rho|_{\Downarrow \mathtt{s}_2}$, as $\rightsquigarrow_{\Uparrow \mathtt{s}_4}$ pops variables from F, pushed only by $\rightsquigarrow_{\Uparrow \mathtt{s}_4}$;

   2.2 $|\rho|_{\Uparrow \mathtt{s}_5} \leqslant |\rho|_{\Uparrow \mathtt{s}_6}$, as $\rightsquigarrow_{\Uparrow \mathtt{s}_5}$ pops pairs $(\bar{t}, \pi)$ from F, pushed only by $\rightsquigarrow_{\Uparrow \mathtt{s}_6}$;

   2.3 $|\rho|_{\Uparrow \mathtt{s}_6} \leqslant |\rho|_{\Downarrow \mathtt{s}_3}$, as $\rightsquigarrow_{\Uparrow \mathtt{s}_6}$ ends backtracking phases, started only by $\rightsquigarrow_{\Downarrow \mathtt{s}_3}$.

   Then $|\rho|_{\Uparrow \mathtt{s}} \leqslant |\rho|_{\Downarrow \mathtt{s}_2} + 2|\rho|_{\Downarrow \mathtt{s}_3} \leqslant 2|\rho|_{\Downarrow \mathtt{s}}$.

3. We have $|\rho|_s = |\rho|_{\Downarrow \mathtt{s}} + |\rho|_{\Uparrow \mathtt{s}} \leqslant_{P.2} = |\rho|_{\Downarrow \mathtt{s}} + 2|\rho|_{\Downarrow \mathtt{s}} =_{P.1} 3 \cdot (1 + |\rho|_e) \cdot |t|$.

Finally, every transition but $\rightsquigarrow_{\mathtt{e}}$ takes a constant time on a RAM. The renaming in a $\rightsquigarrow_{\mathtt{e}}$ step is instead linear in $|\bar{t}|$, by the subterm invariant (Lem. 3.75). □

Chapter 4

Foundations of Strong Call-by-Need

4.1 Introduction

4.1.1 Call-by-Need for Weak Reduction

Mainstream programming languages evaluate programs using *weak reduction*, *i.e.* the body of a function is not evaluated until the function is applied. Suppose for example that we write a function definition:

```
f x = 2 * 3 + x
```

and we evaluate the expression `f` in a typical programming language like OCaml or Haskell. Then the multiplication `2 * 3` will not be performed. Rather, the function `f` itself will be the final answer.

It also makes sense in principle to evaluate the body of a function before applying it. For instance the program above can be transformed into the presumably equivalent one:

```
f x = 6 + x
```

by performing the multiplication `2 * 3`. However, in a typical setting, the body of a function in runtime is not represented by a tree-like expression, but by a sequence of machine instructions, which means that this kind of program transformation corresponds to *preprocessing* or *compile time optimization*, rather than evaluation.

In the λ-calculus, weak reduction is characterized formally by forbidding the congruence rule ξ:

$$\frac{t \to t'}{\lambda x.t \to \lambda x.t'}\,\xi$$

This means that a term like $\lambda x.(\lambda y.y)x$ cannot be evaluated further, even though it contains a redex. When one evaluates a term in the λ-calculus using weak reduction, one does not obtain, in general, a normal form as a result. The set of answers is instead the set of *weak head normal forms*.

Definition 4.1 (Weak head normal form)**.** A λ-term is in weak head normal form if it is of the form $\lambda x.t$, or of the form $x\,t_1\,\ldots\,t_n$.

The goal of this chapter is to extend the *lazy evaluation* mechanism, an evaluation strategy that implements weak reduction, originally proposed by Wadsworth [145], to the setting of *strong reduction*. The (weak) call-by-need strategy is based on the two following principles:

1. **Laziness.** One should only perform steps that are *needed* to obtain a WHNF. For example, a step like $(\lambda x.y)\, t \to (\lambda x.y)\, t'$, internal to the argument, is not needed, as one may simply contract the redex at the root $(\lambda x.y)\, t \to y$ to obtain a WHNF.

   In this respect, call-by-need is similar to call-by-name in that an argument is only evaluated *if needed*, and it improves the situation over call-by-value (in which an argument is always evaluated, even if it is not needed).

2. **Sharing.** The computational work of evaluating an argument should be *shared* among the copies of the argument. For example, let $t := I\, I$ where $I = \lambda x.x$ is the identity function, and note that $t \to I$. Then a step like $(\lambda x.x\, x)\, t \to t\, t$, if implemented naïvely by syntactically copying the term t twice, results in the duplication of the computational work required to perform the step $t \to I$:

   $$(\lambda x.x\, x)\, t \to t\, t \overset{1}{\to} I\, t \to t \overset{2}{\to} I$$

   In contrast, in call-by-need, when the function $\lambda x.x\, x$ is applied to the argument t, the two occurrences of x become bound to a single copy of the term t. In this way, as soon as t is evaluated, both copies of x hold a reference to the same result, and computational work is not duplicated. The sharing may be represented, as we do in this thesis, using the notation of *explicit substitutions*:

   $$(\lambda x.x\, x)\, t \to (x\, x)[x\backslash t] \to (x\, x)[x\backslash I] \to (I\, x)[x\backslash I] \to x[x\backslash I] \to I[x\backslash I] \to I$$

   In this respect, call-by-need is similar to call-by-value in that arguments are evaluated *at most once*, and it improves the situation over call-by-name (in which an argument may be evaluated many times, once per each of its copies).

We remark that some authors may make a distinction between *lazy evaluation* and *call-by-need*, the former referring only to the deferral of the evaluation of expressions, and the latter incorporating also the notion of sharing. In our work, in accordance with existing literature (*e.g.* [12, 113, 13]), we speak of lazy evaluation and (weak) call-by-need as synonyms.

The weak call-by-need strategy provides various benefits over call-by-name and call-by-value. We briefly discuss four aspects: *efficiency*, *expressiveness*, *ease of reasoning* and *declarativity*.

Efficiency. Call-by-need may represent an exponential improvement in efficiency, with respect to call-by-name. For example, consider the family of terms $\{t_n \mid n \in \mathbb{N}\}$, defined recursively on n by:

$$\begin{array}{rcl} t_0 & \overset{\text{def}}{=} & \lambda x.x \\ t_{n+1} & \overset{\text{def}}{=} & (\lambda x.xx)\, t_n \end{array}$$

Then one can see that, in *call-by-name*, evaluating t_n requires a number of steps exponential in n. More precisely, we have that t_n reduces to the identity $\lambda x.x$ in exactly $2^{n+1} - 2$ steps. To see this, proceed inductively. Note that $\lambda x.x$ reduces to $\lambda x.x$ in 0 steps, and:

$$\begin{array}{rcll} t_{n+1} & = & (\lambda x.xx)\, t_n & \to \; t_n\, t_n \\ & & & \xrightarrow{2^{n+1}-2 \text{ steps, by } i.h.} \; (\lambda x.x)\, t_n \\ & & & \to \; t_n \\ & & & \xrightarrow{2^{n+1}-2 \text{ steps, by } i.h.} \; \lambda x.x \end{array}$$

So t_{n+1} reduces to $\lambda x.x$ in exactly $2^{n+2} - 2$ steps.

On the other hand, in *call-by-need*, evaluating t_n requires a number of steps linear in n. More precisely, t_n reduces to the identity $\lambda x.x$ in exactly $5n$ steps[1].

$$\begin{array}{rcll} t_{n+1} & = & (\lambda x.xx)\, t_n & \to \; (xx)[x \backslash t_n] \\ & & & \xrightarrow{5n \text{ steps, by } i.h.} \; (xx)[x \backslash \lambda y.y] \\ & & & \to \; ((\lambda y.y)x)[x \backslash \lambda y.y] \\ & & & \to \; y[y \backslash x][x \backslash \lambda y.y] \\ & & & \to \; y[y \backslash \lambda z.z][x \backslash \lambda y.y] \\ & & & \to \; \lambda z.z \end{array}$$

So t_{n+1} reduces to $\lambda x.x$ in exactly $5(n+1)$ steps.

Expressiveness. Call-by-need allows one to write programs in a style that would not be possible, or convenient, in a more traditional setting with call-by-value. For example, John Hughes [80] describes an architecture for a game-playing engine implementing the minimax decision procedure. The rough idea is that the problem can be modularly decomposed into two subproblems as follows:

1. A function `gametree : Position → Tree Position`, representing the potentially infinite *game tree* starting from the given position. The nodes of the tree are positions in the game and the edges represent moves.

2. A function `minimax : Tree Position → Position`, which determines the best move.

Modularity is achieved thanks to lazy evaluation, which allows the programmer to handle infinite data structures effortlessly, without having to explicitly resort to representing suspended computations as thunks (using constructs such as Scheme's `delay` and `force`).

There are many other well-known examples of the possibilities that lazy evaluation may enable. For example Chris Okasaki [123] shows how to exploit *memoization*, *i.e.* sharing, to implement efficient immutable data structures.

Ease of reasoning. One benefit of lazy evaluation is that it allows to reason about programs equationally. For example, consider the following set of definitions:

[1] In the last step we perform garbage collection implicitly for the sake of clarity. But note that, in the explicit substitution calculi in this thesis, garbage collection is explicit.

```
loop = loop
f x = 1
```

In a programming language using call-by-need evaluation the equality `f loop = 1` holds, while in a programming language using call-by-value evaluation `f loop` is non-terminating. Being able to reason equationally is convenient to prove properties about programs and to derive programs by applying mechanical transformations to existing programs, which may be part, for example, of the optimization phase of a compiler.

The property that one may actually "reason equationally" can be expressed formally as the *completeness* property of call-by-need, which states that if a term t is interconvertible with an answer s, then evaluating t using the call-by-need strategy also leads to an answer. Completeness for *strong* call-by-need is the main result of this chapter.

Declarativity. Declarativity is not a binary property of programs, but more of a *continuous spectrum*, a program being *less declarative* when it resembles a low-level description of a procedure that solves a problem, and *more declarative* when it resembles an abstract specification of the problem.

Lazy evaluation allows to write arguably more declarative programs, in the sense that the order in which expressions are evaluated is not prescribed by the way in which they were written by the programmer. Rather, expressions are evaluated only when they are actually required for the computation to proceed. For example, consider a function definition like the following:

```
f x y = if y == 0
          then x
          else z
  where z = x / y
```

In a programming language using call-by-value evaluation the expression `x / y` is evaluated unconditionally, which leads to an error if `y` equals `0`. Contrast this with call-by-need, in which the evaluation of `x / y` is only triggered when the value of the variable `z` is required.

Formal Definition of Weak Call-by-Need

Wadsworth first proposed call-by-need as an implementation technique for the pure λ-calculus [145] in the 1970s. Later, Ariola and Felleisen [12], and independently at around the same time in the 1990s, Maraist, Odersky and Wadler [113], proposed a different way of defining call-by-need evaluation[2]. Their approach is based on a calculus whose operational semantics follows a call-by-need discipline. This results in the behavior of the source language matching more closely the behavior of its actual implementation.

Following this approach, we study calculi based on a call-by-need evaluation discipline. In the following, we recall the definition of the weak call-by-need strategy that we use in

[2]In fact these two independent works are combined in a joint paper [13].

this chapter (and also in Chapter 3). The weak call-by-need strategy was originally proposed in [4], and it is based on the technology of *explicit substitutions at a distance*, closely related with the Linear Substitution Calculus.

Definition 4.2 (The weak call-by-need strategy)**.** The sets of *terms* ($\mathcal{T}$), *values*, *answers*, *(full) contexts*, *substitution contexts*, and *weak evaluation contexts* are given by the grammars:

Terms	t	$::=$	$x \mid \lambda x.t \mid t\,t \mid t[x\backslash t]$
Values	$\mathtt{v}$	$::=$	$\lambda x.t$
Answers	a	$::=$	$\mathtt{vL}$
(Full) contexts	$\mathtt{C}$	$::=$	$\Box \mid \lambda x.\mathtt{C} \mid \mathtt{C}\,t \mid t\,\mathtt{C} \mid \mathtt{C}[x\backslash t] \mid t[x\backslash \mathtt{C}]$
Substitution contexts	$\mathtt{L}$	$::=$	$\Box \mid \mathtt{L}[x\backslash t]$
Weak evaluation contexts	$\mathtt{E}$	$::=$	$\Box \mid \mathtt{E}\,t \mid \mathtt{E}[x\backslash t] \mid \mathtt{E}\langle\!\langle x \rangle\!\rangle [x\backslash \mathtt{E}]$

The weak call-by-need strategy $\overset{\mathsf{W}}{\rightsquigarrow}$ is given by the following rewriting rules, closed by weak evaluation contexts:

$$\begin{array}{rcl} (\lambda x.t)\mathtt{L}\,s & \overset{\mathsf{W}}{\rightsquigarrow}_{\mathtt{db}} & t[x\backslash s]\mathtt{L} \\ \mathtt{E}\langle\!\langle x \rangle\!\rangle [x\backslash \mathtt{vL}] & \overset{\mathsf{W}}{\rightsquigarrow}_{\mathtt{lsv}} & \mathtt{E}\langle\!\langle \mathtt{v} \rangle\!\rangle [x\backslash \mathtt{v}]\mathtt{L} \end{array}$$

In this chapter, we use squiggly arrows like "$\rightsquigarrow$" to denote reduction strategies, which are usually deterministic, in contrast with typical arrows "$\to$", which represent a (non-deterministic) orientation of an equational theory.

Example 4.3. *The following is a reduction in weak call-by-need:*

$$\begin{array}{rcl} (\lambda x.xx)((\lambda y.y)(\lambda z.z)) & \overset{\mathsf{W}}{\rightsquigarrow} & (xx)[x\backslash(\lambda y.y)(\lambda z.z)] \\ & \overset{\mathsf{W}}{\rightsquigarrow} & (xx)[x\backslash y[y\backslash\lambda z.z]] \\ & \overset{\mathsf{W}}{\rightsquigarrow} & (xx)[x\backslash(\lambda z.z)[y\backslash\lambda z.z]] \\ & \overset{\mathsf{W}}{\rightsquigarrow} & ((\lambda z.z)x)[x\backslash\lambda z.z][y\backslash\lambda z.z] \\ & \overset{\mathsf{W}}{\rightsquigarrow} & z[z\backslash x][x\backslash\lambda z.z][y\backslash\lambda z.z] \\ & \overset{\mathsf{W}}{\rightsquigarrow} & z[z\backslash\lambda z.z][x\backslash\lambda z.z][y\backslash\lambda z.z] \\ & \overset{\mathsf{W}}{\rightsquigarrow} & (\lambda z.z)[z\backslash\lambda z.z][x\backslash\lambda z.z][y\backslash\lambda z.z] \end{array}$$

Observe that the final result is an answer, and that the strategy does not incorporate any kind of garbage collection rule.

4.1.2 Call-by-Need for Strong Reduction

As we have stated, our goal in this chapter is to extend the weak call-by-need strategy to *strong reduction*. In contrast with mainstream programming languages, which use weak reduction, functional programming languages with dependent types —including proof assistants based on dependent type theory— use strong reduction.

Let us exemplify why a type checker for dependent type theory may need to use strong reduction. In dependent type theory, types are allowed to depend on terms. For example, suppose that the type constructor: $\mathtt{Vec} : \mathbb{N} \to \mathtt{Type}$ represents the type family of vectors

of integers, so that, for a given natural number $n : \mathbb{N}$ the expression `Vec` n denotes the type of vectors of length n. Then one may define a function to append two vectors as follows:

```
append :  (n :  ℕ) → (m :  ℕ) → Vec n → Vec m → Vec (n + m)
append zero    m nil        w = w
append (suc n) m (cons x v) w = cons x (append n m v w)
```

In order to accept this definition, the type checker has to verify that the left and the right-hand sides of all the equations have the same type. In the case of the first equation, its left-hand side is of type `Vec (0 + n)`, whereas its right-hand side is of type `Vec n`. Note that these types are not syntactically equal, rather they are *interconvertible*, up to computational reduction rules. Determining that these types are interconvertible may be achieved by evaluating `0 + n` on one hand and `n` on the other, and checking whether the same normal form is reached. This example is captured by means of a general typing rule called *conversion*:

$$\frac{\Gamma \vdash A \quad A \equiv B}{\Gamma \vdash B}\ \textsc{Conv}$$

The judgement $A \equiv B$ establishes that types A and B are equivalent. Typically, this means that A and B are interconvertible, up to computation rules like the β-reduction rule. A simple decision procedure to determine whether $A \equiv B$ holds consists in evaluating A and B to normal form and then comparing their results syntactically. Given that A and B may contain abstractions and free variables, this procedure must use strong reduction.

In this chapter, our goal is to develop the foundations for a correct and efficient strong reduction strategy. The mechanism to decide $A \equiv B$ implemented by practical proof assistants, such as the Coq proof assistant, is more complex than the naïve algorithm proposed above, and it uses a large set of finely tuned heuristics. In the particular case of Coq—at least at the moment of writing this thesis—this mechanism has not been defined other than in the actual OCaml source code of Coq, and it has not been proven correct. Even though there is a significant gap between the complexity of current implementations of proof assistants and the comparatively minimalistic formalisms studied in this thesis, we are certain that implementers and users could benefit from a foundational study of strong reduction.

As a general remark, evaluation strategies for strong reduction are not as well studied in the literature as strategies for weak reduction. One notable exception is the work of Grégoire and Leroy [66], who have proposed a strong normalization function that consists in iterating the weak call-by-value strategy on terms possibly containing free variables.

Our starting point is the observation that rather than iterating call-by-value, one should consider an appropriate notion of call-by-need that computes strong normal forms of open terms. As a matter of fact, we propose a strategy that computes strong normal forms by following a call-by-need discipline.

Defining a strong call-by-need strategy, even before attempting to state or prove any theorems, is a non-obvious task. Let us write $\overset{\mathsf{S}}{\rightsquigarrow}$ for such a strategy. Recall also that, if t is a

term with explicit substitutions, we write $t^\diamond$ for the λ-term that results from the *unfolding* of all the explicit substitutions in t, so for example:

$$(xx)[x\backslash yz][y\backslash \lambda x.x]^\diamond = (\lambda x.x)z((\lambda x.x)z)$$

The following are the main design principles that we have followed in order to arrive at a satisfactory definition of $\overset{\mathsf{S}}{\rightsquigarrow}$:

1. **Strong reduction.** The strong call-by-need strategy should implement *strong reduction*, *i.e.* if a term t is in $\overset{\mathsf{S}}{\rightsquigarrow}$-normal form then, when read back into the λ-calculus by unfolding all the explicit substitutions, the resulting λ-term $t^\diamond$ should be a β-normal form.

   Note that this criterion is lax enough that it allows us, for example, to take the terms $(\lambda x.x)[y\backslash\Omega]$ or $\lambda x.y[y\backslash x]$ as two valid encodings of the β-normal form $\lambda x.x$.

2. **Determinism.** The strong call-by-need strategy should be *deterministic*, that is, if $t \overset{\mathsf{S}}{\rightsquigarrow} s_1$ and $t \overset{\mathsf{S}}{\rightsquigarrow} s_2$ then $s_1 = s_2$.

3. **Conservativity.** The strong call-by-need strategy should be *conservative* with respect to weak call-by-need. That is, if $t \overset{\mathsf{W}}{\rightsquigarrow} s$ then $t \overset{\mathsf{S}}{\rightsquigarrow} s$.

4. **Correctness.** The strong call-by-need strategy should be *correct* with respect to β-equivalence. This means that if $t \overset{\mathsf{S}}{\rightsquigarrow} s$ then $t^\diamond =_\beta s^\diamond$.

5. **Completeness.** The strong call-by-need strategy should be *complete* with respect to β-equivalence. This means that if $t =_\beta s$ in the λ-calculus and s is a β-normal form, then $t(\overset{\mathsf{S}}{\rightsquigarrow})^* u$ and $u^\diamond = s$, *i.e.* u is an encoding of s modulo unfolding all the explicit substitutions.

After we have given a definition of the strong call-by-need strategy, all these principles will be stated and proved as theorems. In the following subsections we mention two non-trivial issues that one must confront in order to define a strong call-by-need strategy, the issue of *frozen variables*, and the issue of *context-dependency*.

Frozen Variables

Strong reduction performs evaluation below abstractions, so evaluation has to deal with open terms, *i.e.* the term may contain free variables. These variables are typically bound somewhere above in the evaluation context. In our presentation, a variable x may be bound by an abstraction $\lambda x.\Box$ or by an explicit substitution $\Box[x\backslash t]$. The behavior of the evaluator depends crucially on the nature of these variables.

For example, the evaluator may have to evaluate an application whose head is a variable, such as $x\,t$. If the variable x is bound to an *answer*, say by an explicit substitution $[x\backslash\lambda y.z]$, then evaluation should proceed by substituting x by the answer. In our example, evaluation proceeds as follows:

$$(x\,t)[x\backslash\lambda y.z] \to ((\lambda y.z)\,t)[x\backslash\lambda x.z] \to z[y\backslash t][x\backslash\lambda x.z]$$

Note that the term t is not evaluated in this case.

On the other hand, the variable x may be bound by an abstraction that cannot possibly become applied to an argument. Then, given that evaluation must implement strong reduction to normal form, the evaluator should go on and evaluate t:

$$\lambda x.x\,t \to \lambda x.x\,t' \to \lambda x.x\,t'' \to \dots$$

If a variable x is bound by an abstraction which cannot possibly become applied to an argument, we say that x is *frozen*. If a variable x is frozen, a term of the form $x\,t_1 \dots t_n$ is called a *structure*. Strictly speaking a structure may also contain explicit substitutions—such as in $y[y\backslash x]\,t$—. The precise definition of structure is postponed until later in the chapter. Variables bound to structures are also considered to be, transitively, frozen.

In the following examples the underlined variables are frozen:

$\lambda x.\lambda y.\underline{y}\,t$	— The abstractions cannot become applied.
$\lambda x.\underline{x}(\lambda y.\underline{y})$	— The abstractions cannot become applied.
$\lambda x.(\underline{z}t)[z\backslash\underline{y}s][y\backslash\underline{x}u]$	— The variables y and z are bound to structures.

In contrast, in the following examples the underlined variables are **not** frozen:

$(\lambda x.\underline{x}t)s$	— The abstraction can become applied.
$(\lambda y.y\,(\lambda z.z))(\lambda x.\underline{x}s)$	— The abstraction can, in principle, become applied.
$\underline{x}[x\backslash\lambda y.y]$	— The variable x is not bound to a structure.

In order to properly deal with all these situations, in our strong call-by-need strategy the notion of evaluation context is *parameterized* with respect to a set ϑ of frozen variables. For example, the context $x\,\Box$ will be considered a ϑ-evaluation context if and only if x is frozen, *i.e.* $x \in \vartheta$.

Context-Dependency

As mentioned in the previous subsection, strong reduction must perform evaluation below an abstraction, but only so if it can be certain that the abstraction cannot possibly become applied to an argument, along any possible reduction. More technically, one could say that the body of an abstraction should be evaluated only if it has already reached a position in which it will form part of the Böhm tree of the term.

For example, in the terms $\lambda x.t$ and $\lambda y.y(\lambda x.t)$ we know that the abstraction $\lambda x.t$ is not going to become applied, so to calculate the normal form of the term we should go on by "opening" the abstraction and evaluating the body t. On the other hand, in a term like $(\lambda x.t)s$ the *weak* call-by-need strategy would perform the db-step $(\lambda x.t)s \overset{\mathsf{W}}{\rightsquigarrow} t[x\backslash s]$ so to abide by the **Conservativity** principle, the strong call-by-need strategy should do the same thing. In particular, it should not evaluate the body t yet. Similarly, in the term $(xs)[x\backslash\lambda x.t]$ the weak

call-by-need strategy performs the substitution step $(xs)[x\backslash\lambda x.t] \overset{\mathsf{W}}{\rightsquigarrow} ((\lambda x.t)s)[x\backslash\lambda x.t]$, so again the strong call-by-need strategy must not evaluate t yet.

To properly deal with the context-dependent nature of strong call-by-need evaluation, we distinguish a particular subset of the evaluation contexts, the set of *inert evaluation contexts*. Intuitively, an evaluation context is *inert* if it can be plugged inside another evaluation context in such a way that the composition is still an evaluation context.

For example, the context $\lambda x.\Box$ is an evaluation context but it is not inert, because plugging it into the evaluation context $\Box\, t$ results in the context $(\lambda x.\Box)t$, which is not an evaluation context. Note that composing the contexts has enabled an interaction, namely it has created a `db`-redex, and evaluation should prioritize contracting the newly created `db`-redex:

$$((\lambda x.\Box)t)\langle s\rangle = (\lambda x.s)t \overset{\mathsf{W}}{\rightsquigarrow} s[x\backslash t]$$

Similarly, the context $(\lambda x.y)[y\backslash\Box]$ is an evaluation context, but it is not inert, because plugging it into the evaluation context $x[x\backslash\Box]$ results in the $x[x\backslash(\lambda x.y)[y\backslash\Box]]$ which is again not an evaluation context. As before, composing the contexts has enabled an interaction, creating an `lsv`-redex, and evaluation should contract it before going on:

$$(x[x\backslash(\lambda x.y)[y\backslash\Box]])\langle s\rangle = x[x\backslash(\lambda x.y)[y\backslash s]] \overset{\mathsf{W}}{\rightsquigarrow} (\lambda x.y)[x\backslash(\lambda x.y)][y\backslash s]$$

To define the strong call-by-need strategy, we shall restrict the composition of evaluation contexts so that only *inert* evaluation contexts can be plugged on the left of an application ($\Box\, t$) and inside explicit substitutions ($t[x\backslash\Box]$), so that nor `db` redexes neither `lsv` redexes are created due to the undesired enabling of an interaction.

4.1.3 Our Work

This chapter is the result of collaboration with Thibaut Balabonski, Eduardo Bonelli, and Delia Kesner, and it is structured as follows. We highlight in boldface what we consider to be the main contributions:

- In Section 4.2 we define the strong call-by-need strategy. Specifically:
  - In Section 4.2.1, we define a theory of strong reduction, the Theory of Sharing (Def. 4.4).
  - In Section 4.2.2, we motivate the definition of strong call-by-need, and we **define a strategy for strong call-by-need-reduction** (Def. 4.4), including various related notions such as normal forms and evaluation contexts.
  - In Section 4.2.3 we prove four basic principles that our strong call-by-need strategy enjoys, namely that it **reaches normal forms** (Prop. 4.16), it is **deterministic** (Prop. 4.18), it is a **conservative extension** of Ariola et al.'s notion of weak call-by-need (Thm. 4.23), and it is **correct** with respect to β-reduction (Prop. 4.25).
- In Section 4.3 we prove that the strong call-by-need strategy is *complete* with respect to β-reduction (Thm. 4.55). This means that if a λ-term has a β-normal form, then the

strong call-by-need strategy always finds it—modulo unfolding of explicit substitutions. The proof of completeness combines a *logical* argument and a *syntactical* argument. The logical argument relies on an auxiliary type system based on non-idempotent intersection types, and it shows that the Theory of Sharing is complete with respect to β-reduction. The syntactical argument shows that the strong call-by-need strategy is complete with respect to the Theory of Sharing. Specifically:

- In Section 4.3.1, we propose a **non-idempotent intersection type system** called $\mathcal{HW}$, for the Theory of Sharing (Def. 4.27). This is a simple adaptation of existing systems, following the line of work proposed by Kesner [91]. We also show that **typability implies normalization** (Thm. 4.43), *i.e.* that terms typable in $\mathcal{HW}$ are weakly normalizing in the Theory of Sharing.
- In Section 4.3.2, we use system $\mathcal{HW}$ to argue that **the Theory of Sharing is complete** (Prop. 4.45) with respect to β-reduction, *i.e.* that β-normalizing terms are also normalizing in the Theory of Sharing.
- In Section 4.3.3, we recall an abstract factorization result due to Accattoli [3]. Using this abstract result, we then argue that **the strong call-by-need strategy is complete** (Prop. 4.54) with respect to the Theory of Sharing. To do so we show that any reduction sequence in the Theory of Sharing may be factorized as a prefix whose steps are in the strong call-by-need strategy, followed by a suffix whose steps are garbage, *i.e.* steps inside unreachable explicit substitutions. The core of the proof is an exhaustive (and delicate) case analysis of permutation diagrams.

In the following chapter (Chapter 5), we extend the results of this section to incorporate pattern matching and recursion (with fixed points). In Section 8.1 in the Conclusion (Chapter 8), we propose an abstract machine for strong call-by-need evaluation. The proof that this machine implements the strong call-by-need strategy is left as future work.

4.2 Strong Call-by-Need

In this section we define the strong call-by-need strategy. Actually we begin by defining, in Section 4.2.1, a calculus which we call the *Theory of Sharing*. By a "calculus" what we mean is, formally speaking, a rewriting system. The objects of the Theory of Sharing are the usual set of terms of the Linear Substitution Calculus (variables, abstractions, applications, and explicit substitutions), as in Def. 4.2. The steps of the Theory of Sharing are given by a non-deterministic rewriting relation $\to_{\mathtt{sh}}$ whose reflexive, symmetric, and transitive closure gives us an equational theory (the equivalence relation $=_{\mathtt{sh}}$).

In Section 4.2.2 we define the strong call-by-need strategy itself. As already mentioned, the strategy is parameterized by a set of variables ϑ, which are supposed to be frozen. This means that, for each set ϑ, we define a deterministic rewriting relation $\overset{\vartheta}{\rightsquigarrow}$ which is a subset of $\to_{\mathtt{sh}}$. The strong call-by-need strategy corresponds to the case in which the set ϑ is empty, *i.e.* $\overset{\mathsf{S}}{\rightsquigarrow} \overset{\text{def}}{=} \overset{\varnothing}{\rightsquigarrow}$. In Section 4.2.3 we study some of its basic properties, namely the four principles of **Strong reduction**, **Determinism**, **Conservativity**, and **Correctness**.

4.2.1 The Theory of Sharing

The strong call-by-need strategy $\overset{\mathsf{S}}{\rightsquigarrow}$ can be seen as part of a bigger picture, the *Theory of Sharing*, given by the rewriting relation $\to_{\mathtt{sh}}$ that we define in this subsection.

A remark on nomenclature: in previous versions of this work, we spoke of "the Strong Call-by-Need Calculus" rather than of "the Theory of Sharing". We believe that the latter name is more appropriate, because the relation $\to_{\mathtt{sh}}$ does not enforce "by-need" evaluation; in fact, it allows to evaluate expressions that are not needed to obtain a result. On the other hand, the relation $\to_{\mathtt{sh}}$ does enforce sharing; in fact, an expression may not be copied unless it is already a value. For example, let $\Delta := (\lambda x.x)y$ and let $\Delta' := x[x\backslash y]$ be its contractum. Then a step like $x[y\backslash\Delta] \to_{\mathtt{sh}} x[y\backslash\Delta']$ is allowed in the Theory of Sharing, even though it is not needed, while a step like $x[x\backslash\Delta] \to_{\mathtt{sh}} \Delta[y\backslash\Delta]$ is not allowed in the Theory of Sharing, because it copies Δ, which is not a value.

Definition 4.4. The *Theory of Sharing* $\lambda_{\mathtt{sh}}$ is given by the set of terms $\mathcal{T}_{\mathtt{sh}}$ as in Def. 4.2, and the reduction relation $\to_{\mathtt{sh}} \overset{\text{def}}{=} \to_{\mathtt{db}} \cup \to_{\mathtt{lsv}} \cup \to_{\mathtt{gc}}$, where for each $\mathcal{R} \in \{\mathtt{db}, \mathtt{lsv}, \mathtt{gc}\}$, $\to_{\mathcal{R}}$ is the closure by full contexts of the corresponding rewrite rules below, *i.e.* $\to_{\mathcal{R}} \overset{\text{def}}{=} \mathtt{C}\langle\mapsto_{\mathcal{R}}\rangle$.

$$\begin{array}{rcll} (\lambda x.t)\mathtt{L}\, s & \mapsto_{\mathtt{db}} & t[x\backslash s]\mathtt{L} & \\ \mathtt{C}\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{v}\mathtt{L}] & \mapsto_{\mathtt{lsv}} & \mathtt{C}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\mathtt{L} & \\ t[x\backslash s] & \mapsto_{\mathtt{gc}} & t & \text{if } x \notin \mathtt{fv}(t) \end{array}$$

Note that the rules $\to_{\mathtt{db}}$ and $\to_{\mathtt{gc}}$ are exactly the $\to_{\mathtt{db}}$ and $\to_{\mathtt{gc}}$ rules of the LSC (*cf.* Def. 2.75). On the other hand, the $\to_{\mathtt{lsv}}$ rule of the Theory of Sharing and the $\to_{\mathtt{ls}}$ rule of the LSC are not instances of each other, since for example:

$$\begin{array}{rcll} x[x\backslash(\lambda y.z)[z\backslash t]] & \to_{\mathcal{R}} & (\lambda y.z)[x\backslash\lambda y.z][z\backslash t] & \text{holds for } \mathcal{R} = \mathtt{lsv} \text{ but not for } \mathcal{R} = \mathtt{ls} \\ x[x\backslash y] & \to_{\mathcal{R}} & y[x\backslash y] & \text{holds for } \mathcal{R} = \mathtt{ls} \text{ but not for } \mathcal{R} = \mathtt{lsv} \end{array}$$

Example 4.5. *The following is a reduction in the Theory of Sharing:*

$$\begin{array}{rcl} (\lambda x.zxx)((\lambda y.y)(\lambda w.w)) & \to_{\mathtt{sh}} & (\lambda x.zxx)(y[y\backslash\lambda w.w]) \\ & \to_{\mathtt{sh}} & (zxx)[x\backslash y[y\backslash\lambda w.w]] \\ & \to_{\mathtt{sh}} & (zxx)[x\backslash(\lambda w.w)[y\backslash\lambda w.w]] \\ & \to_{\mathtt{sh}} & (zx(\lambda w.w))[x\backslash\lambda w.w][y\backslash\lambda w.w] \\ & \to_{\mathtt{sh}} & (zx(\lambda w.w))[x\backslash\lambda w.w] \\ & \to_{\mathtt{sh}} & (z(\lambda w.w)(\lambda w.w))[x\backslash\lambda w.w] \\ & \to_{\mathtt{sh}} & z(\lambda w.w)(\lambda w.w) \end{array}$$

The following lemma characterizes the normal forms of the Theory of Sharing. We write $\mathtt{NF}(\to_{\mathtt{sh}})$ for the set of $\to_{\mathtt{sh}}$-normal forms, and $\mathtt{SNF}(\to_{\mathtt{sh}})$ for the set of $\to_{\mathtt{sh}}$-normal forms that are not answers, *i.e.* $t \in \mathtt{SNF}(\to_{\mathtt{sh}})$ if $t \in \mathtt{NF}(\to_{\mathtt{sh}})$ and t is not of the form $\mathtt{vL}$.

Definition 4.6 (Normal forms of the Theory of Sharing)**.** The set of $\mathtt{sh}$-*structures* ($\overline{\mathtt{S}}$) and the set of $\mathtt{sh}$-*normal forms* ($\overline{\mathtt{N}}$) are defined mutually inductively as follows:

$$\frac{}{x \in \overline{\mathtt{S}}} \qquad \frac{t \in \overline{\mathtt{S}} \quad u \in \overline{\mathtt{N}}}{tu \in \overline{\mathtt{S}}} \qquad \frac{t \in \overline{\mathtt{S}}}{t \in \overline{\mathtt{N}}} \qquad \frac{t \in \overline{\mathtt{N}}}{\lambda x.t \in \overline{\mathtt{N}}} \qquad \frac{t \in \mathbb{X} \quad u \in \overline{\mathtt{S}} \quad x \in \mathtt{fv}(t)}{t[x\backslash u] \in \mathbb{X}}$$

In the last rule, the symbol $\mathbb{X}$ represents either $\overline{\mathtt{S}}$ or $\overline{\mathtt{N}}$.

Lemma 4.7 (Characterization of strong normal forms)**.** *The following hold:*

- $\mathtt{NF}(\to_{\mathtt{sh}}) = \overline{\mathtt{N}}$
- $\mathtt{SNF}(\to_{\mathtt{sh}}) = \overline{\mathtt{S}}$

Proof. Given an arbitrary term $t \in \mathcal{T}$, one can check that $t \in \mathtt{NF}(\to_{\mathtt{sh}}) \iff t \in \overline{\mathtt{N}}$ and that $t \in \mathtt{SNF}(\to_{\mathtt{sh}}) \iff t \in \overline{\mathtt{S}}$. The left-to-right implication is straightforward by induction on t. The right-to-left implication is straightforward by simultaneous induction on the derivation that $t \in \overline{\mathtt{N}}$ and $t \in \overline{\mathtt{S}}$. □

4.2.2 The Strong Call-by-Need Strategy

In this subsection we define a deterministic rewriting relation $\overset{\mathsf{S}}{\rightsquigarrow}$ representing the *strong call-by-need strategy*. Unfortunately, by the nature of the problem that we are confronting, this rewriting relation does *not* enjoy straightforward closure properties under different kinds of contexts. This is due to the fact that some variables may be frozen, or not frozen, by the enclosing context. For example, if we let $\Delta := (\lambda y.y)z$ and $\Delta' := y[y\backslash z]$, then we can note that:

- $\Delta \overset{\mathsf{S}}{\rightsquigarrow} \Delta'$ should hold,
- $\lambda x.x\Delta \overset{\mathsf{S}}{\rightsquigarrow} \lambda x.x\Delta'$ should hold, because x is frozen under the context $\lambda x.\Box$, so $\lambda x.x\Box$ is an evaluation context,
- $(x\Delta)[x\backslash I] \overset{\mathsf{S}}{\rightsquigarrow} (x\Delta')[x\backslash I]$ should not hold, because x is not frozen under the context $\Box[x\backslash I]$, so $(x\Box)[x\backslash I]$ is not an evaluation context.

This means that a naïve contextual closure rule like "$t \overset{\mathsf{S}}{\rightsquigarrow} s$ holds if and only if $\lambda x.t \overset{\mathsf{S}}{\rightsquigarrow} \lambda x.s$ holds" is not valid. In order to be able to reason inductively, we need to consider an appropriate generalization of the strategy. In fact, we define a family of deterministic rewriting relations $\overset{\vartheta}{\rightsquigarrow}$ parameterized by a set ϑ of variables that are considered frozen. The strong call-by-need strategy is then given by $\overset{\mathsf{S}}{\rightsquigarrow} \overset{\text{def}}{=} \overset{\varnothing}{\rightsquigarrow}$. For example, with Δ and Δ' as above, we have that:

- $x\Delta \overset{\varphi}{\rightsquigarrow} x\Delta'$ holds if $x \in \varphi$,
- $x\Delta \overset{\varphi}{\rightsquigarrow} x\Delta'$ does not hold if $x \notin \varphi$.

These generalized relations will enjoy appropriate closure properties. For instance, $\lambda x.t \overset{\vartheta}{\rightsquigarrow} \lambda x.s$ holds if and only if $t \overset{\vartheta\cup\{x\}}{\rightsquigarrow} s$ holds, freezing the variable x.

Given that our aim is to define a strategy for *strong reduction*, *i.e.* reduction to normal form, the behavior of the relation $\overset{\vartheta}{\rightsquigarrow}$ will depend on whether certain subterms have already reached a normal form or not. For example, if we have an application ts, evaluation should focus on the argument s only if the function t is a strong normal form and not an answer.

Bearing this in mind, before defining the relation $\overset{\vartheta}{\rightsquigarrow}$ we will start by defining, syntactically, the set of normal forms that it should reach. This set will also depend on ϑ: for instance the term xy will be a strong normal form under ϑ if and only if $\{x, y\} \subseteq \vartheta$.

Moreover, the set of normal forms of the strategy $\overset{\vartheta}{\rightsquigarrow}$ does not coincide with the set of normal forms of the theory $\to_{\mathtt{sh}}$. By design, our strong call-by-need strategy does *not* perform garbage collection, *i.e.* seen as relations, the intersection $\overset{\vartheta}{\rightsquigarrow} \cap \to_{\mathtt{gc}}$ is empty, so the inclusion $\overset{\vartheta}{\rightsquigarrow} \subseteq \ \to_{\mathtt{db}} \cup \to_{\mathtt{lsv}} \ \subseteq \to_{\mathtt{sh}}$ holds. This means that for example $\lambda x.x[y\backslash t]$ will be a normal form of our strategy.

In the following subsections we define the relations $\overset{\vartheta}{\rightsquigarrow}$ and the corresponding notion of normal form under the set of frozen variables ϑ. But, before going on, we need a few auxiliary definitions, and in particular the notion of *non-garbage variable*.

Definition 4.8 (Garbage collection operation)**.** The operation of *garbage collection* $\downarrow_{\mathtt{gc}}(t)$ is defined as follows:

$$\begin{array}{rcll}
\downarrow_{\mathtt{gc}}(x) & \stackrel{\text{def}}{=} & x & \\
\downarrow_{\mathtt{gc}}(\lambda x.t) & \stackrel{\text{def}}{=} & \lambda x.\downarrow_{\mathtt{gc}}(t) & \\
\downarrow_{\mathtt{gc}}(ts) & \stackrel{\text{def}}{=} & \downarrow_{\mathtt{gc}}(t)\downarrow_{\mathtt{gc}}(s) & \\
\downarrow_{\mathtt{gc}}(t[x\backslash s]) & \stackrel{\text{def}}{=} & \begin{cases} \downarrow_{\mathtt{gc}}(t)[x\backslash\downarrow_{\mathtt{gc}}(s)] & \text{if } x \in \mathtt{fv}(\downarrow_{\mathtt{gc}}(t)) \\ \downarrow_{\mathtt{gc}}(t) & \text{otherwise} \end{cases} &
\end{array}$$

Note that this definition also erases explicit substitutions that are not garbage substitutions *stricto sensu*. For instance, consider the term $x[y\backslash z][z\backslash t]$. Both substitutions are collected by $\downarrow_{\mathtt{gc}}(.)$, even if an occurrence of the variable z temporarily appears in the subterm $x[y\backslash z]$.

Definition 4.9 (Non-garbage variables)**.** The set of *non-garbage variables* of a term t is defined as $\mathsf{ngv}(t) \stackrel{\text{def}}{=} \mathtt{fv}(\downarrow_{\mathtt{gc}}(t))$. Informally, $\mathsf{ngv}(t)$ is the set of free variables of t that are not erased by garbage collection. A free variable is a *garbage variable* if it is not non-garbage.

Lemma 4.10 (Inductive characterization of non-garbage variables)**.** *The set* $\mathsf{ngv}(t)$ *of non-garbage variables can be characterized by the following inductive equations:*

$$\begin{array}{rcl}
\mathsf{ngv}(x) & = & \{x\} \\
\mathsf{ngv}(\lambda x.t) & = & \mathsf{ngv}(t)\backslash\{x\} \\
\mathsf{ngv}(ts) & = & \mathsf{ngv}(t) \cup \mathsf{ngv}(s) \\
\mathsf{ngv}(t[x\backslash s]) & = & (\mathsf{ngv}(t)\backslash\{x\}) \cup \begin{cases} \mathsf{ngv}(s) & \textit{if } x \in \mathsf{ngv}(t) \\ \varnothing & \textit{otherwise} \end{cases}
\end{array}$$

Proof. Straightforward by induction on t. □

Normal Forms and Structures

The definition of the set of normal forms of the strategy $\overset{\vartheta}{\rightsquigarrow}$ depends on the key notion of *structure*. We summarize the three principles that motivate their definition.

1. **Frozen variables define the shape of the structures.** If x is frozen, reduction in a term like $x\,t$ must take place in t and hence the variable x persists in the reduct. This motivates our calling a term such as $x\,t$, with t in normal form, a *structure*. Iterating this idea, if x is frozen and $t_1, \ldots, t_n$ are in normal form then $x\,t_1\ \ldots\ t_n$ is a structure and reduction in a term like $x\,t_1\ \ldots\ t_n\,t_{n+1}$ must take place in t_{n+1}. The set of normal forms includes the set of structures as a proper subset.

   This principle leaves the following question open: is $y[y\backslash xt]$ a structure? The answer depends crucially on whether xt should be substituted for y.

2. **Structures should not be duplicated.** Weak call-by-need only duplicates values, abstractions being the only possible values. In weak reduction the set of values coincides with the set of weak-head normal forms, since all terms are closed. This raises the question of whether structures, which are weak-head normal forms in the setting of strong reduction, should be substituted too.

   The crucial observation is that, in contrast to abstractions, structures cannot contribute in any way to creating new redexes. Contrast for example the step:

   $$(xt)[x\backslash\lambda y.y] \to (\underline{(\lambda y.y)t})[x\backslash\lambda y.y]$$

   in which performing the substitution creates the underlined `db` redex, with the step:

   $$(xt)[x\backslash ys] \to (yst)[x\backslash ys]$$

   in which performing the substitution does not create any new interaction. This is a general phenomenon.

   Given that structures represent an incomplete computation whose evaluation is blocked by a head variable, and given that we do not want to duplicate incomplete computations, we do not substitute structures: structures are not considered values so they cannot be duplicated. This means that, if x is frozen and t is a normal form, the term $y[y\backslash x\,t]$ is a structure.

3. **Variables bound to structures are transitively frozen.** If a variable x is bound to a structure, then x is also considered frozen. Indeed, xy is a structure under the context $\Box[x\backslash zz]$ where x itself is bound to a structure and thus frozen, but it is not a structure under the context $\Box[x\backslash I]$ where x is bound to a value and thus not frozen.

Following these principles, we define the sets of ϑ-normal forms and ϑ-structures:

Definition 4.11. The set of *normal forms under the set of frozen variables* ϑ, also called ϑ-*normal forms* (N_ϑ), and the set of *structures under the set of frozen variables* ϑ, also called ϑ-*structures* (S_ϑ) are defined mutually inductively by the following rules:

$$\frac{x \in \vartheta}{x \in \mathsf{S}_\vartheta}\ \textsc{n-var} \qquad \frac{t \in \mathsf{S}_\vartheta \quad s \in \mathsf{N}_\vartheta}{ts \in \mathsf{S}_\vartheta}\ \textsc{n-app} \qquad \frac{t \in \mathsf{S}_\vartheta}{t \in \mathsf{N}_\vartheta}\ \textsc{nfStruct} \qquad \frac{t \in \mathsf{N}_{\vartheta \cup \{\mathsf{x}\}}}{\lambda x.t \in \mathsf{N}_\vartheta}\ \textsc{nfLam}$$

$$\frac{t \in \mathbb{X}^{\vartheta \cup \{x\}} \quad s \in \mathsf{S}_\vartheta \quad x \in \mathsf{ngv}(t)}{t[x\backslash s] \in \mathbb{X}^\vartheta}\ \textsc{nfSub} \qquad \frac{t \in \mathbb{X}^{\vartheta} \quad x \notin \mathsf{ngv}(t)}{t[x\backslash s] \in \mathbb{X}^\vartheta}\ \textsc{nfSubG}$$

In the last two rules, the symbol $\mathbb{X}$ represents either S or N.

Note that:

1. In the N-APP rule, the head of the application must be a structure, so for example xx is an $\{x\}$-structure while $(\lambda x.x)x$ is not. Intuitively, every ϑ-structure is *headed* by a frozen variable in the set ϑ. For example, both xyy and $z[z\backslash x\, y]$ are $\{x\}$-structures headed by x. Later we will prove this fact more rigorously.

2. In the NFLAM rule, the bound variable is frozen in the body, so for example $\lambda x.xx$ is an $\varnothing$-normal form because xx is an $\{x\}$-normal form.

3. The rules NFSUB and NFSUBG allow normal terms to contain explicit substitutions $[x\backslash t]$, which play two very different roles:

   3.1 The NFSUB rule allows substitutions to contain a structure, shared among the occurrences of x, as in the term $\lambda y.(xx)[x\backslash yt]$. In this rule, the bound variable is frozen in the body, and the argument of the substitution must be a structure, so for example $(xx)[x\backslash y]$ is an $\{y\}$-structure while $(xx)[x\backslash \lambda y.y]$ is not. Moreover, x should be non-garbage, otherwise we should apply the NFSUBG rule.

   3.2 The NFSUBG rule allows substitutions to be "garbage substitutions" as in $\lambda y.y[x\backslash t]$. In this rule, the bound variable should *not* be non-garbage, and then the argument of the substitution is allows to be an arbitrary term, so for example $x[y\backslash z][z\backslash \lambda w.w]$ is an $\varnothing$-normal form.

Evaluation Contexts

The strong call-by-need strategy $\overset{\vartheta}{\rightsquigarrow}$ is given by two reduction rules, which are, respectively, instances of the rules $\to_{\mathtt{db}}$ and $\to_{\mathtt{lsv}}$ of the Theory of Sharing. These rules are applied by focusing on specific locations in a term, as specified by *evaluation contexts*. The two principles of reduction are:

1. Perform function application as soon as db-redexes are found.

2. Evaluate and substitute the values to which variables are bound *on demand*.

Let us see how these principles apply when reducing an application $t\, s$. Evaluation should not focus on the argument s by default, since we do not know yet whether this argument is actually needed. Hence the first step is to reduce t until either it becomes an answer or, on the contrary, it becomes visible that it will never become an answer:

- If t becomes an answer, *i.e.* a λ-abstraction possibly affected by a substitution context $(\lambda x.t')\mathtt{L}$, then we should perform the db-step.

- If t becomes a term headed by a frozen variable, then it will never become an answer: it can only diverge or become a structure. For example, if Ω stands for the usual non-terminating term:

$x(II)$	becomes a structure, namely	$x\,I[y\backslash I]$
$x[x\backslash y(II)](II)$	becomes a structure, namely	$x[x\backslash y\,I[z\backslash I]]I[z\backslash I]$
$x\,\Omega$	diverges	

  In this case both t and s have to be independently evaluated to full normal form. According to our strategy, the evaluation focus should stay in t until it becomes a proper structure, and then continue evaluating s.

Note that the choice of reducing in t or s depends on whether t is a structure, which in turn depends on the variables that are frozen at this point. Thus, as was the case with normal terms and structures, the notion of evaluation context depends on a set ϑ of frozen variables. A context $t\,\mathtt{C}$ is an evaluation context under the set of frozen variables ϑ whenever $\mathtt{C}$ is an evaluation context and t is a structure under the same set of frozen variables ϑ.

Now consider a term of the form $t[x\backslash s]$. Following the second principle of reduction, the evaluation of the term s should be placed on hold until its value is required. Hence reduction should first proceed in t, until x becomes the *focused variable* in t, *i.e.* until an evaluation context reaches an occurrence of x in t. In this case, reduction should focus on s until an answer is obtained. An important subtlety here is how the notion of *focused variable* is to be understood in a strong setting. For example, x is the focused variable in $\lambda y.xy$ and, but also in $\lambda y.yx$. The focused variable is also x in the term $(zy)[z\backslash x\,I]$. In contrast, x is not the focused variable in $(yx)[y\backslash I]$, since y is not frozen under $\Box[y\backslash I]$ so, in this particular case, evaluation should proceed to perform the substitution of I for y. Observe that the focused variable, in case there is one, is always free.

Finally, in the case of a λ-abstraction, evaluation should proceed to evaluate its body (performing proper strong reduction) only if this abstraction can never become applied to an argument. As mentioned, before, to implement this condition we distinguish a particular subset of the evaluation contexts, containing all the evaluation contexts that are not led by a λ-abstraction, which we call *inert evaluation contexts*. There are two places at which only inert evaluation contexts can be plugged: on the left of an application to avoid reduction in the left part of a db-redex, and in a substitution to avoid reduction in a value that should be substituted. This way we ensure that, whenever an evaluation context focuses inside a λ-abstraction $\lambda x.t$, it is guaranteed that this λ-abstraction will never be applied, and thus the variable x can be remembered as frozen during the evaluation of t.

Following these principles, we define the sets of ϑ-evaluation contexts and inert ϑ-evaluation contexts:

Definition 4.12. The sets of *evaluation contexts under the set of frozen variables* ϑ, also called *ϑ-evaluation contexts* (E_ϑ) and of *inert evaluation contexts under the set of frozen variables* ϑ, also called *inert ϑ-evaluation contexts* ($\mathsf{E}^\circ_\vartheta$) are defined mutually inductively by the following rules:

$$\frac{}{\Box \in \mathsf{E}^{\circ}_{\vartheta}}\ \textsc{EBox} \qquad \frac{\mathtt{C} \in \mathsf{E}^{\circ}_{\vartheta}}{\mathtt{C}\,t \in \mathsf{E}^{\circ}_{\vartheta}}\ \textsc{EAppL} \qquad \frac{t \in \mathsf{S}_{\vartheta} \quad \mathtt{C} \in \mathsf{E}_{\vartheta}}{t\,\mathtt{C} \in \mathsf{E}^{\circ}_{\vartheta}}\ \textsc{EAppRStr}$$

$$\frac{\mathtt{C} \in \mathsf{E}^{\circ}_{\vartheta}}{\mathtt{C} \in \mathsf{E}_{\vartheta}}\ \textsc{e-incl} \qquad \frac{\mathtt{C} \in \mathsf{E}_{\vartheta \cup \{x\}}}{\lambda x.\mathtt{C} \in \mathsf{E}_{\vartheta}}\ \textsc{ELam} \qquad \frac{\mathtt{C} \in \mathbb{X}^{\vartheta} \quad t \notin \mathsf{S}_{\vartheta} \quad x \notin \vartheta}{\mathtt{C}[x\backslash t] \in \mathbb{X}^{\vartheta}}\ \textsc{ESubLNonStr}$$

$$\frac{\mathtt{C} \in \mathbb{X}^{\vartheta \cup \{x\}} \quad t \in \mathsf{S}_{\vartheta}}{\mathtt{C}[x\backslash t] \in \mathbb{X}^{\vartheta}}\ \textsc{ESubLStr} \qquad \frac{\mathtt{C}_1 \in \mathbb{X}^{\vartheta} \quad \mathtt{C}_2 \in \mathsf{E}^{\circ}_{\vartheta}}{\mathtt{C}_1\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{C}_2] \in \mathbb{X}^{\vartheta}}\ \textsc{ESubsR}$$

In the last three rules, the symbol $\mathbb{X}$ represents either E or E°.

Note that:

1. According to the EAppL rule, evaluation may proceed on the head of the application as long as the focus of evaluation is below an *inert* context, so for example $\Box((\lambda x.x)y)$ is an $\varnothing$-evaluation context while $(\lambda x.\Box)((\lambda x.x)y)$ is not.

2. According to the EAppRStr rule, evaluation may proceed on the argument of the application as long as the head is a structure, so for example $xx\Box$ is an $\{x\}$-evaluation context while $(\lambda x.x)\Box$ is not.

3. According to the ELam rule, evaluation may proceed on the body of the abstraction, the bound variable is frozen in the body, so for example $\lambda x.x\Box$ is an $\varnothing$-evaluation context, because $x\Box$ is an $\{x\}$-evaluation context. Note that in this case the resulting context is *not* an inert context.

4. The rules ESubLNonStr and ESubLStr allow evaluation to proceed on the body of a substitution, in two different ways:

   4.1 The ESubLNonStr allows evaluation to proceed on the body of a substitution whose argument is not a structure. This may be because the argument has not been fully evaluated yet, *e.g.* $x[x\backslash(\lambda y.y)z]$, or because it has been fully evaluated but it is an answer, *e.g.* $x[x\backslash(\lambda y.w)[w\backslash z]]$. In these cases, the bound variable is not frozen in the body, so for example $\Box[x\backslash\lambda y.y]$ is an $\varnothing$-evaluation context but $(x\Box)[x\backslash\lambda y.y]$ is not an $\varnothing$-evaluation context, because in turn $x\Box$ is not an $\varnothing$-evaluation context.

   4.2 The ESubLStr rule allows evaluation to proceed on the body of a substitution whose argument is a structure, freezing the bound variable, so for example $(x(y\Box))[x\backslash yy]$ is an $\{y\}$-evaluation context because yy is an $\{y\}$-structure and $x(y\Box)$ is a $\{x, y\}$-evaluation context.

5. According to the ESubsR rule, evaluation may proceed on the argument of a substitution, as long as the bound variable is the current focus of evaluation in the body. For example, let $\Delta = (\lambda x.x)(\lambda x.x)$. Then in a term like $(xy)[y\backslash\Delta][x\backslash\Delta]$ the context $(xy)[y\backslash\Delta][x\backslash\Box]$ is an $\varnothing$-evaluation context, because x is the focus of evaluation in xy,

while $(xy)[y\backslash\Box][x\backslash\Delta]$ is not an $\varnothing$-evaluation context, because y is not the focus of evaluation in xy.

Moreover, in this case, evaluation should proceed on the argument of the substitution as long as the focus of evaluation is below an *inert* context, so for example $x[x\backslash y\Box]$ is an $\{y\}$-evaluation context while $x[x\backslash\lambda y.\Box]$ is not an $\{y\}$-evaluation context.

Reduction

We are finally able to define the strong call-by-need strategy as a binary relation $\overset{\vartheta}{\rightsquigarrow}$.

Definition 4.13 (Strong call-by-need reduction)**.** The strong call-by-need strategy $\overset{\vartheta}{\rightsquigarrow}$ is given by the union of the two reduction rules $\overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}}$ and $\overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}}$ below:

$$\begin{array}{rcll} \mathtt{C}\langle(\lambda x.t)\mathtt{L}\,s\rangle & \overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}} & \mathtt{C}\langle t[x\backslash s]\mathtt{L}\rangle & \text{if } \mathtt{C}\in\mathsf{E}_\vartheta \\ \mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}]\rangle & \overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}} & \mathtt{C}_1\langle\mathtt{C}_2\langle\mathtt{v}\rangle[x\backslash\mathtt{v}]\mathtt{L}\rangle & \text{if } \mathtt{C}_1\langle\mathtt{C}_2\langle\Box\rangle[x\backslash\mathtt{vL}]\rangle\in\mathsf{E}_\vartheta \end{array}$$

Note that the strong call-by-need strategy $\overset{\vartheta}{\rightsquigarrow}$ requires that the *anchor* of the step is below a ϑ-evaluation context. In the `db` rule, this means that the contracted *application* $(\lambda x.t)\mathtt{L}\,s$ must lie below a context $\mathtt{C}\in\mathsf{E}_\vartheta$. In the `lsv` rule, this means that the contracted variable x, affected by the explicit substitution, must lie below a context $\mathtt{C}\in\mathsf{E}_\vartheta$. Moreover, since x must be bound to an answer, the context $\mathtt{C}$ has to be of the form $\mathtt{C}=\mathtt{C}_1\langle\mathtt{C}_2\langle\Box\rangle[x\backslash\mathtt{vL}]\rangle$.

Example 4.14. *The following is a reduction in strong call-by-need. In each step we underline the focus of evaluation,* i.e. *the pattern of the* `db` *redex or the variable contracted by the* `ls` *redex:*

$$\begin{array}{rcl} \underline{(\lambda x.xx)(\lambda y.z(Iz)y)} & \overset{\{z\}}{\rightsquigarrow} & (\underline{x}x)[x\backslash\lambda y.z(Iz)y] \\ & \overset{\{z\}}{\rightsquigarrow} & (\underline{(\lambda y'.z(Iz)y')x})[x\backslash\lambda y.z(Iz)y] \\ & \overset{\{z\}}{\rightsquigarrow} & (z(\underline{Iz})y')[y'\backslash x][x\backslash\lambda y.z(Iz)y] \\ & \overset{\{z\}}{\rightsquigarrow} & (z(w[w\backslash z])y')[y'\backslash\underline{x}][x\backslash\lambda y.z(Iz)y] \\ & \overset{\{z\}}{\rightsquigarrow} & (z(w[w\backslash z])\underline{y'})[y'\backslash\lambda y.z(Iz)y][x\backslash\lambda y.z(Iz)y] \\ & \overset{\{z\}}{\rightsquigarrow} & (z(w[w\backslash z])(\lambda y''.z(\underline{Iz})y''))[y'\backslash\lambda y.z(Iz)y][x\backslash\lambda y.z(Iz)y] \\ & \overset{\{z\}}{\rightsquigarrow} & (z(w[w\backslash z])(\lambda y''.z(w[w\backslash z])y''))[y'\backslash\lambda y.z(Iz)y][x\backslash\lambda y.z(Iz)y] \end{array}$$

4.2.3 Basic Properties of Strong Call-by-Need

In Section 4.1.2 we listed five design principles that we followed to define the strong call-by-need strategy. In each of the subsections of this section, we state and prove the first four principles: **Strong reduction** (Prop. 4.16), **Determinism** (Prop. 4.18), **Conservativity** (Thm. 4.23), and **Correctness** (Prop. 4.25). The statement and proof of the fifth principle, **Completeness**, is much more complex and we defer it until the next section.

Strong Reduction

In this subsection, we show that the strong call-by-need strategy reaches normal forms, up to the unfolding of explicit substitutions. The following auxiliary lemma characterizes the set of normal forms of the strategy $\overset{\vartheta}{\rightsquigarrow}$.

Lemma 4.15 (Characterization of ϑ-normal forms — ♣ Lem. A.30)**.** *The following sets are equal:*

- *The set of ϑ-normal forms N_ϑ (*cf. *Def. 4.11).*
- *The set of normal forms of the strong call-by-need strategy $\overset{\vartheta}{\rightsquigarrow}$.*

Proof. See the appendix. □

Proposition 4.16 (Strong reduction)**.** *If t is a $\overset{\vartheta}{\rightsquigarrow}$-normal term, then its unfolding $t^\diamond$ is a λ-term in β-normal form.*

Proof. By Lem. 4.15, it suffices to show that if $t \in \mathsf{N}_\vartheta$ then $t^\diamond$ is a β-normal λ-term. We prove a stronger property, namely that if $t \in \mathsf{N}_\vartheta$ or $t \in \mathsf{S}_\vartheta$ then t is a β-normal λ-term and, moreover, if $t \in \mathsf{S}_\vartheta$ then t is a *neutral term*, *i.e.* of the form $x\, t_1 \dots t_n$. We proceed by mutual induction on the derivations that $t \in \mathsf{N}_\vartheta$ and $t \in \mathsf{S}_\vartheta$. The interesting cases are the rules nfSub and nfSubG. For the rule nfSub, note that the variable is bound to a neutral term, so performing the substitution does not create a β-step. For the rule nfSubG, note that the variable bound by the substitution is *not* a non-garbage variable, so it does not occur free in the unfolding of the body, and the property holds immediately by *i.h.*. □

Determinism

In this subsection we show that the strong call-by-need strategy is deterministic. The following auxiliary lemma states, roughly, that there can be only one redex below an evaluation context.

Lemma 4.17 (Unique decomposition — ♣ Lem. A.34)**.** *If $\mathsf{C}\langle r\rangle$ is a term, we say that r is an* anchor *if it is a* db*-redex or a variable bound to an answer. Let t be a term that can be written as both $\mathsf{C}_1\langle r_1\rangle$ and $\mathsf{C}_2\langle r_2\rangle$, where $\mathsf{C}_1, \mathsf{C}_2 \in \mathsf{E}_\vartheta$ are evaluation contexts and r_1, r_2 are anchors. Then $\mathsf{C}_1 = \mathsf{C}_2$ and $r_1 = r_2$.*

Proposition 4.18 (Determinism)**.** *If $t \overset{\vartheta}{\rightsquigarrow} s$ and $t \overset{\vartheta}{\rightsquigarrow} u$ then $s = u$.*

Proof. An immediate consequence of the unique decomposition lemma (Lem. 4.17). □

Conservativity

In this subsection we show that the strong call-by-need strategy is conservative over weak call-by-need. To do so, we relate the strong call-by-need strategy with the weak call-by-need strategy $\overset{\mathsf{W}}{\rightsquigarrow}$ (*cf.* Def. 4.2), as well as to the original notion of weak call-by-need reduction in [13, 12]. Moreover, we take the opportunity to put in evidence the general scheme followed

by any reduction sequence of [13] (Lem. 4.21), and we also state a clear relation between these two mentioned notions of weak call-by-need (Lem. 4.22)

Definition 4.19 (Ariola et al.'s notion of weak call-by-need)**.** The syntax of the system in [13, 12] is given by the following sets of terms (t), values ($\mathtt{v}$), answers (a), and evaluation contexts ($\mathtt{E}$)[3]:

$$\begin{array}{rcl} t & ::= & x \mid \lambda x.t \mid t\,t \mid t[x\backslash t] \\ \mathtt{v} & ::= & \lambda x.t \\ a & ::= & \mathtt{v} \mid a[x\backslash t] \\ \mathtt{E} & ::= & \Box \mid \mathtt{E}\,t \mid \mathtt{E}\,t \mid \mathtt{E}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{E}] \end{array}$$

There are four rewriting rules:

$$\begin{array}{rcll} (\lambda x.t)\,s & \overset{\mathrm{R}}{\mapsto}_{\mathrm{I}} & t[x\backslash s] & \\ \mathtt{E}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{v}] & \overset{\mathrm{R}}{\mapsto}_{\mathrm{V}} & \mathtt{E}\langle \mathtt{v} \rangle[x\backslash \mathtt{v}] & \\ a[x\backslash t]\,s & \overset{\mathrm{R}}{\mapsto}_{\mathrm{C}} & (a\,s)[x\backslash t] & \text{if } x \notin \mathtt{fv}(s) \\ \mathtt{E}\langle\!\langle x \rangle\!\rangle[x\backslash a[y\backslash t]] & \overset{\mathrm{R}}{\mapsto}_{\mathrm{A}} & \mathtt{E}\langle\!\langle x \rangle\!\rangle[x\backslash a][y\backslash t] & \text{if } y \notin \mathtt{fv}(\mathtt{E}\langle\!\langle x \rangle\!\rangle) \end{array}$$

Reduction is defined by $\mapsto_{\mathrm{need}} \overset{\mathrm{def}}{=} \mapsto_{\mathrm{I}} \cup \mapsto_{\mathrm{V}} \cup \mapsto_{\mathrm{C}} \cup \mapsto_{\mathrm{A}}$, where $\mapsto_{\mathtt{X}}$ is the closure by evaluation contexts of $\overset{\mathrm{R}}{\mapsto}_{\mathtt{X}}$, *i.e.* $\mapsto_{\mathtt{X}} \overset{\mathrm{def}}{=} \mathtt{E}\langle \overset{\mathrm{R}}{\mapsto}_{\mathtt{X}} \rangle$ for each $\mathtt{X} \in \{\mathrm{I}, \mathrm{V}, \mathrm{C}, \mathrm{A}\}$.

It turns out that $\mapsto_{\mathrm{need}}$ is deterministic:

Proposition 4.20 (Determinism of $\mapsto_{\mathrm{need}}$)**.** *If $t \mapsto_{\mathrm{need}} s$ then there exists a unique context $\mathtt{E}$ such that $t = \mathtt{E}\langle t' \rangle$, $s = \mathtt{E}\langle s' \rangle$ and $t' \mapsto s'$, where $\mapsto \overset{\mathrm{def}}{=} \overset{R}{\mapsto}_{\mathrm{I}} \cup \overset{R}{\mapsto}_{\mathrm{V}} \cup \overset{R}{\mapsto}_{\mathrm{C}} \cup \overset{R}{\mapsto}_{\mathrm{A}}$.*

Proof. See [13, Lemma 4.2]. □

Using this property, one can observe that any reduction sequence in $\mapsto_{\mathrm{need}}$ is organized into clusters of the form $\mapsto_{\mathrm{C}}^* \mapsto_{\mathrm{I}}$ or $\mapsto_{\mathrm{A}}^* \mapsto_{\mathrm{V}}$. More precisely:

Lemma 4.21 (Organization of $\mapsto_{\mathrm{need}}$ reduction sequences)**.**

$$\mapsto_{\mathrm{need}}^* \quad = \quad ((\mapsto_{\mathrm{C}}^* \mapsto_{\mathrm{I}}) \cup (\mapsto_{\mathrm{A}}^* \mapsto_{\mathrm{V}}))^* (\mapsto_{\mathrm{C}}^* \cup \mapsto_{\mathrm{A}}^*)$$

Proof. Straightforward by induction on the number of $\mapsto_{\mathrm{need}}$ steps. The key observation is that, after firing a $\mapsto_{\mathrm{C}}$ step, only a step in $\mapsto_{\mathrm{C}} \cup \mapsto_{\mathrm{I}}$ may be fired. Similarly, after firing a $\mapsto_{\mathrm{A}}$ step, only only a step in $\mapsto_{\mathrm{A}} \cup \mapsto_{\mathrm{V}}$ may be fired. □

On the other hand, the weak call-by-need strategy $\overset{\mathsf{W}}{\rightsquigarrow}$ given in Def. 4.2 has the same syntax as Ariola et al.'s system but a different set of rewriting rules. Indeed, recall that it is defined as the union of the two rewrite rules below, closed by evaluation contexts:

$$\begin{array}{rcl} (\lambda x.t)\mathtt{L}\,s & \overset{\mathsf{W}}{\rightsquigarrow}_{\mathtt{db}} & t[x\backslash s]\mathtt{L} \\ \mathtt{E}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{v}\mathtt{L}] & \overset{\mathsf{W}}{\rightsquigarrow}_{\mathtt{lsv}} & \mathtt{E}\langle\!\langle \mathtt{v} \rangle\!\rangle[x\backslash \mathtt{v}]\mathtt{L} \end{array}$$

[3]Remark that Ariola et al. use let syntax ($\mathrm{let}\ x = s\ \mathrm{in}\ t$) rather than explicit substitution syntax ($t[x\backslash s]$), but this is only a change of notation.

The set of terms defined by the grammar $\mathsf{N}^{\mathsf{w}}_{\vartheta} ::= \mathtt{vL} \mid \mathtt{E}\langle\!\langle x \rangle\!\rangle$ for $x \in \vartheta$ characterizes the set of normal forms with respect to the weak call-by-need strategy.

It is quite straightforward to deduce that $\overset{\mathsf{W}}{\rightsquigarrow}$ is included in $\mapsto_{\mathrm{need}}$, in particular, a `db` step (resp. `lsv`) step translates to a $\mapsto_{\mathrm{C}}^{*}\mapsto_{\mathrm{I}}$ cluster (resp. $\mapsto_{\mathrm{A}}^{*}\mapsto_{\mathrm{V}}$ cluster). These clusters in fact characterize $\overset{\mathsf{W}}{\rightsquigarrow}$.

Lemma 4.22 (Decomposition of $\overset{\mathsf{W}}{\rightsquigarrow}$)**.**

$$\overset{\mathsf{W}}{\rightsquigarrow} \quad = \quad (\mapsto_{\mathrm{C}}^{*}\mapsto_{\mathrm{I}}) \cup (\mapsto_{\mathrm{A}}^{*}\mapsto_{\mathrm{V}})$$

Proof.

(⊆) The inclusion $\overset{\mathsf{W}}{\rightsquigarrow} \subseteq (\mapsto_{\mathrm{C}}^{*}\mapsto_{\mathrm{I}}) \cup (\mapsto_{\mathrm{A}}^{*}\mapsto_{\mathrm{V}})$ is proved by cases on the kind of redex contracted.

1. `db` redex
$$\mathtt{E}\langle(\lambda x.t)\mathtt{L}\, s\rangle \mapsto_{\mathrm{C}}^{*} \mathtt{E}\langle((\lambda x.t)\, s)\mathtt{L}\rangle \mapsto_{\mathrm{I}} \mathtt{E}\langle t[x\backslash s]\mathtt{L}\rangle$$
2. `lsv` redex
$$\mathtt{E}\langle\mathtt{E}'\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}]\rangle \mapsto_{\mathrm{A}}^{*} \mathtt{E}\langle\mathtt{E}'\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{v}]\mathtt{L}\rangle \mapsto_{\mathrm{V}} \mathtt{E}\langle\mathtt{E}'\langle\!\langle \mathtt{v} \rangle\!\rangle[x\backslash \mathtt{v}]\mathtt{L}\rangle$$

(⊇) The inclusion $(\mapsto_{\mathrm{C}}^{*}\mapsto_{\mathrm{I}}) \cup (\mapsto_{\mathrm{A}}^{*}\mapsto_{\mathrm{V}}) \subseteq \overset{\mathsf{W}}{\rightsquigarrow}$ follows from the remarks stated below and determinism of $\mapsto_{\mathrm{need}}$ (Prop. 4.20).

$$\begin{array}{lll} t \mapsto_{\mathrm{I}} s & \text{implies} & t \mapsto_{\mathtt{db}} s \\ t \mapsto_{\mathrm{V}} s & \text{implies} & t \mapsto_{\mathtt{lsv}} s \\ t \mapsto_{\mathrm{C}} s & \text{implies} & \exists s'.\ (s \mapsto_{\mathrm{C}}^{*} \mapsto_{\mathrm{I}} s') \wedge (t \mapsto_{\mathtt{db}} s') \\ t \mapsto_{\mathrm{A}} s & \text{implies} & \exists s'.\ (s \mapsto_{\mathrm{A}}^{*} \mapsto_{\mathrm{V}} s') \wedge (t \mapsto_{\mathtt{lsv}} s') \end{array}$$

□

The **Conservativity** principle states that our strong call-by-need strategy is conservative with respect to $\overset{\mathsf{W}}{\rightsquigarrow}$, *i.e.* that $t \overset{\mathsf{W}}{\rightsquigarrow} s$ implies $t \overset{\mathsf{S}}{\rightsquigarrow} s$. More precisely, let $\overset{\vartheta\backslash\mathsf{W}}{\rightsquigarrow}$ stand for $\overset{\vartheta}{\rightsquigarrow} \backslash \overset{\mathsf{W}}{\rightsquigarrow}$. Then we have:

Theorem 4.23 (Conservativity — ♣ Thm. A.50)**.** *If* $t_0 \overset{\vartheta}{\rightsquigarrow} t_1 \overset{\vartheta}{\rightsquigarrow} \dots t_{n-1} \overset{\vartheta}{\rightsquigarrow} t_n$ *there exists an* $1 \leqslant i \leqslant n$ *such that the three following conditions hold:*

1. $t_0 \overset{\mathsf{W}}{\rightsquigarrow} t_1 \overset{\mathsf{W}}{\rightsquigarrow} \dots t_{n-1} \overset{\mathsf{W}}{\rightsquigarrow} t_i$
2. $t_i \overset{\vartheta\backslash\mathsf{W}}{\rightsquigarrow} t_{i+1} \overset{\vartheta\backslash\mathsf{W}}{\rightsquigarrow} \dots t_{n-1} \overset{\vartheta\backslash\mathsf{W}}{\rightsquigarrow} t_n$
3. *If* $i < n$*, then* $t_j \in \mathsf{N}^{\mathsf{w}}_{\vartheta}$ *for all* $i \leqslant j \leqslant n$.

Proof. See the appendix. □

As a corollary of Thm. 4.23 and Lem. 4.22, we deduce that our strategy $\overset{\vartheta}{\rightsquigarrow}$ has Ariola et al.'s notion of weak call-by-need reduction as a prefix.

Corollary 4.24. *If* $t\ (\overset{\vartheta}{\rightsquigarrow})^*\ s$ *then there is a term* u *such that*

$$t\ ((\mapsto_{\mathrm{C}}^*\mapsto_{\mathrm{I}}) \cup (\mapsto_{\mathrm{A}}^*\mapsto_{\mathrm{V}}))^*\ u\ (\overset{\vartheta\setminus \mathsf{W}}{\rightsquigarrow})^*\ s$$

Moreover, if $s \in \mathsf{N}_\vartheta$*, then* u *is a normal form for* $\mapsto_{\mathrm{need}}$ *up to a finite number of* $\mapsto_{\mathrm{C}} \cup \mapsto_{\mathrm{A}}$ *steps.*

Correctness

To conclude this section, we remark that the strong call-by-need strategy $\overset{\vartheta}{\rightsquigarrow}$ is correct with respect to β-reduction:

Proposition 4.25 (Correctness)**.** *If* $t \overset{\vartheta}{\rightsquigarrow} s$ *then* $t^\diamond =_\beta s^\diamond$.

Proof. Observe that $\overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}}$ and $\overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}}$ (*cf.* Def. 4.13) are instances of $\to_{\mathtt{db}}$ and $\to_{\mathtt{lsv}}$ respectively (*cf.* Def. 4.4), so if $t \overset{\vartheta}{\rightsquigarrow} s$ then we have that $t \to_{\mathtt{sh}} s$. Moreover, it is a straightforward exercise to show that the Theory of Sharing $\to_{\mathtt{sh}}$ is correct, *i.e.* that $t \to_{\mathtt{sh}} s$ implies $t^\diamond =_\beta s^\diamond$. □

4.3 Completeness of Strong Call-by-Need

This section is devoted to the proof of the **Completeness** principle for our strong call-by-need strategy. Recall that by completeness we mean completeness with respect to β-reduction, in the sense that whenever a term t admits a β-normal form s in the λ-calculus, then the strategy $\overset{\mathsf{S}}{\rightsquigarrow}$ computes a normal form u, and the normal forms are in a precise correspondence, more specifically $u^\diamond = s$.

A first completeness result for weak call-by-need is found in Ariola et al. [13]. Their proof makes use of various syntactical tools such as sharing, residual theory and standardization. A more abstract proof has been developed more recently by Kesner [86]. Kesner shows that every λ-term that can be reduced to a weak head normal-form is typable in an appropriate typing system with intersection types, and that every typable term is normalizing in the weak call-by-need calculus. Here we adopt similar ideas in order to develop a completeness proof for strong call-by-need.

Suppose that $t =_\beta s$ are interconvertible terms in the λ-calculus, and suppose that s is a β-normal form. Then by confluence of the λ-calculus we have a reduction $t \twoheadrightarrow_\beta s$. Completeness of the strong call-by-need strategy would mean that there exists a term u such that $t \overset{\vartheta}{\rightsquigarrow}{}^*\ u$ and $u^\diamond = s$. To prove this, we decompose the proof of completeness of the strategy in two parts:

1. **Completeness of the Theory of Sharing.** First, we prove that the Theory of Sharing $\to_{\mathtt{sh}}$ is complete with respect to β-reduction. This entails that there is a term r such that $t \twoheadrightarrow_{\mathtt{sh}} r$ and $r^\diamond = s$.

2. **Factorization of the Theory of Sharing.** Second, we prove that any reduction in the Theory of Sharing $\to_{\mathtt{sh}}$ may be factorized as a prefix of *external* steps (*i.e.* a sequence of steps in the strategy $\overset{\vartheta}{\rightsquigarrow}$) followed by a suffix of *internal* steps which preserve unfolding. This entails that there is a term u such that $t \overset{\vartheta}{\rightsquigarrow}{}^{*}\ u$ and such that $u^{\diamond} = r^{\diamond} = s$.

The decomposition is depicted graphically in Figure 4.1.

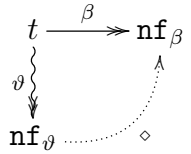
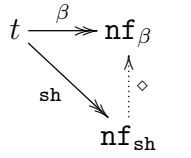
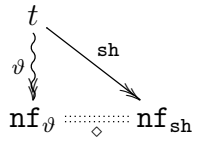
(a)		**Completeness of strong call-by-need** If $t \twoheadrightarrow_{\beta} s \in \mathtt{NF}(\to_{\beta})$ then there exists a term u such that $t \overset{\vartheta}{\rightsquigarrow}{}^{*}\ u \in \mathsf{N}_{\vartheta}$ and $u^{\diamond} = s$, where $\vartheta = \mathtt{fv}(t)$. (See Thm. 4.55).
(b)		**Completeness of the Theory of Sharing** If $t \twoheadrightarrow_{\beta} s \in \mathtt{NF}(\to_{\beta})$ then there exists a term u such that $t \twoheadrightarrow_{\mathtt{sh}} u \in \mathtt{NF}(\to_{\mathtt{sh}})$ and $u^{\diamond} = s$. (See Prop. 4.45).
(c)		**Factorization of the Theory of Sharing** If $t \twoheadrightarrow_{\mathtt{sh}} s \in \mathtt{NF}(\to_{\mathtt{sh}})$ then there exists a term u such that $t \overset{\vartheta}{\rightsquigarrow}{}^{*}\ u \in \mathsf{N}_{\vartheta}$ and $u^{\diamond} = s^{\diamond}$, where $\vartheta = \mathtt{fv}(t)$. (See Prop. 4.54).

Figure 4.1: Decomposition of the proof of **Completeness**: (a) is implied by (b) and (c)

The first step, *i.e.* the proof of completeness of the Theory of Sharing, relies on a type system called $\mathcal{HW}$, introduced by Kesner and Ventura in [91]. System $\mathcal{HW}$ is based on the technology of *non-idempotent intersection types*. It extends Gardner–De Carvalho's system [58, 34] to include terms with explicit substitutions $t[x\backslash s]$, besides pure λ-terms. Our proof of completeness follows closely Kesner's proof of completeness for weak call-by-need [86], extending it to the Theory of Sharing.

The fundamental property of intersection type systems is that they characterize normalization. In particular, non-idempotent intersection type systems may be formulated in such a way that they characterize *weak normalization*, *i.e.* a term has a normal form if and only if it is typable in a non-idempotent intersection type system. The key observation by Kesner is that the proof of completeness, relating weak normalization in two different calculi (in our case, reduction in the λ-calculus ($\to_{\beta}$), and in the Theory of Sharing ($\to_{\mathtt{sh}}$)), may be simplified by relating, on one hand, weak normalization in each of the calculi with, on the other hand, typability in system $\mathcal{HW}$. More precisely, this allows us to decompose completeness of the Theory of Sharing into two implications:

1. If a term t has a normal form in the λ-calculus, *i.e.* $t \in \mathtt{WN}(\to_{\beta})$, then t is typable in $\mathcal{HW}$. Moreover, the typing judgment $\Gamma \vdash t : \tau$ verifies a structural condition, namely it has no *positive occurrences* of the *empty type*.

2. If a term t is typable in $\mathcal{HW}$ and the judgment verifies the same structural condition as above, then t has a normal form in the Theory of Sharing, *i.e.* $t \in \mathtt{WN}(\to_{\mathtt{sh}})$.

By composing the implications we conclude that if t has a $\to_\beta$-normal form in the λ-calculus then t has a $\to_{\mathtt{sh}}$-normal form in the Theory of Sharing, recovering (most of) the completeness result.

The following subsections are organized as follows:

- In Section 4.3.1, we recall the non-idempotent intersection type system $\mathcal{HW}$ from [91]. Furthermore, we prove a result relating weak normalization in the Theory of Sharing with typability in $\mathcal{HW}$.
- In Section 4.3.2, we prove completeness of the Theory of Sharing, as displayed in Figure 4.1(b). As described above, the proof uses typability in $\mathcal{HW}$ as a stepping stone.
- In Section 4.3.2, we prove a factorization result for the Theory of Sharing, as displayed in Figure 4.1(c). If we write $\xrightarrow{\neg\vartheta}_{\mathtt{sh}}$ for $\to_{\mathtt{sh}} \setminus \overset{\vartheta}{\rightsquigarrow}$, the proof is based on repeatedly swapping pairs steps of steps $t \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \overset{\vartheta}{\rightsquigarrow} s$ such that they become of the form $t \overset{\vartheta}{\rightsquigarrow} \twoheadrightarrow_{\mathtt{sh}} s$.

4.3.1 The Non-Idempotent Intersection Type System $\mathcal{HW}$

In contrast to simple types, intersection types are powerful enough to characterize termination properties: a λ-term has a head normal form *if and only if* it is typable in a suitable intersection type system. That means, in particular, that a head normalizing term like $\lambda x.xx$, which is not typable in the simply typed λ-calculus, is typable in certain type systems with intersection types.

This is done by introducing a new type constructor ($\wedge$), representing type *intersection*, together with a corresponding set of typing rules. For instance, in these systems the term $\lambda x.xx$ can be given the type $((\tau \to \tau) \wedge \tau) \to \tau$ in such a way that the first (resp. second) occurrence of the variable x is typed with $\tau \to \tau$ (resp. τ). Typically, intersection is declared to be commutative (*i.e.* $\tau \wedge \sigma \equiv \sigma \wedge \tau$), associative (*i.e.* $(\tau \wedge \sigma) \wedge \rho \equiv \tau \wedge (\sigma \wedge \rho)$) and *idempotent* (*i.e.* $\tau \wedge \tau \equiv \tau$).

In *non-idempotent* intersection type systems [58], intersection is not declared to be idempotent, *i.e.* $\tau \wedge \tau \not\equiv \tau$. These non-idempotent types allow giving types to terms according to a *resource aware* semantics. The informal idea behind the resource aware semantics is that a term of type $\tau_1 \wedge \ldots \wedge \tau_n$ can be understood as a resource that must be used exactly n times, once with type τ_i for each $1 \leqslant i \leqslant n$. Dually, a term of type $(\tau_1 \wedge \ldots \wedge \tau_n) \to \rho$ is a function that uses its argument exactly n times, once with type τ_i for each $1 \leqslant i \leqslant n$. Non-idempotent intersection type systems also provide a simple formal framework to reason about termination properties: in particular, in these systems correctness results are usually proved by simple inductive arguments rather than with more intricate arguments typical of their idempotent counterparts.

From a formal point of view, the result of applying a commutative, associative and non-idempotent binary operation to a collection of elements can be represented by a *multiset* of elements, which provides a very convenient notation to manipulate them. We denote finite

multisets with brackets, so that $[\ ]$ denotes the empty multiset and $[\sigma, \sigma, \tau]$ denotes a multiset having two occurrences of the element σ and one occurrence of τ, corresponding to the intersection type $\sigma \wedge \sigma \wedge \tau$. In this system, we write $+$ for the (additive) union of multisets and $\sqsubseteq$ for multiset inclusion. Below we recall the intersection type system $\mathcal{HW}$ from [91].

Definition 4.26 (Syntax of $\mathcal{HW}$)**.** Given a countable infinite set $\mathcal{B}$ of *base types* $\alpha, \beta, \gamma, \ldots$ the set of *types* and *multisets of types* are defined mutually inductively by the following grammar:

$$\begin{array}{lllll} \text{Types} & \tau, \sigma, \rho & ::= & \alpha \mid \mathcal{M} \to \tau & \\ \text{Multisets of types} & \mathcal{M} & ::= & [\tau_i]_{i \in I} & \text{where } I \text{ is a finite set} \end{array}$$

The *empty multiset* $[\ ]$ plays the rôle of the universal ω type in [40]. The types are *strict* [39, 139], that is, the right-hand sides of function types are never multisets.

A *type assignment* or *typing context*, ranged over by Γ, Δ, etc., is a function mapping variables to multiset types. The domain of Γ is defined by $\mathsf{dom}(\Gamma) := \{x \mid \Gamma(x) \neq [\ ]\}$. We assume that typing contexts have *finite domain*.

The *union of typing contexts*, written $\Gamma + \Delta$, is the typing context defined by $(\Gamma + \Delta)(x) := \Gamma(x) + \Delta(x)$, where the symbol $+$ denotes the additive union of multisets. Note that $\mathsf{dom}(\Gamma + \Delta) = \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Delta)$. We write $\Gamma \oplus \Delta$ to stand for $\Gamma + \Delta$ whenever $\mathsf{dom}(\Gamma)$ and $\mathsf{dom}(\Delta)$ are disjoint. We write $\Gamma +_{i \in I} \Delta_i$ to abbreviate $\Gamma + \sum_{i \in I} \Delta_i$. The *inclusion between typing contexts*, written $\Gamma \sqsubseteq \Delta$, is defined to hold if for every variable x we have that $\Gamma(x) \sqsubseteq \Delta(x)$.

For example $(x : [\sigma], y : [\tau]) + (x : [\sigma], z : [\sigma]) = x : [\sigma, \sigma], y : [\tau], z : [\sigma]$, and $x : [\sigma] \sqsubseteq x : [\sigma, \sigma], y : \rho$.

Definition 4.27 (The $\mathcal{HW}$ type system, [91])**.** *Typing judgments* are of the form $\Gamma \vdash t : \tau$, where Γ is a typing context, t is a term and τ is a type. The *$\mathcal{HW}$-type system* is given by the following rules:

$$\frac{}{x{:}[\tau] \vdash x : \tau}\ \textsc{t-var} \qquad \frac{\Gamma \vdash t : [\sigma_i]_{i \in I} \to \tau \quad (\Delta_i \vdash s : \sigma_i)_{i \in I}}{\Gamma +_{i \in I} \Delta_i \vdash t\, s : \tau}\ \textsc{t-app}$$

$$\frac{\Gamma \oplus (x : \mathcal{M}) \vdash t : \tau}{\Gamma \vdash \lambda x.t : \mathcal{M} \to \tau}\ \textsc{t-lam} \qquad \frac{\Gamma \oplus (x : [\sigma_i]_{i \in I}) \vdash t : \tau \quad (\Delta_i \vdash s : \sigma_i)_{i \in I}}{\Gamma +_{i \in I} \Delta_i \vdash t[x \backslash s] : \tau}\ \textsc{t-sub}$$

Note that the axiom typing rule (T-VAR) is *relevant*, in the sense that no extra hypotheses besides the fact that x has type τ are allowed in the typing context. In proof-theory jargon, there is no *weakening*. Moreover, in the rules for application (T-APP) and substitution (T-SUB), the typing context of the conclusion is obtained by joining all of the typing contexts in the premises. In proof-theory jargon, these rules are *multiplicative*, *i.e.* there is no *contraction*[4]. These characteristics of the type system are consistent with the resource aware interpretation of the calculus.

In line with the resource aware interpretation, the typing context Γ in a judgment $\Gamma \vdash t : \tau$ can be understood as follows: given a variable x, each element in the multiset $\Gamma(x)$ concerns

[4]Weakening is the logical rule that allows including unused hypotheses in the context. Contraction is the logical rule that allows conflating repeated occurrences of a hypothesis in the context.

one potential use of this variable in the computation of t. This informal description helps in understanding the rules (T-APP) and (T-SUB), in which several typing judgments are required in the premises for the term s. Each of typing judgment concerns one of the potential uses of s in the computation of the whole term. A particular case of the rules (T-APP) and (T-SUB) is when $I = \varnothing$, *i.e.* there is no potential use of s: the subterm s occurring in the *typed* term $t\,s$ (resp. $t[x\backslash s]$) *does not need to be typed.*

By restricting the $\mathcal{HW}$-system to λ-terms, so that it only contains the rules (T-VAR), (T-LAM), and (T-APP), we obtain the system presented in [58, 34], which we call here *λ-type system.* Following we recall the usual definition of type derivation:

Definition 4.28 (Derivations)**.** A *(type) derivation* is a finite tree obtained by applying the inductive rules of the type system. We write $\Phi \rhd \Gamma \vdash t : \tau$ if Φ is a derivation typing t, *i.e.* ending in the type judgment $\Gamma \vdash t : \tau$. We write $\Phi \rhd_\lambda \Gamma \vdash t : \tau$ if, moreover, Φ is a valid derivation in the λ-type system. A derivation Φ' is an *immediate subderivation* of Φ if, seen as trees, Φ' is one of the children of Φ. A term t is *typable* if there is a derivation typing t. The *size* of a type derivation Φ is a natural number $\mathtt{size}(\Phi)$ denoting the number of nodes of the tree Φ.

The following is an example of a type derivation in the system $\mathcal{HW}$.

Example 4.29 (A type derivation in $\mathcal{HW}$)**.** *Let Ω denote the non-terminating term $(\lambda z.zz)(\lambda z.zz)$. Moreover, let $\tau = [\sigma] \to \sigma$, where σ is an arbitrary type. Let π be the following derivation:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\tau] \vdash x : [\sigma] \to \sigma}\text{T-VAR} \quad \dfrac{}{x : [\sigma] \vdash x : \sigma}\text{T-VAR}}{x : [\tau, \sigma] \vdash xx : \sigma}\text{T-APP}}{x : [\tau, \sigma] \vdash \lambda y.xx : [\,] \to \sigma}\text{T-LAM}}{x : [\tau, \sigma] \vdash (\lambda y.xx)\Omega : \sigma}\text{T-APP}$$

Then we have that:

$$\dfrac{\dfrac{\vdots}{\pi} \quad \dfrac{}{z : [\tau] \vdash z : \tau}\text{T-VAR} \quad \dfrac{}{z : [\sigma] \vdash z : \sigma}\text{T-VAR}}{z : [\tau, \sigma] \vdash ((\lambda y.xx)\Omega)[x\backslash z] : \sigma}\text{T-SUB}$$

Suppose that a typing judgment of the form $\Gamma \vdash t : \sigma$ is derivable in $\mathcal{HW}$. In contrast with what happens in more traditional type systems, the free variables of t do not necessarily appear in the domain of Γ. For example, $x : [\sigma] \vdash (\lambda y.x)z : \sigma$ is derivable in $\mathcal{HW}$ but $z \notin \mathsf{dom}(\Gamma)$. However, $\mathcal{HW}$ does enjoy the following property. From the logical point of view, it states that all the assumptions in the typing context are used at least once:

Lemma 4.30 (Relevance)**.** *If there is a derivation $\Phi \rhd \Gamma \vdash t : \sigma$ then $\mathsf{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.*

Proof. Straightforward by induction on the derivation of the judgment $\Gamma \vdash t : \sigma$. □

It is also worth noticing that not every typable term reduces to a β-normal form. An example is the term $x(\Delta\Delta)$, where $\Delta = \lambda y.yy$, for which there is a type derivation ending with $x : [[\,] \to \alpha] \vdash x(\Delta\Delta) : \alpha$. In order to characterize weak β-normalization by means of typability we need to restrict the types and the type contexts to those that do not have positive occurrences of the constant $[\,]$. To do so, we introduce the following notion of *positive* and *negative* occurrences of a type.

Definition 4.31 (Positive and negative occurrences of types)**.** The set of types that occur with sign $b \in \{+, -\}$ in a type σ (resp. in a multiset of types $\mathcal{M}$, in a context Γ, and in a pair of context and type (Γ, σ)) is written $\mathcal{O}^b(\sigma)$ (resp. $\mathcal{O}^b(\mathcal{M})$, $\mathcal{O}^b(\Gamma)$, and $\mathcal{O}^b(\Gamma \vdash \sigma)$). The set $\mathcal{O}^+(X)$ is the set of types that occur positively in X and $\mathcal{O}^-(X)$ is the set of types that occur negatively in X. We write $\mathcal{O}^\pm(X)$ for either $\mathcal{O}^+(X)$ or $\mathcal{O}^-(X)$ and $\mathcal{O}^\mp(\ldots)$ for the opposite set in a given rule. All of these sets are defined mutually inductively by the following conditions, where $\mathtt{T}$ denotes either a type or a multiset of types:

$$\frac{}{\sigma \in \mathcal{O}^+(\sigma)} \qquad \frac{}{\mathcal{M} \in \mathcal{O}^+(\mathcal{M})}$$

$$\frac{\mathtt{T} \in \mathcal{O}^\pm(\sigma_i) \quad I \neq \varnothing}{\mathtt{T} \in \mathcal{O}^\pm([\sigma_i]_{i\in I})} \qquad \frac{\mathtt{T} \in \mathcal{O}^\mp(\mathcal{M})}{\mathtt{T} \in \mathcal{O}^\pm(\mathcal{M} \to \tau)} \qquad \frac{\mathtt{T} \in \mathcal{O}^\pm(\tau)}{\mathtt{T} \in \mathcal{O}^\pm(\mathcal{M} \to \tau)}$$

$$\frac{y \in \mathsf{dom}(\Gamma) \quad \mathtt{T} \in \mathcal{O}^\mp(\Gamma(y))}{\mathtt{T} \in \mathcal{O}^\pm(\Gamma)} \qquad \frac{\mathtt{T} \in \mathcal{O}^\pm(\Gamma)}{\mathtt{T} \in \mathcal{O}^\pm(\Gamma \vdash \tau)} \qquad \frac{\mathtt{T} \in \mathcal{O}^\pm(\tau)}{\mathtt{T} \in \mathcal{O}^\pm(\Gamma \vdash \tau)}$$

Example 4.32 (Positive and negative occurrences)**.** *The following hold:*

- $[\,] \in \mathcal{O}^+([\,])$
- $[\,] \in \mathcal{O}^-([\,] \to \sigma)$
- $[\,] \in \mathcal{O}^+(x : [[\,] \to \sigma])$
- $[\,] \in \mathcal{O}^+(x : [[\,] \to \sigma] \vdash \sigma)$

It is an already known fact that the type system $\mathcal{HW}$, restricted to contexts and types in which there are *no* positive occurrences of the empty multiset $[]$, can be used to characterize weakly normalizing terms of the λ-calculus:

Theorem 4.33 (Characterization of weakly normalizing terms in the λ-calculus)**.** *Let t be a λ-term. Then the following are equivalent:*

1. *The term is weakly normalizing,* i.e. $t \in \mathtt{WN}(\to_\beta)$.
2. *The judgment $\Gamma \vdash t : \tau$ is derivable in $\mathcal{HW}$ and $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$*

Proof. A straightforward adaptation of [96] to the non-idempotent case. See [31] for details. □

Extending Typing to Contexts

As mentioned before, we use system $\mathcal{HW}$ as a tool to characterize the set of terms that are weakly normalizing in the Theory of Sharing, in order to relate them with the set of terms that are weakly normalizing in the pure λ-calculus. In order to be able to prove this result for the Theory of Sharing, whose rules operate *at a distance*, a key technical tool is the extension of the typing system given in Def. 4.27 with typing rules for *substitution contexts*.

If $\mathtt{L}$ is a substitution context, we write $\mathsf{dom}(\mathtt{L})$ for the variables bound by $\mathtt{L}$, and $\mathtt{fv}(\mathtt{L})$ for the free variables of $\mathtt{L}$, taking $\mathtt{fv}(\Box) = \varnothing$. Moreover, we use the following notion of *height*:

Definition 4.34. The *height* of a substitution context is defined by:

$$\mathtt{height}(\Box) \stackrel{\text{def}}{=} 1 \quad \mathtt{height}(\mathtt{L}[x\backslash t]) \stackrel{\text{def}}{=} \mathtt{height}(\mathtt{L}) + 1$$

Definition 4.35 (Extension of $\mathcal{HW}$ for substitution contexts)**.** The type system $\mathcal{HW}$ is extended with typing judgments of the form $\Gamma \Vdash \mathtt{L} \rhd \Delta$, where Γ and Δ are typing contexts and $\mathtt{L}$ is a substitution context. The left-hand side Γ of a judgment $\Gamma \Vdash \mathtt{L} \rhd \Delta$ is a typing context for the (typed) free variables of $\mathtt{L}$, while the right-hand side Δ is a typing context for the term which will be plugged into the hole of $\mathtt{L}$. There are two typing rules:

$$\frac{}{\varnothing \Vdash \Box \rhd \varnothing} \qquad \frac{\Gamma \oplus x : [\sigma_i]_{i\in I} \Vdash \mathtt{L} \rhd \Delta \quad x \notin \mathsf{dom}\Delta \quad (\Sigma_k \vdash t : \sigma_k)_{k\in I \uplus J}}{\Gamma +_{k\in I\uplus J} \Sigma_k \Vdash \mathtt{L}[x\backslash t] \rhd \Delta \oplus x : [\sigma_j]_{j\in J}}$$

In the second rule, the sets of indices I and J are supposed to be disjoint.

Note that in the second type rule the context $(\Sigma_i)_{i\in I}$ is used to type the copies of t associated with the free occurrences of x in the list $\mathtt{L}$, while the context $(\Sigma_j)_{j\in J}$ is used to type the copies of t associated with the free occurrences of x in the term which will fill the hole of $\mathtt{L}$.

Example 4.36. *Let π be the following typing derivation for $[x\backslash yz]$:*

$$\dfrac{\dfrac{}{\varnothing \Vdash \Box \rhd \varnothing} \quad \dfrac{\vdots}{y : [[\gamma_1, \gamma_2] \to \alpha], z : [\gamma_1, \gamma_2] \vdash yz : \alpha}}{y : [[\gamma_1, \gamma_2] \to \alpha], z : [\gamma_1, \gamma_2] \Vdash [x\backslash yz] \rhd x : [\alpha]}$$

Then the following is a typing derivation for $[x\backslash yz][y\backslash z]$:

$$\dfrac{\dfrac{\vdots}{\pi} \quad \dfrac{}{z : [[\gamma_1, \gamma_2] \to \alpha] \vdash z : [\gamma_1, \gamma_2] \to \alpha} \quad \dfrac{}{z : [\beta] \vdash z : \beta}}{z : [[\gamma_1, \gamma_2] \to \alpha, \beta, \gamma_1, \gamma_2] \Vdash [x\backslash yz][y\backslash z] \rhd x : [\alpha], y : [\beta]}$$

The following lemma states a few properties that may be easily proved by induction on $\mathtt{L}$.

Lemma 4.37 (Properties of type derivations of substitution contexts)**.**

1. *If* $\Gamma \Vdash \mathtt{L} \rhd \Delta$ *then* $\mathsf{dom}(\Gamma) \subseteq \mathtt{fv}(\mathtt{L})$ *and* $\mathsf{dom}(\Delta) \subseteq \mathsf{dom}(\mathtt{L})$.

2. *There is a derivation* $\Phi_{t\mathtt{L}} \rhd \Lambda \vdash t\mathtt{L} : \sigma$ *if and only if there are contexts* Γ, Δ, Π *such that* $\Lambda = \Gamma + \Pi$*, and there are derivations* $\Phi_{\mathtt{L}} \rhd \Gamma \Vdash \mathtt{L} \rhd \Delta$ *and* $\Phi_t \rhd \Delta; \Pi \vdash t : \sigma$*. Moreover,* $\mathtt{size}(\Phi_{\mathtt{L}[t]}) = \mathtt{size}(\Phi_{\mathtt{L}}) + \mathtt{size}(\Phi_t) - 1$.

3. *If* $(\Phi^j_{\mathtt{L}} \rhd \Gamma_j \Vdash \mathtt{L} \rhd \Delta_j)_{j\in J}$*, then* $\Phi_{\mathtt{L}} \rhd +_{j\in J}\Gamma_j \Vdash \mathtt{L} \rhd +_{j\in J}\Delta_j$*. Moreover,* $\mathtt{size}(\Phi_{\mathtt{L}}) = +_{j\in J}\mathtt{size}(\Phi^j_{\mathtt{L}}) - (\mathtt{height}(\mathtt{L}) \cdot (|J| - 1))$.

The second item of the lemma allows one to decompose the type derivation of a term $t\mathtt{L}$ into two type derivations, one for the context $\mathtt{L}$ and another one for the term t. Reciprocally, context and term derivations can be combined if their types coincide.

On the other hand, the third item of the lemma states that combining different derivable typing judgments of the same substitution context by means of multiset union yields a derivable typing judgment. Moreover, their sizes can be related using the notion of height. Observe that the statement includes the case $J = \varnothing$.

Typability Implies Normalization

Our goal is now to show that terms typable in system $\mathcal{HW}$ are weakly normalizing in the Theory of Sharing. The key technical result is the property known as *weighted subject reduction*. Recall that, in traditional type systems such as the simply typed λ-calculus, the *subject reduction* property states that evaluation preserves types. More precisely, if there is a typing derivation $\Phi \rhd \Gamma \vdash t : \tau$ and a reduction step $t \to t'$ then there is also a typing derivation $\Phi' \rhd \Gamma \vdash t' : \tau$. The weighted subject reduction property states that, assuming further appropriate conditions on the step $t \to t'$, one may also ensure that $\mathtt{size}(\Phi) > \mathtt{size}(\Phi')$.

In our case, we will be able to ensure that the size of the derivation decreases as long as we select a step $t \to t'$ contracting a *typed redex*. Intuitively, from the point of view of the resource aware interpretation, a redex is typed if it lies inside a subterm that will be used at some point in the evaluation of t[5]. For example, the underlined redex $R : f(\underline{(\lambda x.x)y}) \to f\, y$ is typed if the type of the function f is, say, $[\alpha] \to \alpha$, whereas the redex R is untyped if the type of the function f is, say, $[] \to \alpha$.

To define this more precisely, we introduce the notion of *typed occurrences* of a term (abbreviated as T-occurrences). Intuitively, a typed occurrence of t is a position identifying a subterm that will be used at some point in the evaluation of t. We start by recalling the notion of position:

Definition 4.38 (Positions of a term)**.** The set of *positions* of a term t, written $\mathtt{pos}(t)$, is the set of finite words over the alphabet $\{0, 1\}$, inductively defined as follows:

$$\frac{}{\epsilon \in \mathtt{pos}(t)} \quad \frac{p \in \mathtt{pos}(t)}{0p \in \mathtt{pos}(\lambda x.t)} \quad \frac{p \in \mathtt{pos}(t)}{0p \in \mathtt{pos}(ts)} \quad \frac{p \in \mathtt{pos}(t)}{1p \in \mathtt{pos}(st)} \quad \frac{p \in \mathtt{pos}(t)}{0p \in \mathtt{pos}(t[x\backslash s])} \quad \frac{p \in \mathtt{pos}(t)}{1p \in \mathtt{pos}(s[x\backslash t])}$$

The set of positions of a context $\mathtt{C}$ is defined similarly. The subterm of t (resp. $\mathtt{C}$) at position p is written $t|_p$ (resp. $\mathtt{C}|_p$) and defined as expected.

[5] In fact, these intuitions can be formalized; see for example [90].

For example, given $t = x[z\backslash z'](\lambda y.y)$ and $\mathtt{C} = (\lambda x.\Box)yz$, we have that the sets of positions are $\mathtt{pos}(t) = \{\epsilon, 0, 00, 01, 1, 10\}$ and $\mathtt{pos}(\mathtt{C}) = \{\epsilon, 0, 00, 000, 01, 1\}$. Moreover, $t|_1 = \lambda y.y$ and $\mathtt{C}|_{000} = \Box$.

Definition 4.39 (T-occurrence)**.** Suppose given a derivation $\Phi \rhd \Gamma \vdash t : \tau$. A position $p \in \mathtt{pos}(t)$ is a *T-occurrence* of t in Φ if either $p = \epsilon$, or $p = ip'$ $(i = 0, 1)$ and $p' \in \mathtt{pos}(t|_i)$ is a T-occurrence of $t|_i$ in *some* of the immediate subderivations of Φ. A redex occurrence of t which is a T-occurrence of t in Φ is said to be a *redex T-occurrence* of t in Φ.

For example, given the following derivation Φ', we have that ϵ, 0, 1 and 10 are T-occurrences of $x(yz)$ in Φ', while 11 is not a T-occurrence of $x(yz)$ in Φ'.

$$\Phi' \rhd \dfrac{\dfrac{}{x{:}[[\tau,\tau] \to \tau] \vdash x : [\tau,\tau] \to \tau} \quad \dfrac{y{:}[[\,] \to \tau] \vdash y : [\,] \to \tau}{y{:}[[\,] \to \tau] \vdash yz : \tau} \quad \dfrac{y{:}[[\,] \to \tau] \vdash y : [\,] \to \tau}{y{:}[[\,] \to \tau] \vdash yz : \tau}}{x{:}[[\tau,\tau] \to \tau], y{:}[[\,] \to \tau, [\,] \to \tau] \vdash x(yz) : \tau}$$

Note that if an occurrence of a variable x is a T-occurrence of t in Φ, then x occurs free in t. Given $\Phi \rhd \Gamma \vdash t : \tau$, the *no-redex-occurrences* predicate $A(t, \Phi)$ holds if and only if t has no $\mathtt{sh}$-redex T-occurrences in Φ.

The following lemma studies the relation between typing derivations and the substitution of a single occurrence of a variable by a term, namely a typing derivation for $\mathtt{C}\langle\!\langle t \rangle\!\rangle$ may be constructed by combining a typing derivation for $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ and typing derivations for t.

Lemma 4.40 (Partial Substitution)**.** *If* $\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle} \rhd x{:}[\sigma_i]_{i\in I}; \Gamma \vdash \mathtt{C}\langle\!\langle x \rangle\!\rangle : \tau$ *and* $(\Phi^i_u \rhd \Delta_i \vdash u : \sigma_i)_{i\in I}$ *then* $\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle} \rhd x{:}[\sigma_i]_{i\in I\setminus K}; \Gamma +_{k\in K} \Delta_k \vdash \mathtt{C}\langle\!\langle u \rangle\!\rangle : \tau$*, for some* $K \subseteq I$ *where* $\mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle}) = \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}) +_{k\in K} \mathtt{size}(\Phi^k_u) - |K|$*. Moreover, if* $p \in \mathtt{pos}(\mathtt{C})$ *is the occurrence of the hole in* $\mathtt{C}$ *and* p *is a T-occurrence of* $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ *in* $\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}$*, then* $K \neq \varnothing$*.*

Proof. By induction on the typing derivation $\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}$. □

Using this tool we are able to prove the following key result:

Lemma 4.41 (Weighted Subject Reduction for $\mathtt{sh}$)**.** *Let* $\Phi \rhd \Gamma \vdash t : \tau$*. If* $t \to_{\mathtt{sh}} t'$ *reduces a* $\mathtt{sh}$*-redex T-occurrence of* t *in* Φ*, then there exists* Φ' *such that* $\Phi' \rhd \Gamma \vdash t' : \tau$ *and* $\mathtt{size}(\Phi) > \mathtt{size}(\Phi')$*.*

Proof. By induction on the context under which the step $t \to_{\mathtt{sh}} t'$ takes place. The inductive cases are straightforward by *i.h.*. The interesting case is the base case, when the step takes place at the root. Then we consider three subcases, depending on the kind of redex contracted.

1. $\mathtt{db}$ *step, i.e.* $t = (\lambda x.u)\mathtt{L}s \to_{\mathtt{db}} u[x/s]\mathtt{L} = t'$. Then the reduction concerns a $\mathtt{db}$-redex T-occurrence of t in Φ. Then one may show $\Phi' \rhd \Gamma \vdash t' : \tau$ and $\mathtt{size}(\Phi) > \mathtt{size}(\Phi')$ by induction on $\mathtt{L}$.

2. $\mathtt{lsv}$ *step, i.e.* $t = \mathtt{C}\langle\!\langle x \rangle\!\rangle[x/u\mathtt{L}] \to_{\mathtt{lsv}} \mathtt{C}\langle\!\langle u \rangle\!\rangle[x/u]\mathtt{L} = t'$. Then the derivation Φ has the following form, where $\Gamma = \Gamma_0 +_{i\in I} \Delta_i$.

$$\frac{\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle} \rhd x : [\sigma_i]_{i\in I}; \Gamma_0 \vdash \mathtt{C}\langle\!\langle x \rangle\!\rangle : \sigma \qquad \left(\Phi^i_{u\mathtt{L}} \rhd \Delta_i \vdash u\mathtt{L} : \sigma_i\right)_{i\in I}}{\Gamma_0 +_{i\in I} \Delta_i \vdash \mathtt{C}\langle\!\langle x \rangle\!\rangle[x/u\mathtt{L}] : \sigma}$$

By Lem. 4.37, for all $i \in I$, there exist $\Pi^i_1, \Pi^i_2, \Pi^i_3$ such that $\Phi^i_{\mathtt{L}} \rhd \Pi^i_1 \Vdash \mathtt{L} \rhd \Pi^i_2$, $\Phi^i_u \rhd \Pi^i_2; \Pi^i_3 \vdash u : \sigma_i$ and $\Delta_i = \Pi^i_1 + \Pi^i_3$.

From the derivations $\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}$ and $(\Phi^i_u)_{i\in I}$ we get, by Lem. 4.40, a derivation $\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle} \rhd x : [\sigma_i]_{i\in I\setminus K}; \Gamma_0 +_{k\in K} (\Pi^k_2; \Pi^k_3) \vdash \mathtt{C}\langle\!\langle u \rangle\!\rangle : \sigma$ for some $K \subseteq I$. So we can construct the following derivation $\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle[x/u]}$.

$$\frac{\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle} \qquad \left(\Phi^i_u\right)_{i\in I\setminus K}}{\Gamma_0 +_{k\in K} (\Pi^k_2; \Pi^k_3) +_{i\in I\setminus K} (\Pi^i_2; \Pi^i_3) \vdash \mathtt{C}\langle\!\langle u \rangle\!\rangle[x/u] : \sigma}$$

The last sequent can be written $\Gamma_0 + (+_{i\in I}\Pi^i_2; +_{i\in I}\Pi^i_3) \vdash \mathtt{C}\langle\!\langle u \rangle\!\rangle[x/u] : \sigma$.

We thus apply Lem. 4.37 to $(\Phi^i_{\mathtt{L}})_{i\in I}$ and we get $\Phi_{\mathtt{L}} \rhd +_{i\in I}\Pi^i_1 \Vdash \mathtt{L} \rhd +_{i\in I}\Pi^i_2$. We can thus apply Lem. 4.37 to $\Phi_{\mathtt{L}}$ and $\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle[x/u]}$, obtaining $\Phi' \rhd \Gamma_0 +_{i\in I} \Pi^1_1 +_{i\in I} \Pi^i_3 \vdash \mathtt{C}\langle\!\langle u \rangle\!\rangle[x/u]\mathtt{L} : \sigma$.

We can then conclude with the first statement since $\Gamma_0 +_{i\in I} \Pi^1_1 +_{i\in I} \Pi^i_3 = \Gamma_0 +_{i\in I} \Delta_i = \Gamma$ as required. Moreover, for the second one, we assume that the reduction step concerns a $\mathtt{sh}$-redex T-occurrence of t in Φ. Then,

$$\begin{array}{rcl}
\mathtt{size}(\Phi) & = & \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}) +_{i\in I} \mathtt{size}(\Phi^i_{u\mathtt{L}}) + 1 \\
& =_{L.\ 4.37} & \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}) +_{i\in I} (\mathtt{size}(\Phi^i_{\mathtt{L}}) + \mathtt{size}(\Phi^i_u) - 1) + 1 \\
& = & \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}) +_{i\in I} \mathtt{size}(\Phi^i_{\mathtt{L}}) +_{i\in I} \mathtt{size}(\Phi^i_u) - (|I| - 1) \\
& = & Z - (|I| - 1)
\end{array}$$

and

$$\begin{array}{rcl}
\mathtt{size}(\Phi') & =_{L.\ 4.37} & \mathtt{size}(\Phi_{\mathtt{L}}) + \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle[x/u]}) - 1 \\
& = & \mathtt{size}(\Phi_{\mathtt{L}}) + \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle}) +_{i\in I\setminus K} \mathtt{size}(\Phi^i_u) + 1 - 1 \\
& = & \mathtt{size}(\Phi_{\mathtt{L}}) + \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle}) +_{i\in I\setminus K} \mathtt{size}(\Phi^i_u) \\
& =_{L.\ 4.40} & \mathtt{size}(\Phi_{\mathtt{L}}) + \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}) +_{k\in K} \mathtt{size}(\Phi^k_u) - |K| +_{i\in I\setminus K} \mathtt{size}(\Phi^i_u) \\
& =_{L.\ 4.37} & +_{i\in I}\mathtt{size}(\Phi^i_{\mathtt{L}}) - [\mathtt{height}(\mathtt{L}) \cdot (|I| - 1)] + \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}) +_{i\in I} \mathtt{size}(\Phi^i_u) - |K| \\
& = & \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}) +_{i\in I} \mathtt{size}(\Phi^i_{\mathtt{L}}) +_{i\in I} \mathtt{size}(\Phi^i_u) - [\mathtt{height}(\mathtt{L}) \cdot (|I| - 1)] - |K| \\
& = & Z - [\mathtt{height}(\mathtt{L}) \cdot (|I| - 1)] - |K|
\end{array}$$

We know by Lem. 4.40 that $K \neq \varnothing$. Therefore, $|I| - 1 \leqslant \mathtt{height}(\mathtt{L}) \cdot (|I| - 1)$ so that $Z - (|I| - 1) \geqslant Z - [\mathtt{height}(\mathtt{L}) \cdot (|I| - 1)] > Z - [\mathtt{height}(\mathtt{L}) \cdot (|I| - 1)] - |K|$. We thus conclude $\mathtt{size}(\Phi) > \mathtt{size}(\Phi')$ as required.

3. $\mathtt{gc}$ *step.* Immediate.

□

We now relate the notions of T-occurrence and $\mathtt{sh}$-normal form, before concluding with the main result of this subsection.

Lemma 4.42. *Let $\Phi \rhd \Gamma \vdash t : \tau$ such that $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$. Then $A(t, \Phi)$ implies $t \in \mathtt{NF}(\to_{\mathtt{sh}})$.*

Proof. Let $\Phi \rhd \Gamma \vdash t : \tau$ such that $A(t, \Phi)$. First show the following more general property by induction on Φ.

1. If $[\,] \notin \mathcal{O}^+(\Gamma)$, and t is not an answer, then $t \in \overline{\mathtt{S}}$.
2. If $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$, and t is an answer, then $t \in \overline{\mathtt{N}}$.

Moreover, $x \in \mathtt{fv}(t)$ implies x has some T-occurrence in Φ.

Now, suppose $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$. Thus in particular $[\,] \notin \mathcal{O}^+(\Gamma)$. If t is not an answer, then one easily shows that $t \in \overline{\mathtt{S}}$, which gives $t \in \overline{\mathtt{N}}$ since $\overline{\mathtt{S}} \subseteq \overline{\mathtt{N}}$; If t is an answer, then one easily shows $t \in \overline{\mathtt{N}}$. We conclude that $t \in \mathtt{NF}(\to_{\mathtt{sh}})$ by Lem. 4.7. □

Theorem 4.43 (Typability implies sh-normalization)**.** *Let $\Phi \rhd \Gamma \vdash t : \tau$ such that $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$. Then t is weakly normalizing in the Theory of Sharing.*

Proof. Let $\Phi \rhd \Gamma \vdash t : \tau$ such that $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$. By Lem. 4.41 and Lem. 4.42 we can construct a *finite* sh-reduction sequence which only reduces sh-redex T-occurrences, *i.e.* there exist $t_0, t_1, \ldots, t_n$ such that (1) $t = t_0$ and $\Phi = \Phi_0$, (2) $\Phi_i \rhd \Gamma \vdash t_i : \tau$, (3) $t_i \to_{\mathtt{sh}} t_{i+1}$ reduces a sh-redex T-occurrences of t_i in Φ_i, and (5) $A(t_n, \Phi_n)$ holds. This together with $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$ gives $t_n \in \mathtt{NF}(\to_{\mathtt{sh}})$ by Lem. 4.42. We thus conclude $t \in \mathtt{WN}(\to_{\mathtt{sh}})$. □

4.3.2 Completeness of the Theory of Sharing

In this section we prove Fig. 4.1(b), that is completeness of the Theory of Sharing with respect to β-reduction in the λ-calculus. Before doing so, we need to state a few basic properties of unfolding.

Lemma 4.44. *Let $t, s \in \mathcal{T}$ be terms, possibly with explicit substitutions. Then:*

1. *If $t \to_{\mathtt{sh}} s$, then $t^\diamond \twoheadrightarrow_\beta s^\diamond$.*
2. *If $t \in \mathtt{NF}(\to_{\mathtt{sh}})$, then $t^\diamond \in \mathtt{NF}(\to_\beta)$.*

Recall that $\mathtt{NF}(\to)$ stands for the set of $\to$-normal forms.

Proof. By induction on t. □

Indeed, to illustrate the first point we have $t = y[y\backslash(\lambda z.zz)(II)] \to_{\mathtt{db}} y[y\backslash(zz)[z\backslash II]] = u$ and $t^\diamond = (\lambda z.zz)(II) \to_\beta (II)\ (II) = u^\diamond$, and to illustrate the second one we have $t = x[y\backslash I[w'\backslash I]][z\backslash I] \in \mathtt{NF}(\to_{\mathtt{sh}})$ and $t^\diamond = x \in \mathtt{NF}(\to_\beta)$.

We now conclude with the completeness result for the sh-calculus, *cf.* Fig. 4.1(b):

Proposition 4.45 (Completeness of the Theory of Sharing)**.** *If $t \twoheadrightarrow_\beta s \in \mathtt{NF}(\to_\beta)$ then there exists a term u such that $t \twoheadrightarrow_{\mathtt{sh}} u \in \mathtt{NF}(\mathtt{sh})$ and $u^\diamond = s$.*

Proof. Let $t \twoheadrightarrow_\beta \mathtt{nf}_\beta$, where $\mathtt{nf}_\beta$ is in β-nf. Then $\Phi \rhd \Gamma \vdash t : \tau$ and $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$ by Thm. 4.33. But then t is weakly sh-normalizing by Thm. 4.43, so that $t \twoheadrightarrow_{\mathtt{sh}} \mathtt{nf}_{\mathtt{sh}}$, where $\mathtt{nf}_{\mathtt{sh}}$ is in sh-nf. By Lem. 4.44(1) $t^\diamond \twoheadrightarrow_\beta \mathtt{nf}_{\mathtt{sh}}^\diamond$ and by Lem. 4.44(2) $\mathtt{nf}_{\mathtt{sh}}^\diamond \in \mathtt{NF}(\to_\beta)$. Since $t^\diamond = t \twoheadrightarrow_\beta \mathtt{nf}_\beta$ and $t^\diamond \twoheadrightarrow_\beta \mathtt{nf}_{\mathtt{sh}}^\diamond$, then we conclude $\mathtt{nf}_{\mathtt{sh}}^\diamond = \mathtt{nf}_\beta$ because $\to_\beta$ is CR. □

4.3.3 Factorization of the Theory of Sharing

In this section we prove Fig. 4.1(c), that is, factorization of the Theory of Sharing. For this, we show that $\to_{\mathtt{sh}}$ reduction steps which are not $\overset{\vartheta}{\rightsquigarrow}$ steps can always be postponed after $\overset{\vartheta}{\rightsquigarrow}$ reduction steps, that this postponement process terminates, and that, ultimately, all remaining non-$\overset{\vartheta}{\rightsquigarrow}$ steps are erasable by $\mathtt{gc}$ (and thus erased by the unfolding $\_^{\diamond}$). More precisely we proceed in three stages:

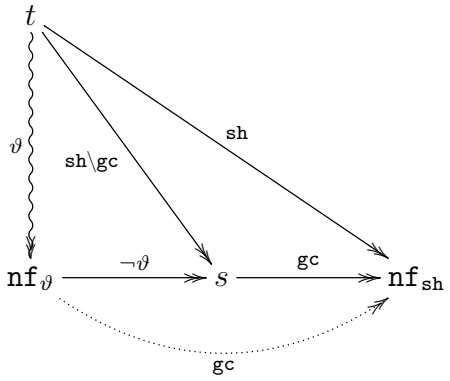


Figure 4.2: Decomposition of Fig. 4.1(c)

1. As a preliminary step, we get $\mathtt{gc}$-steps out of the way: any $\to_{\mathtt{sh}}$ reduction sequence can be factorized into a $\to_{\mathtt{sh}}$ reduction sequence without $\mathtt{gc}$-steps, which we call *strict*, followed by a sequence of $\mathtt{gc}$-steps (*cf.*). The relation of strict reduction is written $\to_{\mathtt{sh}\backslash\mathtt{gc}}$.

2. Then we prove a more involved commutation result: $\to_{\mathtt{sh}}$-reductions without $\mathtt{gc}$-steps can be factored in two parts (*cf.* Prop. 4.51):

   2.1 a sequence of *external* $\to_{\mathtt{sh}}$-steps, which correspond to the strategy $\overset{\vartheta}{\rightsquigarrow}$

   2.2 a sequence of *internal* $\to_{\mathtt{sh}}$-steps, which are not in the strategy $\overset{\vartheta}{\rightsquigarrow}$, written $\xrightarrow{\neg\vartheta}_{\mathtt{sh}}$.

   The proof relies on an abstract factorization result by Accattoli [3]. We write $\xrightarrow{\neg\vartheta}_{\mathtt{sh}}$ instead of $\to_{\mathtt{sh}}$ for such internal steps. Two examples of internal steps are $(xx)[x\backslash I] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (x\ I)[x\backslash I]$ and $(I\ x)[x\backslash I\ I] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (I\ x)[x\backslash z[z\backslash I]]$, where we substitute a value for a variable occurrence that is not focused, or evaluate a substitution whose bound variable is not focused.

3. Finally, we show that internal steps that remain after the $\overset{\vartheta}{\rightsquigarrow}$-normal form is reached only take place inside garbage substitutions, that are removed by the unfolding operation $\_^{\diamond}$ (*cf.* Lem. 4.53).

Postponement of gc

In this subsection we show the reasonable observation that garbage collection steps can always be postponed to the end of a reduction sequence. Reduction steps (resp. sequences) that do not use the gc-rule are called *strict* and are written $\to_{\mathtt{sh}\backslash\mathtt{gc}}$ (resp. $\twoheadrightarrow_{\mathtt{sh}\backslash\mathtt{gc}}$).

Lemma 4.46 (Postponement of gc)**.** *If* $t \twoheadrightarrow_{\mathtt{sh}} s$*, then there is* u *such that* $t \twoheadrightarrow_{\mathtt{sh}\backslash\mathtt{gc}} u \twoheadrightarrow_{\mathtt{gc}} s$*.*

Proof. The proof is by exhaustive case analysis of the relative positions of a gc step followed by a non-gc step, similar to other proofs of postponement of gc in the LSC (see for instance Lem. 6.50). □

Observe that the fact that $s \in \mathtt{NF}(\to_{\mathtt{sh}})$ does not imply that $u \in \mathtt{NF}(\to_{\mathtt{sh}\backslash\mathtt{gc}})$ in general. Indeed, if we take $t = x[y\backslash\Omega]$ and $s = x$, then $u = x[y\backslash\Omega]$, which is not even normalizing for $\to_{\mathtt{sh}\backslash\mathtt{gc}}$. The actual relation of postponement of gc with normal forms is stated in Lem. 4.53.

Factorization of Strict Reduction

In this subsection, we show that a sequence of strict reduction steps $\to_{\mathtt{sh}\backslash\mathtt{gc}}$ can always be factorized as a sequence of steps in the strategy ($\overset{\vartheta}{\rightsquigarrow}$) followed by steps which are not in the strategy ($\xrightarrow{\neg\vartheta}_{\mathtt{sh}}$). More precisely, we say that t_1 reduces in a ϑ*-internal step* to t_2, written $t_1 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_2$, if and only if there is a step in the strict Theory of Sharing that is not a step in the strong call-by-need strategy, *i.e.* $t_1 \ (\to_{\mathtt{sh}\backslash\mathtt{gc}} \setminus \overset{\vartheta}{\rightsquigarrow})\ t_2$. We sometimes call ϑ-internal steps just *internal steps* if ϑ is clear from the context. Steps in the strategy $\overset{\vartheta}{\rightsquigarrow}$ are called ϑ*-external steps* (or just *external steps*).

The proof of factorization is long and technical. We begin by recalling the definition of *square factorization system* and an abstract factorization result due to Accattoli [3]:

Definition 4.47 (Square factorization system)**.** A *square factorization system* is given by a set X and four reduction relations $(\rightsquigarrow_\bullet, \rightsquigarrow_\circ, \mapsto_\bullet, \mapsto_\circ)$ such that:

1. **Termination**: $\rightsquigarrow_\circ$ and $\mapsto_\circ$ are strongly normalizing.
2. **Row-swap 1**: $(\rightsquigarrow_\bullet \rightsquigarrow_\circ) \subseteq (\rightsquigarrow_\circ^+ \rightsquigarrow_\bullet^*)$.
3. **Row-swap 2**: $(\mapsto_\bullet \mapsto_\circ) \subseteq (\mapsto_\circ^+ \mapsto_\bullet^*)$.
4. **Diagonal-swap 1**: $(\mapsto_\bullet \rightsquigarrow_\circ) \subseteq (\rightsquigarrow_\circ \mapsto^*)$.
5. **Diagonal-swap 2**: $(\rightsquigarrow_\bullet \mapsto_\circ) \subseteq (\mapsto_\circ \rightsquigarrow^*)$.

with the following notation:

$$\rightsquigarrow \overset{\text{def}}{=} (\rightsquigarrow_\bullet \cup \rightsquigarrow_\circ) \qquad \mapsto \overset{\text{def}}{=} (\mapsto_\bullet \cup \mapsto_\circ)$$
$$\to_\bullet \overset{\text{def}}{=} (\rightsquigarrow_\bullet \cup \mapsto_\bullet) \qquad \to_\circ \overset{\text{def}}{=} (\rightsquigarrow_\circ \cup \mapsto_\circ)$$

$$\to \overset{\text{def}}{=} (\to_\bullet \cup \to_\circ)$$

Theorem 4.48 (Abstract factorization, Accattoli 2012)**.** *Let* $(\rightsquigarrow_\bullet, \rightsquigarrow_\circ, \mapsto_\bullet, \mapsto_\circ)$ *be a square factorization system. Then* $\to^* \subseteq (\to_\circ^* \to_\bullet^*)$.

Proof. See [3, Theorem 5.2]. □

Below we state the two main lemmas, *Backward stability by internal steps* and *Postponement of internal steps*, whose full proofs may be found in the appendix (Section A.2.5). The following lemma states that important notions of the strong call-by-need strategy, such as *answers*, *normal forms*, and *evaluation contexts*, are preserved by expansion via internal steps:

Lemma 4.49 (Backward stability by internal steps — ♣)**.** *Let* $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t$ *be a* ϑ*-internal step. Then:*

1. *If* t *is an answer (resp. a* `db`*-redex) then* t_0 *is also an answer (resp. a* `db`*-redex).*
2. *If* t *is a* ϑ*-normal form (resp.* ϑ*-structure) then* t_0 *is also a* ϑ*-normal form (resp.* ϑ*-structure).*
3. *If* $t = \mathtt{C}\langle\!\langle x \rangle\!\rangle$ *where* `C` *is a* ϑ*-evaluation context (resp. inert* ϑ*-evaluation context), then* t_0 *is also of the form* $\mathtt{C}_0\langle\!\langle x \rangle\!\rangle$*, where* $\mathtt{C}_0$ *is a* ϑ*-evaluation context (resp. inert* ϑ*-evaluation context).*

Proof. See Section A.2.5 in the appendix. □

The following key lemma states that an external step can be commuted before an internal step. In particular, an internal step cannot create an external step (neither by creating a redex in an external position, nor by turning an internal position into an external one).

Lemma 4.50 (Postponement of internal steps — ♣ Lem. A.73)**.** *Let* $\mathtt{fv}(t_0) \subseteq \vartheta$*. If* $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_1 \overset{\vartheta}{\rightsquigarrow} t_3$*, then there is a term* t_2 *such that* $t_0 \overset{\vartheta}{\rightsquigarrow}{}^* t_2 \twoheadrightarrow_{\mathtt{sh}}^{\neg\vartheta} t_3$*, where the reduction from* t_0 *to* t_2 *includes at least one step and the one from* t_2 *to* t_3 *has at most two steps.*

Proof. The proof is by induction on the evaluation context defining the external step and then by case analysis on the position of the internal step relative to this evaluation context. See Section A.2.5 in the appendix. □

Proposition 4.51 (External–internal factorization)**.** *Let* $\mathtt{fv}(t) \subseteq \vartheta$*. If* $t \twoheadrightarrow_{\mathtt{sh}\setminus\mathtt{gc}} r$ *then there is* u *such that* $t \overset{\vartheta}{\rightsquigarrow}{}^* u \twoheadrightarrow_{\mathtt{sh}}^{\neg\vartheta} r$.

Proof. This is a consequence of Thm. 4.48 and Lem. 4.50. Indeed, by the construction given in the proof of Lem. 4.50 one has that $(\xrightarrow{\neg\vartheta}_{\mathtt{shdb}}, \overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}}, \xrightarrow{\neg\vartheta}_{\mathtt{shlsv}}, \overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}})$ forms a square factorization system, taking $\overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}}$ (resp. $\overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}}$) to be the external `db` (resp. `lsv`) reduction, and $\xrightarrow{\neg\vartheta}_{\mathtt{shdb}}$ (resp. $\xrightarrow{\neg\vartheta}_{\mathtt{shlsv}}$) to be the internal `db` (resp. `lsv`) reduction. □

Erasure of Final Internal Steps

The previous two subsections ensure that any $\to_{\mathtt{sh}}$ reduction sequence can be factored into a $\overset{\vartheta}{\rightsquigarrow}$ reduction prefix followed by internal or gc steps. Here we further show that if the $\to_{\mathtt{sh}}$ reduction sequence reaches a $\to_{\mathtt{sh}}$-normal form, then all the internal steps factored out by Prop. 4.51 can be erased by gc steps.

Lemma 4.52 (Inclusion of normal forms)**.** *Let ϑ, t be such that $\mathtt{fv}(t) \subseteq \vartheta$. If $t \in \mathtt{NF}(\to_{\mathtt{sh}})$, then $t \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$.*

Proof. This is immediate since $\overset{\vartheta}{\rightsquigarrow} \subseteq \to_{\mathtt{sh}}$. □

Lemma 4.53 (Normal forms modulo internal and gc steps)**.** *Let ϑ, t be such that $\mathtt{fv}(t) \subseteq \vartheta$.*

1. *If $t \to_{\mathtt{gc}} \mathtt{nf}_\vartheta$ with $\mathtt{nf}_\vartheta \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ then $t \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$.*
2. *If $t \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{nf}_\vartheta$ with $\mathtt{nf}_\vartheta \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ then $t \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ and there is u such that $t \twoheadrightarrow_{\mathtt{gc}} u$ and $\mathtt{nf}_\vartheta \twoheadrightarrow_{\mathtt{gc}} u$.*

Diagrammatically, see Fig. 4.2.

Proof. We show that the following conditions are equivalent for any term t such that $\mathtt{fv}(t) \subseteq \vartheta$. They imply items (1) and (2) of this lemma: (i) t is a $\overset{\vartheta}{\rightsquigarrow}$-normal form, (ii) $\downarrow_{\mathtt{gc}}(t)$ is a $\to_{\mathtt{sh}}$-normal form, (iii) $t \equiv_{\neg\vartheta} s$ for some $s \in \mathtt{NF}(\to_{\mathtt{sh}})$, (iv) $t \equiv_{\neg\vartheta} s$ for some $s \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$. Here $\equiv_{\neg\vartheta}$ stands for the least equivalence relation containing $\to_{\mathtt{gc}} \cup \xrightarrow{\neg\vartheta}_{\mathtt{sh}}$. □

As an example of this lemma, consider the sequence $x[y\backslash z[z\backslash I]] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} x[y\backslash I[z\backslash I]] \to_{\mathtt{gc}} x$. All three terms are in $\mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$: this is straightforward for x, and due to the fact that the substitution is garbage for the two others. Moreover, although we do not have $x[y\backslash z[z\backslash I]] \to_{\mathtt{gc}} x[y\backslash I[z\backslash I]]$, both terms reduce in one gc-step to the same term x.

The results in this section can now be assembled to complete the argument outlined in Fig. 4.2 to prove Fig. 4.1(c):

Proposition 4.54 (Factorization of the Theory of Sharing)**.** *Let $\vartheta = \mathtt{fv}(t)$. If $t \twoheadrightarrow_{\mathtt{sh}} s \in \mathtt{NF}(\to_{\mathtt{sh}})$, then there exists a term $u \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ such that $t \overset{\vartheta}{\rightsquigarrow}{}^* u$ and $u^\diamond = s^\diamond$. (More precisely, $u \twoheadrightarrow_{\mathtt{gc}} s$).*

Proof. Combining postponement of gc (Lem. 4.46), the external–internal factorization result (Prop. 4.51), and Lem. 4.53. □

Finally, we obtain the full completeness theorem of Fig. 4.1(a):

Theorem 4.55 (Completeness of $\overset{\vartheta}{\rightsquigarrow}$ with respect to β-reduction)**.** *Let $\vartheta = \mathtt{fv}(t)$. If $t \twoheadrightarrow_\beta s \in \mathtt{NF}(\to_\beta)$ then there exists a term $u \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ such that $t \overset{\vartheta}{\rightsquigarrow}{}^* u$ and $u^\diamond = s$.*

Proof. Immediate, combining Prop. 4.45 and Prop. 4.54 as described in Fig. 4.1. □

Chapter 5

Strong Call-by-Need for Pattern Matching and Fixed Points

5.1 Introduction

This chapter is devoted to generalizing the strong call-by-need strategy of the preceding chapter (Chapter 4) to the **Extended λ-Calculus** of Grégoire and Leroy [67], and which they call the "type-erased λ-calculus". The extended λ-calculus, denoted λ^{e}, extends the lambda calculus with constants, pattern matching and fixed-points.

Here is an example of a term in λ^{e} that computes the length of a list encoded with constants **nil** and **cons**:

$$\texttt{fix}\,(l.\ \lambda xs.\ \texttt{case}\ xs\ \texttt{of}\ (\mathbf{nil} \Rightarrow \mathbf{zero}) \cdot (\mathbf{cons}\ hd\ tl \Rightarrow \mathbf{succ}\,(l\ tl)))$$

The Extended Lambda Calculus is a subset of Gallina, the specification language of the Coq proof assistant. Grégoire and Leroy [67] study mechanisms for implementing strong reduction in λ^{e} in order to apply it to check type conversion. They propose a notion of strong reduction for λ^{e} on open terms, *i.e.* terms possibly containing free variables, called *symbolic call-by-value*. Symbolic call-by-value iterates call-by-value, accumulating terms for which computation cannot progress. No notion of sharing is addressed. Indeed, unnecessary computation may be performed. For example, consider the following λ^{e} term, where I abbreviates the identity term $\lambda z.z$:

$$\texttt{case}\ \mathbf{c}\,(I\ I)\ \texttt{of}\ \mathbf{c}\,x \Rightarrow \mathbf{d} \tag{5.1}$$

This term is a case expression that has *condition* $\mathbf{c}\,(I\ I)$ and *branch* $\mathbf{c}\,x \Rightarrow \mathbf{d}$, the pattern of the branch being $\mathbf{c}\,x$ and the target $\mathbf{d}$. Notice that the branch does not make use of x in the target. However, symbolic call-by-value contracts the redex $I\ I$ since the argument of $\mathbf{c}$ must be a value before selecting the matching branch.

In this chapter, we propose a strong call-by-need strategy that generalizes the strong call-by-need strategy of the previous chapter to the setting of the extended λ-calculus. Informally:

$$\frac{\text{strong call-by-need (Chapter 4)}}{\lambda\text{-calculus}} :: \frac{\text{extended strong call-by-need (this chapter)}}{\text{extended } \lambda\text{-calculus ([67])}}$$

The development of the extended strong call-by-need strategy is split into three steps:

The Extended Theory of Sharing $\to^{\mathsf{e}}_{\mathtt{sh}}$. The first step in this chapter is introducing the Extended Theory of Sharing $\lambda^{\mathsf{e}}_{\mathtt{sh}}$. This theory generalizes Def. 4.4 to deal with `case` constructs and fixed points.

The Extended Non-Idempotent Intersection Type System $\mathcal{HW}^{\mathsf{e}}$. The second step in this chapter is to adapt the non-idempotent intersection type system $\mathcal{HW}$ to an extended non-idempotent intersection type system $\mathcal{HW}^{\mathsf{e}}$ that characterizes weakly normalizing terms in the extended theory of sharing $\to^{\mathsf{e}}_{\mathtt{sh}}$.

It turns out that it is relatively easy to extend $\mathcal{HW}$ to deal with fixed points. The challenge lies in dealing with `case` constructs. For example, consider the term:

$$\mathtt{case}\ \mathbf{c}\ \mathtt{of}\ (\mathbf{c} \Rightarrow \mathbf{d}) \cdot (\mathbf{d} \Rightarrow \Omega)$$

It will evaluate to $\mathbf{d}$ and hence should be typable in the extended non-idempotent intersection type system $\mathcal{HW}^{\mathsf{e}}$. Since Ω does not participate at all in computing $\mathbf{d}$, there is no need for $\mathcal{HW}^{\mathsf{e}}$ to account for it. Thus our proposed typing rules will only type branches that are actually used to compute the normal form. This, however, raises the question of what happens with case expressions that are "blocked". For example, in an expression such as:

$$\mathtt{case}\ \mathbf{c}\ \mathtt{of}\ (\mathbf{d} \Rightarrow \mathbf{d}) \cdot (\mathbf{e} \Rightarrow \mathbf{e})$$

all the subexpressions are part of the normal form and hence should be typed. Our proposed typing rule shall ensure this, thus avoiding typing terms such as:

$$\mathtt{case}\ \mathbf{c}\ \mathtt{of}\ (\mathbf{d} \Rightarrow \mathbf{d}) \cdot (\mathbf{e} \Rightarrow \Omega)$$

where, although matching is blocked, have no strong normal form in λ^{e} or $\lambda^{\mathsf{e}}_{\mathtt{sh}}$. Since blocked case expressions could be applied to arguments, further considerations are required. Consider the term:

$$(\mathtt{case}\ \mathbf{c}\ \mathtt{of}\ \mathbf{d} \Rightarrow \mathbf{d})\,\Omega$$

It does not have a normal form in λ^{e} or $\lambda^{\mathsf{e}}_{\mathtt{sh}}$ and hence should not be typable. To ensure that, we need the type assigned to this term to provide access to the types of the arguments to which it is applied, namely Ω, so that constraints on these types may be placed. In other words, we need to devise $\mathcal{HW}^{\mathsf{e}}$ such that it gives $\mathtt{case}\ \mathbf{c}\ \mathtt{of}\ (\mathbf{d} \Rightarrow \mathbf{d})\,\Omega$ a type that *includes* that of Ω. This would enable us to state conditions that do not allow this term to be typed but do allow a term such as $(\mathtt{case}\ \mathbf{c}\ \mathtt{of}\ \mathbf{d} \Rightarrow \mathbf{d})\,\mathbf{e}$ to be typed. This motivates our notions of *error type* and *error log*.

The above examples were all closed terms. Open terms pose additional problems. Consider the term:

$$\mathtt{case}\ x\ \mathtt{of}\ (\mathbf{c} \Rightarrow \mathbf{d}) \cdot (\mathbf{e} \Rightarrow \Omega)$$

Although it does not have a normal form in $\lambda^{\mathsf{e}}_{\mathtt{sh}}$, it is typable with type $\mathbf{d}$ in the typing context in which $x : [\mathbf{c}]$. Note, moreover, that the empty multiset of types does not occur in the type of x (in fact, it meets all the requirements of [87]). The reason it is typable is that Ω is never accounted: since x is known to have type $[\mathbf{c}]$, only the $\mathbf{c} \Rightarrow \mathbf{d}$ branch is typed. Hence some restrictions on the types of free variables must be put forward—variables cannot be assigned any type. In particular, it seems we should not allow constant types such as $\mathbf{c}$ to occur *positively* in the types of free variables. Indeed, we will require that constant types do not occur *positively* in the typing context and *negatively* in error logs and in the predicate. Note that constants can occur *negatively* in the types of variables. This allows terms such as $x\,\mathbf{c}$ to be typable.

One final consideration is that collecting all the requirements, both on empty multiset types and type constants, should still allow weakly normalizing terms in λ^{e} to be typable in $\mathcal{HW}^{\mathsf{e}}$. We will see that this will indeed be the case.

As a closely related work, we should mention that in his PhD thesis [25], Bernadet proposes a non-idempotent intersection type system for a calculus similar to the extended λ-calculus, which includes fixed-points and case expressions. However, his goal is to characterize a *subset* of the *strongly* normalizing terms, while, in order to prove completeness of the strong call-by-need strategy, we need to characterize *all* of the *weakly* normalizing terms.

The Extended Strong Call-by-Need Strategy $\rightsquigarrow^{\mathsf{e}}$. As mentioned, reduction in the theory of sharing may involve reducing redexes that are not needed. By restricting reduction in $\rightarrow^{\mathsf{e}}_{\mathtt{sh}}$ to a subset of the contexts where reduction can take place, we can ensure that only needed redexes are reduced. We next illustrate, through an example, our call-by-need strategy. The strategy will be denoted $\rightsquigarrow^{\mathsf{e}}$. Consider the term:

$$(\mathtt{case}\ (\lambda y.x\,y)(I\ I)\ \mathtt{of}\ \mathbf{c} \Rightarrow \mathbf{d})\ (I\,\mathbf{c})$$

It consists of a case expression applied to an argument. This case expression has a *condition* $(\lambda y.x\,y)(I\ I)$, a *branch* $\mathbf{c} \Rightarrow \mathbf{d}$ with *pattern* $\mathbf{c}$ and *target* $\mathbf{d}$, and is applied to an argument $I\,\mathbf{c}$. The first reduction step for this term is the same as for weak call-by-need, namely reducing the β-redex $(\lambda y.x\,y)(I\ I)$ in the condition of the case. It must be reduced in order to determine which branch, if any, is to be selected. This β-redex is turned into $(x\,y)[y\backslash I\ I]$. The resulting term is:

$$(\mathtt{case}\ (x\,y)[y\backslash I\ I]\ \mathtt{of}\ \mathbf{c} \Rightarrow \mathbf{d})(I\,\mathbf{c})$$

A weak call-by-need strategy would stop there, since the case expression is stuck. In the strong case, however, reduction should continue to complete the evaluation of the term until a strong normal form is reached. Both the body of the explicit substitution $I\ I$ and also the argument of the stuck case expression $I\,\mathbf{c}$ are needed to produce the strong normal form. Thus evaluation must continue with these redexes. That these redexes are indeed selected and, moreover, which one is selected first, depends on an appropriate notion of *evaluation context*. Our strategy will include an evaluation context $\mathtt{C}$ of the form $(\mathtt{case}\ (x\,y)[y\backslash\Box]\ \mathtt{of}\ \mathbf{c} \Rightarrow \mathbf{d})(I\,\mathbf{c})$ and hence the body of the explicit substitution will be reduced next. Notice that in order for the focus of computation to be placed in the body of an explicit substitution, its target y should be needed. In this particular case, it is because x is free but y is needed for computing

the strong normal form. However, in a term such $\lambda x.\mathbf{c}[y\backslash I\ I]$, the β-redex $I\ I$ is not needed for the strong normal form and hence will not be selected by the strategy.

The remaining computation steps leading to the strong normal form are depicted below.

$$
\begin{array}{rll}
 & (\texttt{case}\ (\lambda y.x\ y)(I\ I)\ \texttt{of}\ \mathbf{c} \Rightarrow \mathbf{d})(I\ \mathbf{c}) & \\
\rightsquigarrow^{\mathsf{e}} & (\texttt{case}\ (x\ y)[y\backslash I\ I]\ \texttt{of}\ \mathbf{c} \Rightarrow \mathbf{d})(I\ \mathbf{c}) & \\
\rightsquigarrow^{\mathsf{e}} & (\texttt{case}\ (x\ y)[y\backslash z[z\backslash I]]\ \texttt{of}\ \mathbf{c} \Rightarrow \mathbf{d})(I\ \mathbf{c}) & \\
\rightsquigarrow^{\mathsf{e}} & (\texttt{case}\ (x\ y)[y\backslash I[z\backslash I]]\ \texttt{of}\ \mathbf{c} \Rightarrow \mathbf{d})(I\ \mathbf{c}) & \\
\rightsquigarrow^{\mathsf{e}} & (\texttt{case}\ (x\ I)[y\backslash I][z\backslash I]\ \texttt{of}\ \mathbf{c} \Rightarrow \mathbf{d})(I\ \mathbf{c}) & (*) \\
\rightsquigarrow^{\mathsf{e}} & (\texttt{case}\ (x\ I)[y\backslash I][z\backslash I]\ \texttt{of}\ \mathbf{c} \Rightarrow \mathbf{d})(z[z\backslash \mathbf{c}]) & \\
\rightsquigarrow^{\mathsf{e}} & (\texttt{case}\ (x\ I)[y\backslash I][z\backslash I]\ \texttt{of}\ \mathbf{c} \Rightarrow \mathbf{d})(\mathbf{c}[z\backslash \mathbf{c}]) &
\end{array}
$$

Note that in the fourth step (indicated with an asterisk), y has been replaced by I. As in weak call-by-need, only *answers* shall be substituted for variables. Answers are abstractions under a possibly empty list of explicit substitutions or data structures possibly interspersed with explicit substitutions. Finally, crucial to defining the strong call-by-need strategy will be identifying variables and case expressions that will persist. The former are referred to as *frozen* variables and are free variables (or those that are bound under abstractions and branches of case expressions) that we know will never be substituted by an answer. The latter are referred to as *error terms* and are case expressions that we know will be stuck forever. An example of the former is $x\ y$ in $(x\ y)[y\backslash I\ I]$; an example of the latter is $\texttt{case}\ (x\ I)[y\backslash I][z\backslash I]\ \texttt{of}\ \mathbf{c} \Rightarrow \mathbf{d}$ in $(\texttt{case}\ (x\ I)[y\backslash I][z\backslash I]\ \texttt{of}\ \mathbf{c} \Rightarrow \mathbf{d})(I\ \mathbf{c})$.

5.1.1 Our Work

This chapter is the result of collaboration with Eduardo Bonelli and Kareem Mohamed. Generally speaking, systems in this chapter are an extension of the ones in Chapter 4 to account for pattern matching and fixed points. As a result, there are more syntactic constructs, more inference rules, and more complex definitions, but essentially the proof techniques of the previous chapter are applied without radical changes. Most proofs have been omitted from this chapter.

This chapter is structured as follows. We highlight in boldface what we consider to be the main contributions:

- In Section 5.2, we recall the definition of Grégoire and Leroy's extended λ-calculus (Def. 5.3), we generalize the Theory of Sharing for the extended λ-calculus (Def. 5.7), and we provide a syntactic characterization of the normal forms (Def. 5.7).

- In Section 5.3, we propose a **non-idempotent intersection type system $\mathcal{HW}^{\mathsf{e}}$ for λ^{e}** (Def. 5.10), and we show that **weakly normalizing terms in λ^{e} are typable** (Thm. 5.13) and that **typable terms are weakly normalizing in $\lambda^{\mathsf{e}}_{\mathtt{sh}}$** (Thm. 5.14). More precisely, both theorems require not only that the term is typable, but also that the typing judgment is "good" in a precise sense (*cf.* Def. 5.12). This notion of goodness generalizes the usual condition that there are no positive occurrences of the empty multiset $[]$.

- In Section 5.4, we propose a **strong call-by-need strategy $\rightsquigarrow^{\mathsf{e}}$ for λ^{e}** (Def. 5.17), and we show that the strategy enjoys good properties. Namely, it is deterministic (Prop. 5.21), it conservatively extends the strong call-by-need strategy of the previous chapter (Prop. 5.21), it is correct (Prop. 5.22) and it is **complete with respect to reduction in the extended λ-calculus** (Thm. 5.23).

5.2 Extending the Theory of Sharing

In this section we extend the Theory of Sharing (*cf.* Def. 4.4) to the extended λ-calculus. In Section 5.2.1, we begin by recalling the definition of the extended λ-calculus of Grégoire and Leroy [67]. In Section 5.2.2 we give the actual definition of the Extended Theory of Sharing $\lambda^{\mathsf{e}}_{\mathtt{sh}}$.

5.2.1 The Extended λ-Calculus

Definition 5.1 (Syntax of the extended λ-calculus, *cf.* [67])**.** Assume given a denumerable set of variables $x, y, z, \ldots$ and constants $\mathbf{c}, \mathbf{c}', \mathbf{c}'', \ldots$. The set of *terms* $\mathcal{T}^{\mathsf{e}}$ of the extended λ-calculus are defined as follows, mutually inductively with the set of *branches* (branches of `case`-constructs):

$$\begin{array}{lrcl} \text{Terms} & t, s, u, \ldots & ::= & x \mid \lambda x.t \mid t\, s \mid \mathbf{c} \mid \mathtt{fix}(x.t) \mid \mathtt{case}\ t\ \mathtt{of}\ \bar{b} \\ \text{Branches} & b & ::= & \mathbf{c}\bar{x} \Rightarrow t \end{array}$$

Contexts are defined as expected.

In addition to the usual terms of the λ-calculus, the calculus has *constants*, *case expressions* and *fixed-point expressions*. In $\mathtt{case}\ t\ \mathtt{of}\ \bar{b}$ we say t is the *condition* of the case and $\bar{b}$ are its *branches*; $\bar{b}$ represents a possibly empty sequence of branches. If $I = \{1, 2, \ldots, n\}$, we sometimes write $(\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I}$ for a list of branches $(\mathbf{c}_1\bar{x}_1 \Rightarrow s_1) \ldots (\mathbf{c}_n\bar{x}_n \Rightarrow s_n)$. Branches are assumed to be syntactically restricted so that if $i \neq j$ then $(\mathbf{c}_i, |\bar{x}_i|) \neq (\mathbf{c}_j, |\bar{x}_j|)$, where $|\bar{x}_j|$ denotes the length of the sequence $\bar{x}_j$. Moreover, the list $\bar{x}_i$ of formal parameters in each branch is assumed to have no repeats.

The expression $\mathtt{fix}(x.t)$ is a fixed-point expression. We often write $\lambda\bar{x}.t$ for $\lambda x_1. \ldots \lambda x_n.t$ if $\bar{x}$ is the sequence of variables $x_1 \cdot \ldots \cdot x_n$ and similarly $t\bar{s}$ stands for $ts_1 \ldots s_n$ if $\bar{s} = s_1 \cdot \ldots \cdot s_n$. *Free* and *bound* variables are defined as expected. In particular, x is bound by a fixed point operator $\mathtt{fix}(x.t)$, and all the variables $x_1, \ldots, x_n$ are bound in a branch $\mathbf{c}x_1 \ldots x_n \Rightarrow t$.

Remark 5.2. In [67] a family of fixed-point operators fix_n, for n a positive integer, is used. The index n indicates the expected number of arguments and also the index of the argument that is used to guard recursion to avoid infinite unfoldings. The type system of the Calculus of Constructions guarantees that the recursive function is applied to strict subterms of the n-th argument. Although we use the more general fixed-point operator fix in our calculus similar ideas to "case" can be applied to fix_n which "blocks" if given less than n arguments.

Definition 5.3 (The extended λ-calculus, *cf.* [67])**.** The λ^{e}*-calculus* is given by the following reduction rules over $\mathcal{T}^{\mathsf{e}}$, closed by arbitrary contexts. We write $\to^{\mathsf{e}}$ for the resulting reduction relation.

$$\begin{array}{rlll} (\lambda x.t)s & \mapsto_{\mathtt{db}} & t\{x := s\} & (\beta) \\ \mathtt{fix}(x.t) & \mapsto_{\mathtt{fix}} & t\{x := \mathtt{fix}(x.t)\} & (\mathtt{fix}) \\ \mathtt{case}\ \mathbf{c}_j\bar{t}\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} & \mapsto_{\mathtt{case}} & s_j\{\bar{x}_j := \bar{t}\} & (\mathtt{case}) \\ & & \text{if } j \in I \text{ and } |\bar{t}| = |\bar{x}_j| & \end{array}$$

The *simultaneous* capture-avoiding substitution of a list of variables $\bar{x}$ by a list of terms $\bar{s}$ of the same length in a term t is written $t\{\bar{x} := \bar{s}\}$. A term t *matches* with a branch $\mathbf{c}\bar{x} \Rightarrow s$ if $t = \mathbf{c}\bar{s}$ with $|\bar{s}| = |\bar{x}|$. A term t matches with a list of branches if it matches with at least one branch. Given our syntactic formation condition on case-expressions, terms match with at most one branch. Note that term reduction may become blocked if the condition of a case does not match any branch (and never will). The normal forms of λ^{e} may be characterized as follows:

Lemma 5.4 (Normal forms)**.** *The normal forms of* λ^{e} *are characterized by the grammar:*

$$N \quad ::= \quad \lambda\bar{x}.x\bar{N} \mid \lambda\bar{x}.\mathbf{c}\bar{N} \mid \lambda\bar{x}.(\mathtt{case}\ N_0\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow N_i)_{i\in I})\bar{N}$$

where N_0 *does not match with* $(\mathbf{c}_i\bar{x}_i \Rightarrow N_i)_{i\in I}$*. Note that the lists* $\bar{x}$ *and* $\bar{N}$ *may be empty.*

Proof. By structural induction on the set of terms. □

5.2.2 The Extended Theory of Sharing

Definition 5.5 (Syntax of the Extended Theory of Sharing)**.** The *terms of the Extended Theory of Sharing* $\mathcal{T}^{\mathsf{e}}_{\mathtt{sh}}$ are defined as follows, extending the syntax of λ^{e} with explicit substitutions:

$$t, s, u, \ldots \quad ::= \quad x \mid \lambda x.t \mid t\,s \mid \mathtt{fix}(x.t) \mid \mathbf{c} \mid \mathtt{case}\ t\ \mathtt{of}\ \bar{b} \mid t[x\backslash s]$$

Recall that terms without explicit substitutions are called *pure terms*. A pure term $t^{\diamond}$ is obtained from any $t \in \mathcal{T}^{\mathsf{e}}_{\mathtt{sh}}$ by unfolding explicit substitutions, *e.g.* $((\mathtt{case}\ z\ \mathtt{of}\ \mathbf{c} \Rightarrow z)[z\backslash\mathbf{d}\,\mathbf{d}])^{\diamond} = \mathtt{case}\ \mathbf{d}\,\mathbf{d}\ \mathtt{of}\ \mathbf{c} \Rightarrow \mathbf{d}\,\mathbf{d}$.

In order to describe reduction in the Extended Theory of Sharing $\lambda^{\mathsf{e}}_{\mathtt{sh}}$, we need to introduce additional syntactic categories that generalize the notions of *answer* and *value* in presence of constructors:

Answers	a	$::=$	$\mathtt{vL}$
Values	$\mathtt{v}$	$::=$	$\lambda x.t \mid \mathtt{A}\langle\mathbf{c}\rangle$
Applicative contexts	$\mathtt{A}$	$::=$	$\Box \mid \mathtt{AL}\,t$
Substitution contexts	$\mathtt{L}$	$::=$	$\Box \mid \mathtt{L}[x\backslash t]$

An answer of the form $(\lambda x.t)\mathtt{L}$ is an *abstraction answer* and one of the form $\mathtt{A}\langle\mathbf{c}\rangle\mathtt{L}$ is an *applicative answer*. An example of the latter is $((\mathbf{c}\,x)[x\backslash y]\,\mathbf{d})[y\backslash s]$.

Definition 5.6 (Extended Theory of Sharing)**.** The *Extended Theory of Sharing* $\lambda^{\mathtt{e}}_{\mathtt{sh}}$ consists of the reduction rules over $\mathcal{T}^{\mathtt{e}}_{\mathtt{sh}}$ given below, closed by arbitrary contexts. We write $\to^{\mathtt{e}}_{\mathtt{sh}}$ for the reduction relation.

$$\begin{array}{rcll}
(\lambda x.t)\mathtt{L}\, s & \mapsto_{\mathtt{db}} & t[x\backslash s]\mathtt{L} & \\
\mathtt{C}\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{vL}] & \mapsto_{\mathtt{lsv}} & \mathtt{C}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\mathtt{L} & \\
t[x\backslash s] & \mapsto_{\mathtt{gc}} & t & \text{if } x \notin \mathsf{fv}(t) \\
\mathtt{fix}(x.t) & \mapsto_{\mathtt{fix}} & t[x\backslash \mathtt{fix}(x.t)] & \\
\mathtt{case}\ \mathtt{A}\langle \mathbf{c}_j\rangle\mathtt{L}\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} & \mapsto_{\mathtt{case}} & s_j[\bar{x}_j\backslash \mathtt{A}]\mathtt{L} & \text{if } |\mathtt{A}\langle\Box\rangle| = |\bar{x}_j| \text{ and } j \in I
\end{array}$$

The rules `db`, `lsv`, and `gc` are similar as in the (non-extended) Theory of Sharing (*cf.* Def. 4.4). The rules `fix` and `case` are similar to the corresponding rules in $\lambda^{\mathtt{e}}$, but using explicit substitutions. Note that the condition $\mathtt{A}\langle \mathbf{c}_j\rangle\mathtt{L}$ may have explicit substitutions interspersed. The *length* of an applicative context is defined as follows: $|\Box| \stackrel{\text{def}}{=} 0$ and $|\mathtt{AL}\, t| \stackrel{\text{def}}{=} 1 + |\mathtt{A}|$. Given a list of variables $\bar{x}$ and an applicative context `A` such that their lengths coincide, we define the substitution context $[\bar{x}\backslash\mathtt{A}]$ as follows: $[\epsilon\backslash\Box] \stackrel{\text{def}}{=} \Box$ and $[\bar{x}, y\backslash \mathtt{AL}\, t] \stackrel{\text{def}}{=} [\bar{x}\backslash\mathtt{A}]\mathtt{L}[y\backslash t]$. The reduct of $\mapsto_{\mathtt{case}}$ uses this notion to build an appropriate list of explicit substitutions for each parameter of the branch.

An inductive characterization of the $\to^{\mathtt{e}}_{\mathtt{sh}}$-normal forms is given in the following definition.

Definition 5.7 (Normal forms of $\lambda^{\mathtt{e}}_{\mathtt{sh}}$)**.** A term t *enables* a list of branches $(\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I}$, written $t \succ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I}$, if the term is of the form $t = \mathtt{A}\langle \mathbf{c}_j\rangle\mathtt{L}$, for some `A`, `L`, and $j \in I$ such that $|\mathtt{A}| = |\bar{x}_j|$. The judgment defining the set of *normal forms* ($t \in \mathcal{N}$) is defined simultaneously with four other judgments, namely *constant normal forms* ($t \in \mathcal{K}$), *structure normal forms* ($t \in \mathcal{S}$), *error normal forms* ($t \in \mathcal{E}$), and *abstraction normal forms* ($t \in \mathcal{L}$).

$$\frac{}{\mathbf{c} \in \mathcal{K}}\ \textsc{cNfCons} \qquad \frac{t \in \mathcal{K} \quad s \in \mathcal{N}}{t\, s \in \mathcal{K}}\ \textsc{cNfApp}$$

$$\frac{}{x \in \mathcal{S}}\ \textsc{sNfVar} \qquad \frac{t \in \mathcal{S} \quad s \in \mathcal{N}}{t\, s \in \mathcal{S}}\ \textsc{sNfApp}$$

$$\frac{t \in \mathcal{K} \cup \mathcal{L} \cup \mathcal{S} \quad t \not\succ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} \quad (s_i \in \mathcal{N})_{i\in I}}{\mathtt{case}\ t\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} \in \mathcal{E}}\ \textsc{eNfStrt}$$

$$\frac{t \in \mathcal{E} \quad s \in \mathcal{N}}{t\, s \in \mathcal{E}}\ \textsc{eNfApp} \qquad \frac{t \in \mathcal{E} \quad (s_i \in \mathcal{N})_{i\in I}}{\mathtt{case}\ t\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} \in \mathcal{E}}\ \textsc{eNfCase}$$

$$\frac{t \in \mathcal{N}}{\lambda x.t \in \mathcal{L}}\ \textsc{lNfLam} \qquad \frac{t \in \mathbb{X} \quad s \in \mathcal{S} \cup \mathcal{E} \quad x \in \mathsf{fv}(t)}{t[x\backslash s] \in \mathbb{X}}\ \textsc{nfSub}$$

$$\frac{t \in \mathcal{K}}{t \in \mathcal{N}}\ \textsc{nfCons} \quad \frac{t \in \mathcal{S}}{t \in \mathcal{N}}\ \textsc{nfStruct} \quad \frac{t \in \mathcal{E}}{t \in \mathcal{N}}\ \textsc{nfError} \quad \frac{t \in \mathcal{L}}{t \in \mathcal{N}}\ \textsc{nfLam}$$

Note that rule eNfStrt captures a blocked case where its condition is not a blocked case

itself. If the condition of the case is $t \in \mathcal{L} \cup \mathcal{S}$, then we know that it cannot possibly match any branch. If $t \in \mathcal{K}$, we must make sure of this, requiring that t does not enable the branches.

Lemma 5.8 (Characterization of normal forms in $\lambda^{\mathtt{e}}_{\mathtt{sh}}$)**.** *The following are equivalent:*

1. $t \in \mathcal{N}$
2. t *is in* $\to^{\mathtt{e}}_{\mathtt{sh}}$*-normal form.*

Proof. We omit the detailed proof. To show the implication $1 \implies 2$, one checks that if $t \in \mathcal{N} \cup \mathcal{K} \cup \mathcal{S} \cup \mathcal{E} \cup \mathcal{L}$ then t is in $\to^{\mathtt{e}}_{\mathtt{sh}}$-normal form, by induction on the derivation of the corresponding judgment. To show the implication $2 \implies 1$, proceed by induction on t. □

5.3 Extending the Type System

In this section we introduces $\mathcal{HW}^{\mathtt{e}}$, a non-idempotent intersection type system for the Extended Theory of Sharing $\lambda^{\mathtt{e}}_{\mathtt{sh}}$, and we argue that it characterizes normalization.

5.3.1 The Extended Non-Idempotent Intersection Type System

We assume $\alpha, \beta, \gamma, \ldots$ to range over a set of *type variables*. The set of *types* is ranged over by $\tau, \sigma, \rho, \ldots$, and *finite multisets of types* are ranged over by $\mathcal{M}, \mathcal{N}, \mathcal{P}, \ldots$. The empty multiset is written $[]$, and $[\tau_1, \ldots, \tau_n]$ stands for the multiset containing each of the types τ_i with their corresponding multiplicities. Moreover, $\mathcal{M} + \mathcal{N}$ stands for the (additive) union of multisets. For instance $[\mathbf{a}, \mathbf{b}] + [\mathbf{b}, \mathbf{c}] = [\mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{c}]$.

Definition 5.9 (Syntax of types)**.** The set of *types* of $\mathcal{HW}^{\mathtt{e}}$ is defined by the following grammar, mutually recursively with the sets of *datatypes*, *pre-error types*, *error types*, and *branch types*:

$$
\begin{array}{lrcl}
\text{Types} & \tau & ::= & \alpha \mid \mathcal{M} \to \tau \mid D \mid E \\
\text{Datatypes} & D & ::= & \mathbf{c} \mid D\,\mathcal{M} \\
\text{Pre-Error types} & G & ::= & \mathbb{E}\,\tau\,\bar{B} \mid G\,\tau \\
\text{Error types} & E & ::= & \langle G \rangle \mid E\,\tau \\
\text{Branch types} & B & ::= & \bar{\mathcal{M}} \Rightarrow \tau
\end{array}
$$

A *type* τ *matches* with a branch $\mathbf{c}\bar{x} \Rightarrow s$ if it is of the form $\tau = \mathbf{c}\bar{\mathcal{M}}$ with $|\bar{\mathcal{M}}| = |\bar{x}|$. A type matches with a list of branches if it matches with at least one branch.

The type α is a type variable, $\mathcal{M} \to \tau$ is a *function type*, D is a *datatype*, and E is an *error type*. A *datatype* is either a constant type $\mathbf{c}$ or an applied datatype $D\,\mathcal{M}$. Informally, $\mathbf{c}\mathcal{M}_1 \ldots \mathcal{M}_n$ is the type of a constant applied to n arguments, each of which has been assigned a multiset of types. PreError types are solely introduced for building error types; error types are used for typing `case` expressions which will eventually become *stuck*. A case is stuck if, intuitively, it can be decided that the condition cannot match any branch. An *error type* $\langle \mathbb{E}\,\tau\,(\bar{\mathcal{M}}_i \Rightarrow \sigma_i)_{i \in I} \rho_1 \ldots \rho_j \rangle\, \rho_{j+1} \ldots\ \rho_k$ is the type of a `case` expression such that:

1. its condition has type τ and its branches have type $\bar{\mathcal{M}}_i \Rightarrow \sigma_i$;
2. it is stuck;
3. it has been applied to arguments of type $\rho_1 \dots \rho_j$;
4. it is expecting arguments of type $\rho_{j+1} \dots \rho_k$.

We call $\mathbb{E}$ an *error type constructor*. Typing judgments involve two kinds of contexts:

1. On one hand, *typing contexts*, ranged over by $\Gamma, \Delta, \Theta, \dots$ are functions mapping variables to multisets of types, as in the system $\mathcal{HW}$ of [91], recalled in Chapter 4 (*cf.* Def. 4.26).
2. On the other hand, *error logs*, ranged over by $\Sigma, \Upsilon, \dots$ are sets of error types.

As in Chapter 4, we write $\Gamma + \Delta$ for the sum of typing contexts, and $\Gamma \oplus \Delta$ for their disjoint sum. Also, we write $\bar{x} : \bar{\mathcal{M}}$ for the context $((x_i)_{i\in I} : (\mathcal{M}_i)_{i\in I}) \stackrel{\text{def}}{=} \sum_{i\in I}(x_i : \mathcal{M}_i)$.

Definition 5.10 (The type system $\mathcal{HW}^{\mathsf{e}}$)**.** The typing system $\mathcal{HW}^{\mathsf{e}}$ is defined by means of the inductive typing rules below. These rules define the derivability for four forms of *typing judgments*, with the following informal interpretations:

1. **Typing** ($\Gamma; \Sigma \vdash t : \tau$) — The term t has type τ under the context Γ and the error log Σ.
2. **Multi-typing** ($\Gamma; \Sigma \vdash t : \mathcal{M}$) — The term t has the types in $\mathcal{M}$ under the context Γ and the error log Σ.
3. **Application** ($\tau \ @ \ \mathcal{M} \Rightarrow \sigma$) — A term of type τ may be applied to an argument that has all the types in $\mathcal{M}$, resulting in a term of type σ.
4. **Matching** ($\tau \ \langle \bar{b} \rangle \ \Gamma; \Sigma, \sigma$) — The type τ might be the condition of a case with branches $\bar{b}$, which will result in a term of type σ, assuming certain hypotheses Γ and error logs Σ, or else fail.

The rules of $\mathcal{HW}^{\mathsf{e}}$ are:

$$\frac{}{x : [\tau]; \Sigma \vdash x : \tau}\ \textsc{tVar} \qquad \frac{}{\varnothing; \Sigma \vdash \mathbf{c} : \mathbf{c}}\ \textsc{tCons}$$

$$\frac{\Gamma, x : \mathcal{M}; \Sigma \vdash t : \tau}{\Gamma; \Sigma \vdash \lambda x.t : \mathcal{M} \to \tau}\ \textsc{tAbs} \qquad \frac{\Gamma; \Sigma \vdash t : \tau \quad \tau \ @ \ \mathcal{M} \Rightarrow \sigma \quad \Delta; \Sigma \vdash s : \mathcal{M}}{\Gamma + \Delta; \Sigma \vdash ts : \sigma}\ \textsc{tApp}$$

$$\frac{\Gamma, x : \mathcal{M}; \Sigma \vdash t : \tau \quad \Delta; \Sigma \vdash \mathtt{fix}(x.t) : \mathcal{M}}{\Gamma + \Delta; \Sigma \vdash \mathtt{fix}(x.t) : \tau}\ \textsc{tFix} \qquad \frac{\Gamma; \Sigma \vdash t : \tau \quad \tau \ \langle \bar{b} \rangle \ \Delta; \Sigma, \sigma}{\Gamma + \Delta; \Sigma \vdash \mathtt{case}\ t\ \mathtt{of}\ \bar{b} : \sigma}\ \textsc{tCase}$$

$$\frac{\Gamma, x : \mathcal{M}; \Sigma \vdash t : \tau \quad \Delta; \Sigma \vdash s : \mathcal{M}}{\Gamma + \Delta; \Sigma \vdash t[x \backslash s] : \tau}\ \textsc{tES} \qquad \frac{(\Gamma_i; \Sigma \vdash t : \tau_i)_{1 \leqslant i \leqslant n} \quad (n \geqslant 0)}{\sum_{i=1}^{n} \Gamma_i; \Sigma \vdash t : \sum_{i=1}^{n} [\tau_i]}\ \textsc{tMulti}$$

$$\frac{}{\mathcal{M} \to \tau \ @ \ \mathcal{M} \Rightarrow \tau}\ \textsc{tAppFun} \qquad \frac{}{D \ @ \ \mathcal{M} \Rightarrow D\mathcal{M}}\ \textsc{tAppData}$$

$$\frac{(n \geqslant 1)}{\langle G \rangle\, \tau_1 \ldots \tau_n \mathbin{@} [\tau_1] \Rightarrow \langle G\, \tau_1 \rangle\, \tau_2 \ldots \tau_n}\ \textsc{tAppErr}$$

$$\frac{\mathbf{c}_j \bar{\mathcal{M}} \text{ matches } (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \quad \Gamma, \bar{x}_j : \bar{\mathcal{M}}; \Sigma \vdash s_j : \sigma_j}{\mathbf{c}_j \bar{\mathcal{M}}\, \langle (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \rangle\, \Gamma; \Sigma, \sigma_j}\ \textsc{tCMatch}$$

$$\frac{\tau \text{ does not match } (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \quad \left(\Gamma_i, \bar{x}_i : \bar{\mathcal{M}}_i; \Sigma \vdash s_i : \sigma_i\right)_{i \in I}}{\tau\, \langle (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \rangle\, (\sum_{i \in I} \Gamma_i); \Sigma \cup \{\langle \mathbb{E}\, \tau\, (\bar{\mathcal{M}}_i \Rightarrow \sigma_i)_{i \in I} \rangle\, \bar{\rho}\}, \langle \mathbb{E}\, \tau\, (\bar{\mathcal{M}}_i \Rightarrow \sigma_i)_{i \in I} \rangle\, \bar{\rho}}\ \textsc{tCMismatch}$$

We write $\pi, \xi, \ldots$ for typing derivations and $\pi(\Gamma; \Sigma \vdash t : \tau)$ if π is a typing derivation of the judgment $\Gamma; \Sigma \vdash t : \tau$. The rules are *linear* with respect to the typing context in the sense that each assumption is used exactly once. The rules are, however, *cartesian* with respect to the error log, in the sense that each assumption may be used zero, one, or more times.[1] The rule tApp allows typing applications of functions of to arguments by means of the *application judgment* $\tau \mathbin{@} \mathcal{M} \Rightarrow \sigma$. The application judgment allows that the function be an abstraction, a data structure, or an error term. The restriction to a singleton type in the tAppErr rule is to enforce that the arguments of a stuck case be typable.

The tFix rule splits the resources so that they are distributed to be used for the outermost unfolding (Γ) and for the rest of the unfoldings (Δ). The tCase rule relies on the *matching judgment* $\tau\ \langle \bar{b} \rangle\ \Delta; \Sigma, \sigma$, which checks whether the type of the condition τ matches the list of branches. If τ matches with a branch, then that branch is typed (*cf.* tCMatch). On the other hand, if τ does not match any branch (*cf.* tCMismatch), then *all* branches have to be accounted for by the type system. Moreover, in that case, the type of the `case` expression is an error type of the form $\langle \mathbb{E}\, \tau\, (\bar{\mathcal{M}}_i \Rightarrow \sigma_i)_{i \in I} \rangle\, \bar{\rho}$, which is recorded in the error log. Note that $\bar{\rho} = \rho_1, \ldots, \rho_k$ are the types of the arguments to which the stuck case expression will be allowed to be applied to. Finally, tMulti allows a term to be typed with a multiset type. In this rule, if $n = 0$, then $\sum_{i=1}^{n} [\tau_i]$ denotes the empty multiset $[]$.

5.3.2 Characterization of Weakly Normalizing Terms

In Chapter 4, we related typability in the type system $\mathcal{HW}$ with weak normalization in the λ-calculus (Thm. 4.33) and weak normalization in the Theory of Sharing (Thm. 4.43). In this subsection, we state a similar result for the extended system $\mathcal{HW}^{\mathsf{e}}$, relating it with weak normalization in the extended λ-calculus, and weak normalization in the Extended Theory of Sharing. Recall that, in Chapter 4, the results related the property that a term is weakly normalizing with the property that it is typable in $\mathcal{HW}$ in such a way that the judgment is "good" in the sense that it has no positive occurrences of the empty multiset $[]$.

Below we start by defining an appropriate notion of "good" judgment for $\mathcal{HW}^{\mathsf{e}}$ (Def. 5.12). Roughly speaking, a judgment is good if it has no positive occurrences of $[]$ and no negative occurrences of constructors. The reason to reject negative occurrences of constructors is illustrated by a term like $\texttt{case}\ x\ \texttt{of}\ (\mathbf{c} \Rightarrow \mathbf{d}) \cdot (\mathbf{e} \Rightarrow \Omega)$. This term is typable with type $\mathbf{d}$ if one

[1] Note that rules are *multiplicative* for typing contexts and *additive* for error logs.

assumes that $x : [\mathbf{c}]$. However, it is not weakly normalizing in $\lambda^{\mathtt{e}}_{\mathtt{sh}}$. Note that a free variable of type $\mathbf{c}$ corresponds to a negative occurrence of the constructor $\mathbf{c}$ (in the typing context).

However forbidding positive occurrences of $[]$ and negative occurrences of constructors alone does not suffice. The reason is the presence of blocked `case` expressions. Consider for example the term $(\mathtt{case}\ \mathbf{c}\ \mathtt{of}\ (\mathbf{d} \Rightarrow \mathbf{d}))\,\Omega$. This term is typable; for example, it may be assigned the type $\langle \mathbb{E}\,\mathbf{c}\,([\mathbf{d}] \Rightarrow \mathbf{d})\,[]\rangle$. Note that the type of the blocked case includes the types of arguments to which it is applied—in this case the empty multiset type. Moreover this type is registered in the error log. This allows us to extend the constraints that $[]$ does not occur positively and constructors do not occur negatively to type blocked `case` expressions.

As a further remark, note that a term such as $\mathtt{case}\ x\ \mathtt{of}\ (\mathbf{c} \Rightarrow \mathbf{d}) \cdot (\mathbf{e} \Rightarrow \mathbf{d})$ is in normal form, so it should be typable. Indeed, it shall be typed it by assigning x an appropriate *error type*.

Definition 5.11 (Positive and negative occurrences of types)**.** The set of positive (resp. negative) types occurring in τ, denoted $\mathcal{P}(\tau)$ (resp. $\mathcal{N}(\tau)$), is defined as follows:

$$
\begin{array}{rcl@{\qquad}rcl}
\mathcal{P}(\alpha) & \stackrel{\text{def}}{=} & \{\alpha\} & \mathcal{N}(\alpha) & \stackrel{\text{def}}{=} & \varnothing \\
\mathcal{P}(\mathcal{M} \to \tau) & \stackrel{\text{def}}{=} & \mathcal{N}(\mathcal{M}) \cup \mathcal{P}(\tau) \cup \{\mathcal{M} \to \tau\} & \mathcal{N}(\mathcal{M} \to \tau) & \stackrel{\text{def}}{=} & \mathcal{P}(\mathcal{M}) \cup \mathcal{N}(\tau) \\
\mathcal{P}(\mathbf{c}) & \stackrel{\text{def}}{=} & \{\mathbf{c}\} & \mathcal{N}(\mathbf{c}) & \stackrel{\text{def}}{=} & \varnothing \\
\mathcal{P}(D\,\mathcal{M}) & \stackrel{\text{def}}{=} & \mathcal{P}(D) \cup \mathcal{P}(\mathcal{M}) \cup \{D\,\mathcal{M}\} & \mathcal{N}(D\,\mathcal{M}) & \stackrel{\text{def}}{=} & \mathcal{N}(D) \cup \mathcal{N}(\mathcal{M}) \\
\mathcal{P}(E\,\tau) & \stackrel{\text{def}}{=} & \mathcal{P}(E) \cup \mathcal{P}(\tau) \cup \{E\,\tau\} & \mathcal{N}(E\,\tau) & \stackrel{\text{def}}{=} & \mathcal{N}(E) \cup \mathcal{N}(\tau) \\
\mathcal{P}(\langle G\rangle) & \stackrel{\text{def}}{=} & \mathcal{P}(G) \cup \{G\} & \mathcal{N}(\langle G\rangle) & \stackrel{\text{def}}{=} & \mathcal{N}(G) \\
\mathcal{P}(G\,\tau) & \stackrel{\text{def}}{=} & \mathcal{P}(G) \cup \mathcal{P}(\tau) \cup \{G\,\tau\} & \mathcal{N}(G\,\tau) & \stackrel{\text{def}}{=} & \mathcal{N}(G) \cup \mathcal{N}(\tau) \\
\mathcal{P}(\mathbb{E}\,\tau\,\bar{B}) & \stackrel{\text{def}}{=} & \mathcal{P}(\tau) \cup \mathcal{P}(\bar{B}) \cup \{\mathbb{E}\,\tau\,\bar{B}\} & \mathcal{N}(\mathbb{E}\,\tau\,\bar{B}) & \stackrel{\text{def}}{=} & \mathcal{N}(\tau)\ \cup \mathcal{N}(\bar{B}) \\
\mathcal{P}(\mathcal{M}_1, \ldots, \mathcal{M}_n \Rightarrow \tau) & \stackrel{\text{def}}{=} & \bigcup_{i\in 1..n} \mathcal{N}(\mathcal{M}_i) \cup \mathcal{P}(\tau) \cup \{\bar{\mathcal{M}} \Rightarrow \tau\} & \mathcal{N}(\mathcal{M}_1, \ldots, \mathcal{M}_n \Rightarrow \tau) & \stackrel{\text{def}}{=} & \bigcup_{i\in 1..n} \mathcal{P}(\mathcal{M}_i) \cup \mathcal{N}(\tau) \\
\mathcal{P}(\mathcal{M}) & \stackrel{\text{def}}{=} & \bigcup_{\tau\in\mathcal{M}} \mathcal{P}(\tau) \cup \{\mathcal{M}\} & \mathcal{N}(\mathcal{M}) & \stackrel{\text{def}}{=} & \bigcup_{\tau\in\mathcal{M}} \mathcal{N}(\tau) \\
\mathcal{P}(\Gamma; \Sigma \vdash \tau) & \stackrel{\text{def}}{=} & \mathcal{N}(\Gamma) \cup \mathcal{P}(\Sigma) \cup \mathcal{P}(\tau) & \mathcal{N}(\Gamma; \Sigma \vdash \tau) & \stackrel{\text{def}}{=} & \mathcal{P}(\Gamma) \cup \mathcal{N}(\Sigma) \cup \mathcal{N}(\tau) \\
\mathcal{P}(\Gamma) & \stackrel{\text{def}}{=} & \bigcup_{(x\in\mathsf{dom}\Gamma)} \mathcal{P}(\Gamma(x)) & \mathcal{N}(\Gamma) & \stackrel{\text{def}}{=} & \bigcup_{(x\in\mathsf{dom}\Gamma)} \mathcal{N}(\Gamma(x))
\end{array}
$$

Moreover, let $\mathtt{X}$ be a type (resp. datatype, pre-error type, error type, branch type, typing context). Then we say that $\mathtt{X}$ is *covered* by an error log Σ, written $\mathrm{covered}_\Sigma(\mathtt{X})$, if for every error type E such that E is a subformula of $\mathtt{X}$, *i.e.* it occurs anywhere in the syntactic tree of $\mathtt{X}$, one has that $E \in \Sigma$.

Definition 5.12 (Good types and typing judgements)**.** A *type τ is good* if $\mathbf{c} \notin \mathcal{P}(\tau)$ and $[] \notin \mathcal{N}(\tau)$. We say $\mathcal{M}$ is good if each $\tau \in \mathcal{M}$ is good. A *typing context Γ is good* if it can be written as $\Gamma = \Gamma_g\Gamma_e$ in such a way that $\Gamma_g(x)$ is good for every $x \in \mathsf{dom}\Gamma_g$, and $\Gamma_e(x)$ is an error type for every $x \in \mathsf{dom}\Gamma_e$. A *typing judgement* $\Gamma; \Sigma \vdash t : \tau$ *is good* if all of the following hold:

1. Γ is good;
2. $[] \notin \mathcal{P}(\Sigma)$ and $[] \notin \mathcal{P}(\tau)$;
3. $\mathbf{c} \notin \mathcal{N}(\Sigma)$ and $\mathbf{c} \notin \mathcal{N}(\tau)$ for every constructor $\mathbf{c}$;
4. $\mathrm{covered}_\Sigma(\Gamma)$ and $\mathrm{covered}_\Sigma(\tau)$.

Below we state the two main results of this section, which relate typability and normalization. Note that:

- Thm. 5.13 relates normal forms in λ^{e} with typability in $\mathcal{HW}^{\mathsf{e}}$, extending Thm. 4.33 from the previous chapter (which relates normal forms in the λ-calculus with typability in $\mathcal{HW}$).
- Thm. 5.14 relates normal forms in $\lambda^{\mathsf{e}}_{\mathtt{sh}}$ with typability in $\mathcal{HW}^{\mathsf{e}}$, extending Thm. 4.43 from the previous chapter (which relates normal forms in the Theory of Sharing with typability in $\mathcal{HW}$).

Theorem 5.13 (Weakly normalizing terms in λ^{e} are typable)**.** *Let t be weakly normalizing in λ^{e}. Then there exist a context Γ, an error log Σ, a type τ such that $\Gamma; \Sigma \vdash t : \tau$ is derivable and good.*

Proof. We omit the detailed proof. The proof relies on the two following claims:

> **Normal forms are typable.** Let t be a $\rightarrow^{\mathsf{e}}$-normal form. Then there exist a context Γ, an error log Σ and a type τ such that $\Gamma; \Sigma \vdash t : \tau$ is derivable and good.
>
> **Subject expansion.** If $t \rightarrow^{\mathsf{e}} s$ and $\Gamma; \Sigma \vdash s : \tau$, then $\Gamma; \Sigma \vdash t : \tau$.

□

Theorem 5.14 (Typable terms are weakly normalizing in $\lambda^{\mathsf{e}}_{\mathtt{sh}}$)**.** *If $\Gamma; \Sigma \vdash t : \tau$ is derivable and good, then t is weakly normalizing in $\lambda^{\mathsf{e}}_{\mathtt{sh}}$.*

Proof. We omit the detailed proof. The proof requires adapting the notion of T-occurrence (*cf.* Def. 4.39) to $\mathcal{HW}^{\mathsf{e}}$ and it relies on the following claim:

> **Weighted subject reduction.** Let $\pi(\Gamma; \Sigma \vdash t : \tau)$. If $t \rightarrow^{\mathsf{e}}_{\mathtt{sh}} t'$, then there exists π' such that $\pi'(\Gamma; \Sigma \vdash t' : \tau)$. Moreover, if this step reduces a T-occurrence in π, then either:
>
> 1. $\mathtt{size}(\pi) > \mathtt{size}(\pi')$; or
> 2. $\mathtt{size}(\pi) = \mathtt{size}(\pi')$ and $\mathtt{fix}(\pi) > \mathtt{fix}(\pi')$
>
> where $\mathtt{size}(\pi)$ denotes the size of the derivation π, seen as a tree, and $\mathtt{fix}(\pi)$ denotes the number of nodes in the derivation π that are instances of the tFix rule.

□

5.4 Extending the Strong Call-by-Need Strategy

In this section, we extend the strong call-by-need strategy $\overset{\mathsf{S}}{\rightsquigarrow}$ for the Theory of Sharing from Chapter 4, to a strong call-by-need strategy $\rightsquigarrow^{\mathsf{e}}$ for the Extended Theory of Sharing. Moreover, we show that the strategy is complete with respect to the extended λ-calculus λ^{e}.

5.4.1 The Extended Strong Call-by-Need Strategy

Similarly as in the previous chapter, the extended strong call-by-need *strategy* $\overset{\vartheta}{\rightsquigarrow}_{\mathsf{e}}$ is a binary relation over the set of extended terms $\mathcal{T}^{\mathsf{e}}_{\mathtt{sh}}$, and it is parameterized over a set ϑ of frozen variables. Its reduction rules are an instance of the rewriting rules of the Extended Theory of Sharing (Def. 5.6), with two differences: (1) the garbage collection rule is absent, and (2) reduction is not closed under arbitrary contexts but under *evaluation contexts*. Exactly as we did in Section 4.2.2, in order to define the set of evaluation contexts, we start by defining (syntactically) the set of normal forms of the strategy, and next we describe evaluation contexts.

Definition 5.15 (Normal forms of the extended strong call-by-need strategy)**.** The set of *non-garbage variables* of a term t, denoted $\mathsf{ngv}(t)$ is defined as $\mathsf{fv}(\downarrow_{\mathtt{gc}}(t))$ where $\downarrow_{\mathtt{gc}}(t)$ is the gc-normal form of t.

For each set of variables ϑ, the sets of *constant normal forms* ($\mathcal{K}_\vartheta$), *structure normal forms* ($\mathcal{S}_\vartheta$), *error normal forms* ($\mathcal{E}_\vartheta$), *abstraction normal forms* ($\mathcal{L}_\vartheta$), and (plain) *normal forms* ($\mathcal{N}_\vartheta$) are defined, mutually inductively by the following judgments. In the rules for explicit substitutions, $\mathbb{X}_\vartheta$ stands for any of the sets $\mathcal{K}_\vartheta$, $\mathcal{S}_\vartheta$, $\mathcal{E}_\vartheta$, or $\mathcal{L}_\vartheta$:

$$\frac{}{\mathbf{c} \in \mathcal{K}_\vartheta}\ \textsc{cNfCons} \qquad \frac{t \in \mathcal{K}_\vartheta \quad s \in \mathcal{N}_\vartheta}{t\,s \in \mathcal{K}_\vartheta}\ \textsc{cNfApp} \qquad \frac{x \in \vartheta}{x \in \mathcal{S}_\vartheta}\ \textsc{sNfVar} \qquad \frac{t \in \mathcal{S}_\vartheta \quad s \in \mathcal{N}_\vartheta}{t\,s \in \mathcal{S}_\vartheta}\ \textsc{sNfApp}$$

$$\frac{t \in \mathcal{K}_\vartheta \cup \mathcal{L}_\vartheta \cup \mathcal{S}_\vartheta \quad t \not\succ (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \quad (s_i \in \mathcal{N}_{\vartheta \cup \bar{x}_i})_{i \in I}}{\mathtt{case}\ t\ \mathtt{of}\ (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \in \mathcal{E}_\vartheta}\ \textsc{eNfStrt}$$

$$\frac{t \in \mathcal{E}_\vartheta \quad s \in \mathcal{N}_\vartheta}{t\,s \in \mathcal{E}_\vartheta}\ \textsc{eNfApp} \qquad \frac{t \in \mathcal{E}_\vartheta \quad (s_i \in \mathcal{N}_{\vartheta \cup \bar{x}_i})_{i \in I}}{\mathtt{case}\ t\ \mathtt{of}\ (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \in \mathcal{E}_\vartheta}\ \textsc{eNfCase} \qquad \frac{t \in \mathcal{N}_{\vartheta \cup \{x\}}}{\lambda x.t \in \mathcal{L}_\vartheta}\ \textsc{lNfLam}$$

$$\frac{t \in \mathbb{X}_{\vartheta \cup \{x\}} \quad s \in \mathcal{S}_\vartheta \cup \mathcal{E}_\vartheta \quad x \in \mathsf{ngv}(t)}{t[x \backslash s] \in \mathbb{X}_\vartheta}\ \textsc{nfSubNG} \qquad \frac{t \in \mathbb{X}_\vartheta \quad x \notin \mathsf{ngv}(t)}{t[x \backslash s] \in \mathbb{X}_\vartheta}\ \textsc{nfSubG}$$

$$\frac{t \in \mathcal{K}_\vartheta}{t \in \mathcal{N}_\vartheta}\ \textsc{nfCons} \qquad \frac{t \in \mathcal{S}_\vartheta}{t \in \mathcal{N}_\vartheta}\ \textsc{nfStruct} \qquad \frac{t \in \mathcal{L}_\vartheta}{t \in \mathcal{N}_\vartheta}\ \textsc{nfLam} \qquad \frac{t \in \mathcal{E}_\vartheta}{t \in \mathcal{N}_\vartheta}\ \textsc{nfError}$$

The syntactic definition of normal forms given above is similar to the syntactic characterization of $\rightarrow^{\mathsf{e}}_{\mathtt{sh}}$-normal forms given in Def. 5.7, except that: (1) the set of frozen variables is explicitly tracked, (2) rule nfSub is refined into rules nfSubNG, and (3) a new rule nfSubG is added due to the absence of gc in $\overset{\vartheta}{\rightsquigarrow}_{\mathsf{e}}$.

Definition 5.16 (Extended evaluation contexts)**.** Judgments defining the sets of evaluation contexts are of the form $\mathtt{C} \in \mathsf{E}^h_\vartheta$ where $\mathtt{C}$ is an arbitrary context, ϑ is a set of variables, and h is a symbol called *discriminator* of the context. This symbol may be one of '$\circ$', 'λ' or any constructor $\mathbf{c}, \mathbf{d}, \ldots$ and its role is to discriminate the head constructor in the context. Note that evaluation context formation rules place requirements on discriminators. An *evaluation context* is a context $\mathtt{C}$ such that the evaluation context judgement $\mathtt{C} \in \mathsf{E}^h_\vartheta$ is derivable for some set of variables ϑ and some discriminator h, using the following rules:

$$\frac{}{\Box \in \mathsf{E}^\circ_\vartheta}\ \textsc{eBox}$$

$$\frac{\mathtt{C} \in \mathsf{E}^h_\vartheta \quad h \neq \lambda}{\mathtt{C}\, t \in \mathsf{E}^h_\vartheta}\ \textsc{eAppL} \qquad \frac{t \in \mathcal{S}_\vartheta \cup \mathcal{E}_\vartheta \quad \mathtt{C} \in \mathsf{E}^h_\vartheta}{t\, \mathtt{C} \in \mathsf{E}^\circ_\vartheta}\ \textsc{eAppRStruct} \qquad \frac{t \in \mathcal{K}_\vartheta \quad \mathtt{C} \in \mathsf{E}^h_\vartheta}{t\, \mathtt{C} \in \mathsf{E}^{\mathsf{hc}(t)}_\vartheta}\ \textsc{eAppRCons}$$

$$\frac{\mathtt{C} \in \mathsf{E}^h_\vartheta \quad t \notin \mathcal{S}_\vartheta \cup \mathcal{E}_\vartheta \quad x \notin \vartheta}{\mathtt{C}[x\backslash t] \in \mathsf{E}^h_\vartheta}\ \textsc{eSubsLNonStruct} \qquad \frac{\mathtt{C} \in \mathsf{E}^h_{\vartheta \cup \{x\}} \quad t \in \mathcal{S}_\vartheta \cup \mathcal{E}_\vartheta}{\mathtt{C}[x\backslash t] \in \mathsf{E}^h_\vartheta}\ \textsc{eSubsLStruct}$$

$$\frac{\mathtt{C}_1 \in \mathsf{E}^h_\vartheta \quad \mathtt{C}_2 \in \mathsf{E}^\circ_\vartheta}{\mathtt{C}_1\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{C}_2] \in \mathsf{E}^h_\vartheta}\ \textsc{eSubsR} \qquad \frac{\mathtt{C} \in \mathsf{E}^h_{\vartheta \cup \{x\}}}{\lambda x.\mathtt{C} \in \mathsf{E}^\lambda_\vartheta}\ \textsc{eLam}$$

$$\frac{\mathtt{C} \in \mathsf{E}^h_\vartheta \quad h \notin \{\mathbf{c}_i\}_{i \in I} \text{ or } h = \mathbf{c}_j \in \{\mathbf{c}_i\}_{i \in I} \text{ and } |\mathtt{C}\langle y \rangle| \neq |\bar{x_j}|}{\mathtt{case}\ \mathtt{C}\ \mathtt{of}\ (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \in \mathsf{E}^\circ_\vartheta}\ \textsc{eCase1}$$

$$\frac{t \in \mathcal{N}_\vartheta \quad t \not> (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \quad t_k \in \mathcal{N}_{\vartheta \cup \bar{x_k}} \text{ for all } k < j \quad \mathtt{C} \in \mathsf{E}^h_{\vartheta \cup \bar{x}_i}}{\mathtt{case}\ t\ \mathtt{of}\ (\mathbf{c}_1 \bar{x_1} \Rightarrow t_1) \ldots (\mathbf{c}_j \bar{x_j} \Rightarrow \mathtt{C}) \ldots (\mathbf{c}_n \bar{x_n} \Rightarrow t_n) \in \mathsf{E}^\circ_\vartheta}\ \textsc{eCase2}$$

The function $\mathsf{hc}(-)$ used in rule eAppRCons is defined as follows, by induction on the derivation that $t \in \mathcal{K}_\vartheta$:

$$\mathsf{hc}(\mathbf{c}) \stackrel{\text{def}}{=} \mathbf{c} \qquad \mathsf{hc}(t\, s) \stackrel{\text{def}}{=} \mathsf{hc}(t) \qquad \mathsf{hc}(t[x\backslash s]) \stackrel{\text{def}}{=} \mathsf{hc}(t)$$

Note in particular that $\mathsf{hc}(\mathtt{A}\langle \mathbf{c} \rangle \mathtt{L}) = \mathbf{c}$. The notation $|\mathtt{C}\langle y \rangle|$ used in rule eCase1 counts the number of arguments in the spine of the term $\mathtt{C}\langle y \rangle$, more precisely:

$$\begin{aligned} |x| &\stackrel{\text{def}}{=} 0 \\ |\mathbf{c}| &\stackrel{\text{def}}{=} 0 \\ |\lambda x.t| &\stackrel{\text{def}}{=} 0 \\ |t\, s| &\stackrel{\text{def}}{=} 1 + |t| \end{aligned} \qquad \begin{aligned} |\mathtt{fix}(x.t)| &\stackrel{\text{def}}{=} 0 \\ |t[x\backslash s]| &\stackrel{\text{def}}{=} |t| \\ |\mathtt{case}\ t\ \mathtt{of}\ \bar{b}| &\stackrel{\text{def}}{=} 0 \end{aligned}$$

Rule eBox states that any redex at the root is needed. Rule eApp-L allows reduction to take place to the left of an application; in that case $\mathtt{C}$ must not be an abstraction. This is achieved by requiring that $h \neq \lambda$ (*cf.* eLam and how all rules persist h). In this way, the discriminator generalizes the distinction between arbitrary and *inert* evaluation contexts of Def. 4.12. In particular, in the fragment without pattern matching and fixed points, the set of arbitrary evaluation contexts E_ϑ of Chapter 4 corresponds to $\mathsf{E}^\circ_\vartheta \cup \mathsf{E}^\lambda_\vartheta$. Note that the set of inert contexts is written $\mathsf{E}^\circ_\vartheta$ in both presentations.

Rule eAppRStruct allows reduction to take place to the right of an application when it is an argument of a term t that is a structure normal form or an error normal form. The '$\circ$' in $t\, \mathtt{C} \in \mathsf{E}^\circ_\vartheta$ reflects that t is not headed by a constant and that $t\, \mathtt{C}$ is not an abstraction. Rule eAppRCons is similar only that the discriminator is set to the head variable of t via $\mathsf{hc}(t)$ and it will be checked when deciding if reduction can take place in the condition of a case (*cf.* eCase1). Frozen variables play the same role as in the (unextended) strong call-by-need strategy of the previous chapter.

There is no rule for $\mathtt{fix}(x.t)$ since reduction must necessarily take place at the root in such a term. Regarding case expressions, in order for reduction to take place in the condition

we must ensure that reduction at the root is not possible (*cf.* eCase1). This is achieved by requiring that the discriminator either is not a constant listed in the branches ($h \notin \{\mathbf{c}_i\}_{i\in I}$) or that, if it is, then the number of expected arguments by the branch are not met ($|\mathtt{C}\langle y\rangle| \neq |\bar{x}_j|$).

For reduction to proceed in a branch j (*cf.* eCase2), the condition must be in normal form, each branch i with $i \in 1..j$ must be in normal form and the condition must not enable any branch ($t \not> (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I}$). Note that the bound variables in branch j, are added to the set of frozen variables. We now define the strategy itself.

Definition 5.17 (Extended strong call-by-need strategy)**.** The $\overset{\vartheta}{\rightsquigarrow}^{\mathsf{e}}$ strategy is defined by the following rules.

$$
\begin{array}{llrcll}
(\mathtt{db}) & & \mathtt{C}\langle(\lambda x.t)\mathtt{L}\, s\rangle & \overset{\vartheta}{\rightsquigarrow}^{\mathsf{e}} & \mathtt{C}\langle t[x\backslash s]\mathtt{L}\rangle & \text{if } \mathtt{C} \in \mathsf{E}^h_\vartheta \\
(\mathtt{lsv}) & & \mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{vL}]\rangle & \overset{\vartheta}{\rightsquigarrow}^{\mathsf{e}} & \mathtt{C}_1\langle \mathtt{C}_2\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\mathtt{L}\rangle & \text{if } \mathtt{C}_1\langle \mathtt{C}_2\langle\Box\rangle[x\backslash \mathtt{vL}]\rangle \in \mathsf{E}^h_\vartheta \\
(\mathtt{fix}) & & \mathtt{C}\langle \mathtt{fix}(x.t)\rangle & \overset{\vartheta}{\rightsquigarrow}^{\mathsf{e}} & \mathtt{C}\langle t[x\backslash \mathtt{fix}(x.t)]\rangle & \text{if } \mathtt{C} \in \mathsf{E}^h_\vartheta \\
(\mathtt{case}) & \mathtt{C}\langle \mathtt{case}\ \mathtt{A}\langle \mathbf{c}_j\rangle \mathtt{L}\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I}\rangle & \overset{\vartheta}{\rightsquigarrow}^{\mathsf{e}} & \mathtt{C}\langle s_j[\bar{x}_j\backslash \mathtt{A}]\mathtt{L}\rangle & \\
 & & & & & \text{if } \mathtt{C} \in \mathsf{E}^h_\vartheta \text{ with } j \in I \text{ and } |\mathtt{A}\langle\Box\rangle| = |\bar{x}_j|
\end{array}
$$

Note that the discriminator h in the conditions of all rules is existentially quantified.

Properties of the Extended Strong Call-by-Need Strategy

The extended strong call-by-need strategy has the following properties. Contrast them with the properties studied in the previous chapter (Section 4.2.3, Section 4.3).

Lemma 5.18 (Characterization of normal forms)**.** *Let t be any term. Then the following are equivalent:*

1. *t is in $\overset{\vartheta}{\rightsquigarrow}^{\mathsf{e}}$-normal form,*
2. *$t \in \mathcal{N}_\vartheta$.*

Proof. We omit the full proof. To show $1 \implies 2$, proceed by induction on t. To show $2 \implies 1$, prove the more general statement that if $t \in \mathcal{N}_\vartheta \cup \mathcal{K}_\vartheta \cup \mathcal{S}_\vartheta \cup \mathcal{E}_\vartheta \cup \mathcal{L}_\vartheta$ then $t \in \mathsf{NF}(\overset{\vartheta}{\rightsquigarrow}^{\mathsf{e}})$, by simultaneous induction on the derivation that $t \in \mathcal{N}_\vartheta$ (resp. $t \in \mathcal{K}_\vartheta$, $t \in \mathcal{S}_\vartheta$, $t \in \mathcal{E}_\vartheta$, $t \in \mathcal{L}_\vartheta$). □

Proposition 5.19 (Strong reduction)**.** *If t is in $\overset{\vartheta}{\rightsquigarrow}^{\mathsf{e}}$-normal form, then its unfolding $t^\diamond$ is in $\to^{\mathsf{e}}$-normal form.*

Proof. We omit the full proof, which goes by induction on t, using the characterization of normal forms of Lem. 5.18. □

Proposition 5.20 (Determinism)**.** *If $t \overset{\vartheta}{\rightsquigarrow}^{\mathsf{e}} s$ and $t \overset{\vartheta}{\rightsquigarrow}^{\mathsf{e}} u$ then $s = u$.*

Proof. We omit the detailed proof. It relies on the following claim:

> **Unique decomposition.** If $\mathtt{C}\langle r\rangle$ is a term, we say that r is an *anchor* if it is a db-redex, a fix-redex, a case-redex or a variable bound to an answer. If $\mathtt{C}_1[r_1] = \mathtt{C}_2[r_2]$, where $\mathtt{C}_1, \mathtt{C}_2 \in \mathsf{E}^h_\vartheta$ and r_1, r_2 are anchors, then $\mathtt{C}_1 = \mathtt{C}_2$ and $r_1 = r_2$.

□

Proposition 5.21 (Conservativity)**.** *The extended strong call-by-need strategy is conservative with respect to the strong call-by-need strategy of Chapter 4,* i.e. *if* $t \overset{\vartheta}{\rightsquigarrow} s$ *then* $t \overset{\vartheta}{\rightsquigarrow}_{\mathsf{e}} s$.

Proof. The key point is that the notion of evaluation context (Def. 4.12) may be related with the notion of extended evaluation context (Def. 5.16). Indeed, it can be checked by induction on the derivation that if $\mathtt{C} \in \mathsf{E}_\vartheta$ then $\mathtt{C} \in \mathsf{E}^h_\vartheta$ for some discriminator $h \in \{\lambda, .\circ\}$ and that if $\mathtt{C} \in \mathsf{E}^\circ_\vartheta$ then $\mathtt{C} \in \mathsf{E}^h_\vartheta$ with $h = \circ$. □

Proposition 5.22 (Correctness)**.** *If* $t \overset{\vartheta}{\rightsquigarrow} s$ *then* $t^\diamond \twoheadrightarrow_{\mathsf{e}} s^\diamond$.

Proof. We omit the detailed proof, which goes by induction on t. □

Theorem 5.23 (Completeness)**.** *Let* $\vartheta = \mathsf{fv}(t)$*. If* $t \twoheadrightarrow_{\mathsf{e}} s \in \mathtt{NF}(\to_{\mathsf{e}})$*, then there exists a term* $u \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow}_{\mathsf{e}})$ *such that* $t \ (\overset{\vartheta}{\rightsquigarrow}_{\mathsf{e}})^* \ u$ *and* $u^\diamond = s$.

Proof. We omit the detailed proof. The argument follows the same lines as the completeness theorem of the previous chapter (Thm. 4.55), in particular relying on the fact that $\to_{\mathsf{e}}$-normalization implies typability in $\mathcal{HW}_{\mathsf{e}}$ (Thm. 5.13), the fact that typability in $\mathcal{HW}_{\mathsf{e}}$ implies $\to_{\mathtt{sh}_{\mathsf{e}}}$-normalization (Thm. 5.14). □

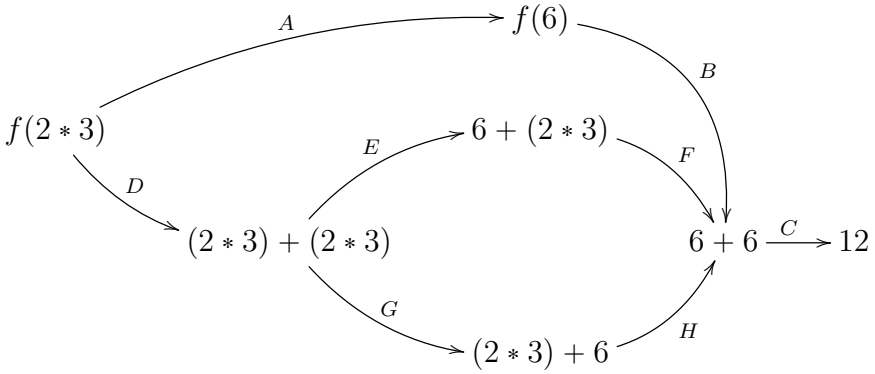
Chapter 6

A Labeled Linear Substitution Calculus

6.1 Introduction

6.1.1 Optimality and Redex Families

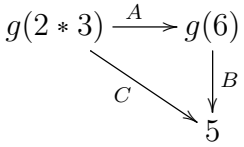
Consider the function $f(x) = x + x$. In general there may be many possible ways to rewrite an arithmetic expression in order to calculate its final result. If, for instance, one starts from the expression $f(2 * 3)$, there are three possible ways to calculate its value: the rewriting sequences ABC, $DEFC$, and $DGHC$ in the diagram below.



The rewriting sequences that start with D, namely $DEFC$ and $DGHC$, follow a *call-by-name* convention for parameter-passing, and they both require four computation steps. In contrast, the rewriting sequence at the top, ABC, follows a *call-by-value* convention for parameter-passing, and requires only three computation steps. The sequences that start with D are more computationally onerous than ABC: the reason is that the step D duplicates the subexpression $2 * 3$, which in turn calls for the duplication of the computational work required to calculate it.

One may wonder if consistently following a call-by-value convention always turns out to have the lowest computational cost, compared to other evaluation mechanisms. It is not too difficult to convince oneself that this is not the case. Consider, as a dual example, the constant

function $g(x) = 5$. There are two possible ways to rewrite $g(2 * 3)$ to calculate its final result:



The rewriting sequence AB follows a call-by-value convention, and yet it is clear that it is not "optimal", since it performs some unnecessary work: indeed, the first step A calculates the result of the subexpression $2 * 3$, which is immediately discarded. In this example, it is following a call-by-name convention, rather than a call-by-value convention, which realizes the minimal cost.

The previous examples motivate some natural questions. Given a programming language, how can the final result of a computation be obtained with minimal computational cost, that is, in an "optimal" way? In what precise sense one can define an evaluation mechanism to be optimal? Does an optimal evaluation mechanism always exist? Can it be computed and efficiently implemented?

Questions regarding optimal evaluation were first studied in the 1970s. Vuillemin studied the problem of optimal evaluation in the framework of recursive program schemes [143, 144], proving that, under certain sequentiality conditions, expressions can be optimally evaluated by using a sharing mechanism. Staples studied optimal evaluation for combinatory logic [132].

A major step forward was taken by Lévy [109, 110] together with Berry [27], who studied this problem in the context of the λ-calculus. In particular, in his 1978 PhD thesis, Lévy gave sufficient conditions for an evaluation mechanism to be optimal, in an appropriate sense. For an evaluation mechanism to be optimal, it suffices that all the computation steps that belong to the same *redex family* (to be defined later) are shared by the implementation, *i.e.* that no computational work is performed twice. Moreover, only *needed* steps should be performed, *i.e.* the implementation should not engage in superfluous computation. It is worth noting that these conditions, especially the condition that computation steps in the same redex family are *shared*, are quite demanding, and an implementation meeting these requirements was elusive for some time. A data structure effectively implementing the necessary amount of sharing was first proposed by Lamping [98] more than a decade after Lévy's seminal work. Following, we summarize some of the important results regarding optimal reduction in the context of the λ-calculus.

According to the standard nomenclature, let us define a *strategy* in a rewriting system to be a function $S : \mathsf{Term} \to \mathsf{Term}$ such that $t \to S(t)$ for every term t not in normal form. Given a starting term t, a strategy S induces a rewriting sequence

$$t \to S(t) \to S(S(t)) \to \dots$$

stopping whenever it reaches a normal form, and possibly infinite. A strategy S can then be defined to be *optimal* if, given a normalizable term t, the rewriting sequence induced by S reaches the normal form in a minimal number of steps. It is immediate to observe that,

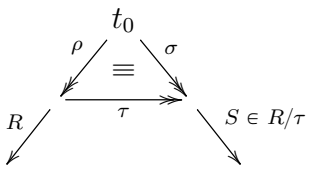
in a non-constructive sense, an optimal strategy *exists*, given that the length of a rewriting sequence is a natural number, and natural numbers are well-ordered. On the other hand, there is no hope of exhibiting an optimal strategy explicitly: Barendregt et al. [21] showed that, in the λ-calculus, no computable strategy is optimal.

This impossibility result would seem to defeat any attempt to devise a sensible notion of optimal reduction. However, one may conceive implementations that do not necessarily represent terms using a straightforward tree-like representation, but rather in some other form. For instance, terms may be represented as graphs, with pointers that allow sharing subterms, as in Wadsworth's lazy evaluation [145], or even sharing subterm "slices" (*i.e.* contexts) as in Lamping's sharing graphs [98]. An execution step in a sharing implementation can plausibly be simulated by the simultaneous contraction of many β-redexes, *i.e.* as a *multistep* $t \Rightarrow s$. Until now, we have been considering strategies in the λ-calculus. This reasoning leads us to consider strategies in the rewriting system of multisteps $\Lambda^{\mathfrak{M}}$ — a single step in $\Lambda^{\mathfrak{M}}$ is given by a multistep in the λ-calculus. Lévy's optimality result asserts that there are computable strategies in $\Lambda^{\mathfrak{M}}$ that reach the normal form, if it exists, with minimal cost. The contraction of a multistep $\mathcal{M} : t \Rightarrow s$ is considered to have unitary cost as long as all the steps in $\mathcal{M}$ belong to the same redex family. As already anticipated, a strategy can be shown to be optimal, according to Lévy's result, if it is a *family reduction*, *i.e.* each multistep is a maximal set of redexes that belong to the same family, and, moreover, each multistep contains at least one needed step. We now turn our attention to the notion of redex family, which plays a central role in the theory of optimal reductions.

Three Characterizations of Redex Families

A *redex family* is, intuitively, a set of computation steps that have a common origin, and whose calculation should be shared by an optimal implementation. In the λ-calculus, redex families were first defined by Lévy by giving three equivalent characterizations: *zig-zag*, *labels*, and *extraction*. Let us describe each of these characterizations.

Zig-zag. The first characterization of redex families is based on residual theory, and is abstract enough that it can be adapted to any other rewriting system admitting a sensible notion of residual, such as orthogonal term rewriting systems. Let t_0 be a fixed starting term. A *redex with history* from t_0 (*hredex* for short) is a derivation ρ starting from t_0, followed by a single step R. Equivalently, an hredex can be thought of as a non-empty derivation ρR, where $\rho : t_0 \twoheadrightarrow t_1$ is the *history* that has led us to t_1, and we are interested in the last step $R : t_1 \to t_2$. We say that an hredex σS is a *copy* of an hredex ρR, written $\rho R \leqslant \sigma S$, if there exists a derivation τ such that $\rho\tau \equiv \sigma$, *i.e.* $\rho\tau$ and σ are permutation equivalent (*cf.* Def. 2.40), and moreover $S \in R/\tau$. Graphically:



The *zig-zag relation* over redexes with history, written $\rho R \leftrightsquigarrow \sigma S$, is the least equivalence relation containing $\leqslant$. A *redex family* is an equivalence class of the relation $\leftrightsquigarrow$.

For example, consider the term $\Delta(FI)$ where $\Delta = \lambda x.xx$, $F = \lambda x.xz$, and $I = \lambda x.x$. Its reduction graph is depicted in Figure 6.1. We claim that the hredexes $S_1R_2T_3$ and $R_1S_3T_2$ are

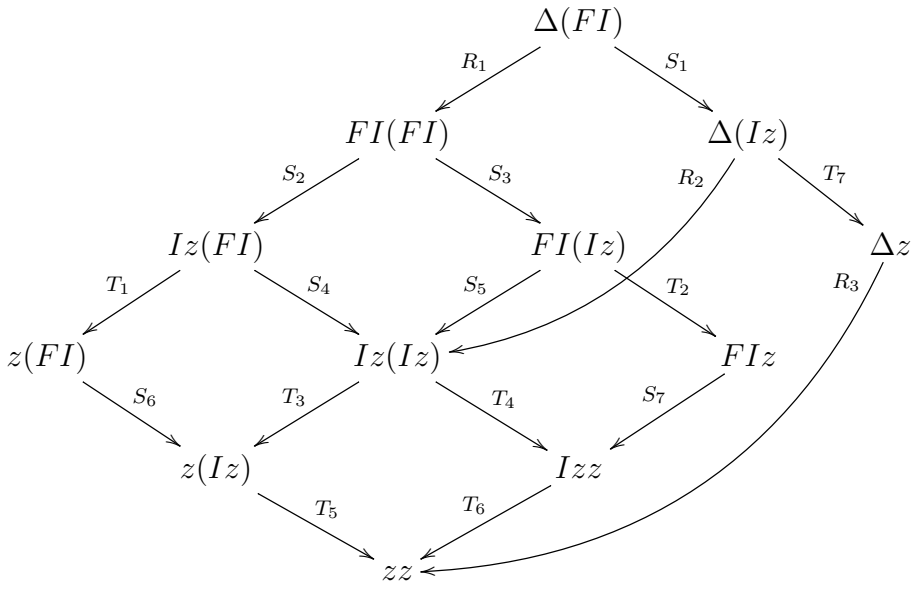


Figure 6.1: Reduction graph of $\Delta(FI)$, with $\Delta = \lambda x.xx$, $F = \lambda x.xz$, and $I = \lambda x.x$

in the same family, *i.e.* that $S_1R_2T_3 \leftrightsquigarrow R_1S_3T_2$. To see this, consider the hredex $R_1S_3S_5T_3$ and note that the following hold:

$S_1R_2T_3 \leqslant R_1S_3S_5T_3$ by taking the empty derivation and noting that $S_1R_2 \equiv R_1S_3S_5$
$R_1S_3T_2 \leqslant R_1S_3S_5T_3$ by noting that $T_3 \in T_2/S_5$

Labels. The second characterization of redex families is based on an auxiliary labeled variant of the λ-calculus, furnished with so-called *Lévy labels*. Consider a denumerable set of *initial labels* ranged over by $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$. The set of *labels* is defined by the following grammar:

$$\alpha, \beta, \gamma, \ldots ::= \mathbf{a} \mid \lceil\alpha\rceil \mid \lfloor\alpha\rfloor \mid \alpha\beta$$

Labels are considered up to associativity of label juxtaposition, *i.e.* for all labels α, β and γ the equality $(\alpha\beta)\gamma = \alpha(\beta\gamma)$ is declared to hold; n-way juxtaposition is usually written $\alpha_1\alpha_2\ldots\alpha_n$ for $n \geqslant 1$. The terms of the labeled λ-calculus are the *labeled terms*, defined by the grammar:

$$t, s, u, \ldots ::= x^\alpha \mid \lambda^\alpha x.t \mid @^\alpha(t, s)$$

Note that labels decorate each and every subterm of a term. An *initially labeled term* is a labeled term t such that the labels decorating its subterms are all initial and pairwise distinct; for instance $@^{\mathbf{a}}(\lambda^{\mathbf{b}}x.x^{\mathbf{c}}, y^{\mathbf{d}})$ is initially labeled. The idea behind labels is that they serve to trace the full history of a term. Each reduction step R in the labeled λ-calculus propagates

the labels in such a way as to leave a record that R has been contracted, making apparent the *contribution* of R to the ongoing computation. The operation of adding a label to a term, written $\alpha : t$ is defined by cases:

$$\begin{array}{rcl} \alpha : x^{\beta} & \stackrel{\text{def}}{=} & x^{\alpha\beta} \\ \alpha : \lambda^{\beta} x.t & \stackrel{\text{def}}{=} & \lambda^{\alpha\beta} x.t \\ \alpha : @^{\beta}(t,s) & \stackrel{\text{def}}{=} & @^{\alpha\beta}(t,s) \end{array}$$

Capture-avoiding substitution of a term for a variable $t\{x := s\}$ is defined as usual, except for the base case:

$$x^{\alpha}\{x := s\} \stackrel{\text{def}}{=} \alpha : s$$

For example, $@^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{c}})\{x := z^{\mathbf{d}}\} = @^{\mathbf{a}}(z^{\mathbf{bd}}, z^{\mathbf{cd}})$. Reduction in the labeled λ-calculus is defined as the closure by arbitrary contexts of the following labeled β-rule:

$$@^{\alpha}(\lambda^{\beta} x.t, s) \to \alpha\lceil\beta\rceil : t\{x := \lfloor\beta\rfloor : s\} \tag{6.1}$$

It is easy to prove that in general $\alpha : (t\{x := s\}) = (\alpha : t)\{x := s\}$, so the parenthesization of the right-hand side of the labeled β-rule is irrelevant. Each step in the labeled λ-calculus has a *name*. The name of a step like in (6.1) is the label β that decorates the abstraction. Some of the key properties of redex names are the following:

1. In an initially labeled term, different redexes have different names.
2. If R is an ancestor of R', then R and R' have the same name.
3. Whenever a redex R creates a redex S, the name of R is a sublabel of the name of S.

A term t^{ℓ} in the labeled λ-calculus is said to be a variant of a term t in the (unlabeled) λ-calculus if t results from t^{ℓ} by erasing all the labels. Given a step $R : t \to s$ in the λ-calculus and a labeled variant t^{ℓ} of t, the step R it can be *lifted* to a step $R^{\ell} : t^{\ell} \to s^{\ell}$ in the labeled λ-calculus, such that s^{ℓ} is a variant of s. Similarly, given a derivation $\rho : t \twoheadrightarrow s$ and a variant t^{ℓ} of t, the derivation ρ can be lifted to a derivation $\rho^{\ell} : t^{\ell} \twoheadrightarrow s^{\ell}$. Finally, it can be shown that labels characterize redex families as follows. Let t_0 be a starting term, and let ρR and σS be two hredexes in the λ-calculus, whose source is t_0. Then ρR and σS are in the same family if and only if for an initially labeled variant t_0^{ℓ} of t_0 the corresponding lifts $\rho^{\ell} R^{\ell}$ and $\sigma^{\ell} S^{\ell}$ verify that R^{ℓ} and S^{ℓ} have the same name.

Going back to our example of Figure 6.1, let us show that the hredexes $S_1 R_2 T_3$ and $R_1 S_3 T_2$ are in the same family, this time using the labeled λ-calculus. That is, let us show that T_3 and T_2 are assigned the same name, when starting from the same initially labeled term. Consider an initially labeled variant of $\Delta(FI)$:

$$@^{\mathbf{a}}(\underbrace{\lambda^{\mathbf{b}} x.@^{\mathbf{c}}(x^{\mathbf{d}}, x^{\mathbf{e}})}_{\Delta}, \underbrace{@^{\mathbf{f}}(\underbrace{\lambda^{\mathbf{g}} y.@^{\mathbf{h}}(y^{\mathbf{i}}, z^{\mathbf{j}})}_{F}, \underbrace{\lambda^{\mathbf{k}} w.w^{\mathbf{l}}}_{I})}_{})$$

and consider the lifted derivations $S_1^\ell R_2^\ell T_3^\ell$ and $R_1^\ell S_3^\ell T_3^\ell$ of the hredexes in question. Note that the name of T_3^ℓ is $\mathbf{i}[\mathbf{g}]\mathbf{k}$:

$$\begin{array}{ll} & @^{\mathbf{a}}(\lambda^{\mathbf{b}}x.@^{\mathbf{c}}(x^{\mathbf{d}}, x^{\mathbf{e}}), \underline{@^{\mathbf{f}}(\lambda^{\mathbf{g}}y.@^{\mathbf{h}}(y^{\mathbf{i}}, z^{\mathbf{j}}), \lambda^{\mathbf{k}}w.w^{\mathbf{l}})}) \\ \xrightarrow{S_1^\ell} & \underline{@^{\mathbf{a}}(\lambda^{\mathbf{b}}x.@^{\mathbf{c}}(x^{\mathbf{d}}, x^{\mathbf{e}}), @^{\mathbf{f}[\mathbf{g}]\mathbf{h}}(\lambda^{\mathbf{i}[\mathbf{g}]\mathbf{k}}w.w^{\mathbf{l}}, z^{\mathbf{j}}))} \\ \xrightarrow{R_2^\ell} & @^{\mathbf{a}[\mathbf{b}]\mathbf{c}}(\underbrace{@^{\mathbf{d}[\mathbf{b}]\mathbf{f}[\mathbf{g}]\mathbf{h}}(\lambda^{\mathbf{i}[\mathbf{g}]\mathbf{k}}w.w^{\mathbf{l}}, z^{\mathbf{j}})}_{\text{The redex } T_3^\ell.}, @^{\mathbf{e}[\mathbf{b}]\mathbf{f}[\mathbf{g}]\mathbf{h}}(\lambda^{\mathbf{i}[\mathbf{g}]\mathbf{k}}w.w^{\mathbf{l}}, z^{\mathbf{j}})) \end{array}$$

and the name of T_2^ℓ is indeed also $\mathbf{i}[\mathbf{g}]\mathbf{k}$:

$$\begin{array}{ll} & \underline{@^{\mathbf{a}}(\lambda^{\mathbf{b}}x.@^{\mathbf{c}}(x^{\mathbf{d}}, x^{\mathbf{e}}), @^{\mathbf{f}}(\lambda^{\mathbf{g}}y.@^{\mathbf{h}}(y^{\mathbf{i}}, z^{\mathbf{j}}), \lambda^{\mathbf{k}}w.w^{\mathbf{l}}))} \\ \xrightarrow{R_1^\ell} & @^{\mathbf{a}[\mathbf{b}]\mathbf{c}}(@^{\mathbf{d}[\mathbf{b}]\mathbf{f}}(\lambda^{\mathbf{g}}y.@^{\mathbf{h}}(y^{\mathbf{i}}, z^{\mathbf{j}}), \lambda^{\mathbf{k}}w.w^{\mathbf{l}}), \underline{@^{\mathbf{e}[\mathbf{b}]\mathbf{f}}(\lambda^{\mathbf{g}}y.@^{\mathbf{h}}(y^{\mathbf{i}}, z^{\mathbf{j}}), \lambda^{\mathbf{k}}w.w^{\mathbf{l}})}) \\ \xrightarrow{S_3^\ell} & @^{\mathbf{a}[\mathbf{b}]\mathbf{c}}(@^{\mathbf{d}[\mathbf{b}]\mathbf{f}}(\lambda^{\mathbf{g}}y.@^{\mathbf{h}}(y^{\mathbf{i}}, z^{\mathbf{j}}), \lambda^{\mathbf{k}}w.w^{\mathbf{l}}), \underbrace{@^{\mathbf{e}[\mathbf{b}]\mathbf{f}[\mathbf{g}]\mathbf{h}}(\lambda^{\mathbf{i}[\mathbf{g}]\mathbf{k}}w.w^{\mathbf{l}}, z^{\mathbf{j}})}_{\text{The redex } T_2^\ell.}) \end{array}$$

Extraction. The third and last characterization of redex families is based on an algorithmic procedure that, given an hredex ρR, calculates a *canonical representative* $\rho_0 R_0$ of its family. The difficulty of defining this relation, as noted by Lévy, is that given two hredexes that are in the same family according to the zig-zag relation, $\rho R \leftrightsquigarrow \sigma S$, they do not necessarily have a common ancestor, *i.e.* it is not necessarily the case that there exists an hredex τT such that $\tau T \leqslant \rho R$ and $\tau T \leqslant \sigma S$. In our running example of Figure 6.1, it would seem at first sight that the common ancestor of the hredexes $S_1 R_1 T_2$ and $R_1 S_3 T_2$ should be the hredex $S_1 T_7$. Actually, even though $S_1 T_7 \leqslant S_1 R_1 T_2$ holds, it is not the case that $S_1 T_7 \leqslant R_1 S_3 T_2$, as this would imply that $S_1 \sqsubseteq R_1 S_3$, but S_3 is only one of the two copies of S_1, *i.e.* it is not a complete development of S_1 / R_1.

The solution proposed by Lévy is to introduce a binary relation $(\rhd)$ between hredexes for which a common ancestor property does hold. Let us first mention a few auxiliary definitions.

1. If R is a redex and σ is a coinitial derivation, then σ is *disjoint* from R if the source is of the form $\mathtt{C}\langle t, s\rangle$, where $\mathtt{C}$ is a two-hole context, the step R takes place inside t, and the derivation σ takes place inside s.

2. If R is a redex and σ is a coinitial derivation, then σ is *internal to the body* of R if the source is of the form $\mathtt{C}\langle(\lambda x.t)s\rangle$, the step R contracts $(\lambda x.t)s$, and the derivation σ takes place inside t.

3. If R is a redex and σ is a composable derivation (*i.e.* $R\sigma$ is well-defined), then σ is *internal to the i-th copy of the argument* of R if the source is of the form $\mathtt{C}\langle(\lambda x.t)s\rangle$, the step R contracts $(\lambda x.t)s$, and the derivation σ takes place inside the i-th copy of s (corresponding to the i-th occurrence of x in t).

The extraction relation is a binary relation $\rhd$ between hredexes, defined as the union of the following four rules:

$$\begin{array}{rcll} \rho R S & \rhd_1 & \rho S_0 & \text{if } S \in S_0/R \\ \rho(R \sqcup \sigma) & \rhd_2 & \rho\sigma & \text{if } \sigma \text{ is not empty and it is disjoint from } R \\ \rho(R \sqcup \sigma) & \rhd_3 & \rho\sigma & \text{if } \sigma \text{ is not empty and it is internal to the body of } R \\ \rho R \sigma & \rhd_4^i & \rho\sigma_0 & \text{if } \sigma \text{ is not empty, it is internal to the } i\text{-th copy of } R, \text{ and } \sigma_0/R = \sigma \,||_i\, R \end{array}$$

The notation $\sigma \,||_i\, R$ stands for the *parallelization* of σ with respect to R, defined as follows by induction on σ:

$$\begin{array}{rcl} \epsilon \,||_i\, R & \stackrel{\text{def}}{=} & \epsilon \\ T\tau \,||_i\, R & \stackrel{\text{def}}{=} & (T_0/R)((\tau/(T_0/RT)) \,||_i\, (R/T_0)) \quad \text{if } T \in T_0/R \end{array}$$

(For more details on the motivation and properties of this definition see [110, Def. 4.7] or [14, Sec. 5.2]). This algorithmic extraction procedure can be shown to be terminating and confluent. Moreover, it characterizes redex families as follows: two hredexes are in the same family if and only if there exists an hredex τT such that $\rho R \rhd^\star \tau T$ and $\sigma S \rhd^\star \tau T$.

To complete the example of Figure 6.1, let us show that the hredexes $S_1 R_2 T_3$ and $R_1 S_3 T_2$ belong to the same family, this time using the extraction procedure. Indeed, note that:

$$\begin{array}{rcll} S_1 R_2 T_3 & \rhd_1 & S_1 T_7 & \text{since } T \in T_0/S \\ \\ R_1 S_3 T_2 & \rhd_4^2 & S_1 T_7 & \text{since } S_3 T_2 \text{ is internal to the second copy of } R_1 \\ & & & \text{and } S_3 T_2 \,||_2\, R_1 = S_3 S_5 T_3 T_5 = S_1 T_7/R_1 \end{array}$$

Finite Family Developments

A remarkable result that can only be stated and proved after the notion of redex family has been introduced is the *Finite Family Developments* theorem. Recall that the λ-calculus is an orthogonal axiomatic rewriting system (Thm. 2.73), and in particular it enjoys the finite developments property (Def. 2.33). It states that, in the λ-calculus, given a starting term t_0 and a set $\mathcal{M}$ of redexes of t_0, there are no infinite *developments* of $\mathcal{M}$. That is, there are no infinite sequences that only contract residuals of redexes in the set $\mathcal{M}$. The Finite Family Developments theorem is a strong generalization of this result. Rather than considering a set $\mathcal{M}$ of *redexes* of t_0, it allows us to consider a set $\mathcal{F}$ of *redex families* of t_0. In turn, *developments* of $\mathcal{M}$ are generalized to *family developments* of $\mathcal{F}$. A *family development* of $\mathcal{F}$ is any reduction sequence $R_1 \dots R_n$ such that, for all $i \in \{1, ..., n\}$ the family of the hredex $R_1 \dots R_i$ is in $\mathcal{F}$. The *Finite Family Developments* theorem states that if $\mathcal{F}$ is a finite set of redex families, there are no infinite family developments of $\mathcal{F}$.

Below we compare the notions involved in the Finite Developments theorem with the notions involved in the Finite Family Developments theorem:

Finite Developments	Finite Family Developments
redex of t_0	redex family of t_0
set $\mathcal{M}$ of redexes of t_0	set of $\mathcal{F}$ of redex families of t_0
development of $\mathcal{M}$	family development of $\mathcal{F}$
all developments of $\mathcal{M}$ are finite	all family developments of $\mathcal{F}$ are finite *if $\mathcal{F}$ is finite*

In the last entry, note that the Finite Developments theorem does not need to explicitly require that $\mathcal{M}$ is a finite set, since the λ-calculus is *finitely branching*, so this requirement is automatically met.

For example, consider once again the term $\Delta(FI)$ whose reduction graph is depicted in Figure 6.1. There are three redex families in total:

1. the redex family of the hredex $\Delta(FI) \xrightarrow{R_1} FI(FI)$,
2. the redex family of the hredex $\Delta(FI) \xrightarrow{S_1} \Delta(Iz)$,
3. the redex family of the hredex $\Delta(FI) \xrightarrow{S_1} \Delta(Iz) \xrightarrow{T_7} \Delta\, z$.

Every hredex starting from $\Delta(FI)$ is in one of these three families. In Figure 6.1 the names of the steps have been chosen deliberately so that all hredexes ending in a step R_k belong to the first family, all hredexes ending in a step S_k belong to the second family, and all hredexes ending in a step T_k belong to the third family. For instance, the hredex:

$$\Delta(FI) \xrightarrow{R_1} FI(FI) \xrightarrow{S_3} FI(Iz)$$

is in the same redex family as S_1. Let us write $\mathsf{Fam}_{\leftrightsquigarrow}(\rho R)$ for the redex family of the hredex ρR, that is, for its $\leftrightsquigarrow$-equivalence class. Then the following are all the possible family developments, not necessarily maximal, of the set of redex families $\mathcal{F} = \{\mathsf{Fam}_{\leftrightsquigarrow}(R_1), \mathsf{Fam}_{\leftrightsquigarrow}(S_1)\}$:

$$\begin{array}{lllllll}
\Delta(FI) & \xrightarrow{R_1} & FI(FI) & & & & \\
\Delta(FI) & \xrightarrow{R_1} & FI(FI) & \xrightarrow{S_2} & Iz(FI) & & \\
\Delta(FI) & \xrightarrow{R_1} & FI(FI) & \xrightarrow{S_2} & Iz(FI) & \xrightarrow{S_4} & Iz(Iz) \\
\Delta(FI) & \xrightarrow{R_1} & FI(FI) & \xrightarrow{S_3} & FI(Iz) & \xrightarrow{S_5} & Iz(Iz) \\
\Delta(FI) & \xrightarrow{S_1} & \Delta(Iz) & & & & \\
\Delta(FI) & \xrightarrow{S_1} & \Delta(Iz) & \xrightarrow{R_2} & Iz(Iz) & &
\end{array}$$

In fact, all the redex families in $\mathcal{F}$ have a representative that consists of a single step, which means that family developments of $\mathcal{F}$ are actually ordinary developments.

For a different example, let $\mathcal{F} = \{\mathsf{Fam}_{\leftrightsquigarrow}(S_1), \mathsf{Fam}_{\leftrightsquigarrow}(S_1T_7)\}$. Now there are only two possible family developments of $\mathcal{F}$:

$$\begin{array}{llll}
\Delta(FI) & \xrightarrow{S_1} & \Delta(Iz) & \\
\Delta(FI) & \xrightarrow{S_1} & \Delta(Iz) & \xrightarrow{T_7} \quad \Delta\, z
\end{array}$$

In this case, S_1T_7 is not a development of any set of redexes, since the step T_7 has been *created* by S_1, *i.e.* it has no ancestor.

For a slightly more interesting application of the Finite Family Developments theorem, consider the well-known non-terminating term Ω where $\Omega = (\lambda x.xx)\lambda x.xx$. It has a single redex R:

$$\Omega \xrightarrow{R} \Omega$$

this results in an infinite number of hredexes. Let us write R^n for the hredex of the form $\underbrace{R \ldots R}_{n \text{ times}}$ for each $n \in \mathbb{N}$:

$$
\begin{array}{lllllllll}
R^1: & \Omega & \xrightarrow{R} & \Omega & & & & & \\
R^2: & \Omega & \xrightarrow{R} & \Omega & \xrightarrow{R} & \Omega & & & \\
R^3: & \Omega & \xrightarrow{R} & \Omega & \xrightarrow{R} & \Omega & \xrightarrow{R} & \Omega \\
\ldots &
\end{array}
$$

It can be checked that R^n and R^m belong to the same family if and only if $n = m$. The intuitive reason is that in a reduction sequence like RR the second step is created by the first one, and has no ancestor. The infinite reduction $\Omega \xrightarrow{R} \Omega \xrightarrow{R} \Omega \ldots$ is a family development of the *infinite* set of redex families $\mathcal{F} = \{\mathsf{Fam}_{\leftsquigarrow}(R^n) \mid n \in \mathbb{N}\}$. The Finite Family Developments theorem ensures that, given any *finite* subset $\mathcal{G} \subseteq \mathcal{F}$, any family development of $\mathcal{G}$ must terminate.

Pointers on Optimality Theory

There has been much work surrounding the theory of optimal reductions. We have already mentioned the foundational works of Vuillemin [143, 144] on recursive program schemes, Staples [132] on combinatory logic, and Lévy [109, 110] together with Berry [27] on the λ-calculus.

John Lamping was the first to propose a data structure (*sharing graphs*) capable of implementing Lévy's optimal reduction [98].

Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy [63] explained Lamping's sharing graphs in terms of Girard's Geometry of Interaction.

Cosimo Laneve [101] studied optimality in the very general context of interaction systems.

Andrea Asperti and Cosimo Laneve [15] characterized redex families by characterizing *proper paths*: paths in the graph-representation of a λ-term that connect an application and an abstraction forming a *virtual redex*, *i.e.* a potential interaction.

John Glauert and Zurab Khasidashvili [61] generalized Lévy's optimality result in an axiomatic framework (*Deterministic Family Structures*).

Julia Lawall and Harry Mairson [104, 105] studied the question of what constitutes a cost model for the λ-calculus, proposed a cost model based on Lévy labels, and proved that Lamping's sharing graphs satisfy the proposed cost model.

Stefano Guerrini [69] studied the general theory of sharing graphs, independently of the calculus to be implemented, using Girard's Geometry of Interaction.

Andrea Asperti and Harry Mairson showed [16] that, after a sequence of n steps of β-reduction, the number of redexes belonging to a given redex family is not necessarily bounded by $O(2^n)$, $O(2^{2^n})$ or, in general, $O(K_\ell(n))$ where $K_\ell(n)$ is a tower of ℓ 2s with an n on top.

Vincent van Oostrom et al. [141, 30] studied the notion of redex family in the context of higher-order rewriting.

More recently, Thibaut Balabonski studied optimal reduction for a calculus with dynamic patterns [19], and proved that, in the case of *weak* reduction, *i.e.* disallowing the contraction of redexes below lambdas, call-by-need is an optimal evaluation strategy [20].

In [70], Stefano Guerrini and Marco Solieri show that, in the case of light linear logics, sharing graphs do not require bookkeeping, and they obtain a bound for the overhead introduced by sharing.

A thorough reference book on optimal reductions is Asperti and Guerrini's [14].

6.1.2 Our Work

This chapter is the result of collaboration with Eduardo Bonelli and it is structured as follows. We highlight in boldface what we consider to be the main contributions:

- In Section 6.2, we motivate some design decisions behind a calculus with Lévy labels, and we **define a variant of the LSC with Lévy labels**, the LLSC (Def. 6.6).
- In Section 6.3, we study the properties of the LLSC. In particular:
  1. In Section 6.3.1 we study its basic syntactical properties.
  2. In Section 6.3.2 we show that **the LLSC is an orthogonal axiomatic rewriting system** (Prop. 6.32).
  3. In Section 6.3.3 we prove that the LLSC is weakly normalizing for bounded reduction (Prop. 6.45), *i.e.* when reduction is restricted to labels of bounded height.
  4. In Section 6.3.4 we strengthen this result, proving that the **LLSC is strongly normalizing for bounded reduction** (Thm. 6.51).
  5. In Section 6.3.5 we give two proofs that the LLSC is confluent, building on previous results.

In the following chapter (Chapter 7), we apply the LLSC to derive results about the LSC without labels; in particular, optimality, standardization, and normalization results.

6.2 The LSC with Lévy Labels

6.2.1 What is a Calculus with Lévy Labels?

Our aim is to define a variant of the Linear Substitution Calculus (LSC) with *Lévy labels*. We are interested in Lévy labels both for a conceptual reason—gaining understanding of the ways in which computations can interact, contribute to each other, and be shared—and a practical one—labels can be a helpful syntactical tool for attacking further problems. Regarding the latter goal, any conceivable notion of labeling would be welcome as long as it aids us in proving theorems. The former goal, instead, is much less clearly defined, and one may wonder what abstract properties make a "Lévy labeled" calculus worthy of its name.

There does not seem to be a completely satisfactory answer to this question. Let us take a look at a list of properties that Lévy labels enjoy in the context of the λ-calculus. We will take these properties as guiding principles for designing a Lévy labeled variant of the LSC.

Bestiary of Principles of Lévy Labels

1. **Lift.** Unlabeled reduction sequences may be lifted to labeled reduction sequences, giving an arbitrary labeling to the starting term.

   For instance, the step
   $$(\lambda x.x)y \to y$$
   may be lifted to a labeled step
   $$@^{\alpha}(\lambda^{\beta}x.x^{\gamma}, y^{\delta}) \to y^{\alpha[\beta]\gamma\lfloor\beta\rfloor\delta}$$
   regardless of the choice of the labels $\alpha, \beta, \gamma, \delta$.

2. **Initial.** In an initially labeled term, different redexes have different names.

   Indeed, the name of a redex is the label decorating its abstraction and, in an initially labeled term, labels decorating different nodes are required to be pairwise distinct.

3. **Copy.** If a hredex $\rho' R'$ is a copy of the hredex ρR, then ρR and $\rho' R'$ have the same name.

   For instance, in the permutation diagram of Figure 6.2 the names of R and SR' are both **b**, and the names of S and RS' are both **d**.

   $$\begin{array}{ccc} @^{\mathbf{a}}(\lambda^{\mathbf{b}}x.@^{\mathbf{c}}(\lambda^{\mathbf{d}}y.y^{\mathbf{e}}, x^{\mathbf{f}}), z^{\mathbf{g}}) & \xrightarrow{R} & @^{\mathbf{a}[\mathbf{b}]\mathbf{c}}(\lambda^{\mathbf{d}}y.y^{\mathbf{e}}, z^{\mathbf{f}[\mathbf{b}]\mathbf{g}}) \\ \downarrow S & & \downarrow S' \\ @^{\mathbf{a}}(\lambda^{\mathbf{b}}x.x^{\mathbf{c}\lceil\mathbf{d}\rceil\mathbf{e}\lfloor\mathbf{d}\rfloor\mathbf{f}}, z^{\mathbf{g}}) & \xrightarrow{R'} & z^{\mathbf{a}[\mathbf{b}]\mathbf{c}\lceil\mathbf{d}\rceil\mathbf{e}\lfloor\mathbf{d}\rfloor\mathbf{f}[\mathbf{b}]\mathbf{g}} \end{array}$$

   Figure 6.2: Permutation diagram of $(\lambda x.Ix)z$ in the labeled λ-calculus

   More strongly, redex names characterize exactly redex families, as defined using the zig-zag relation.

4. **Creation.** Whenever a redex R creates a redex S, the name of R is a *sublabel* of the name of RS.

   As an example, observe that this is the case in the following three representative cases of redex creation. We write $\alpha \subseteq \beta$ for the binary relation stating that α is a sublabel of β:

   4.1 **Creation case I:** $IIz \xrightarrow{R} Iz \xrightarrow{S} z$. Then $\mathbf{c} \subseteq \mathbf{b}\lceil\mathbf{c}\rceil\mathbf{d}\lfloor\mathbf{c}\rfloor\mathbf{e}$:
   $$@^{\mathbf{a}}(@^{\mathbf{b}}(\lambda^{\mathbf{c}}x.x^{\mathbf{d}}, \lambda^{\mathbf{e}}y.y^{\mathbf{f}}), z^{\mathbf{g}}) \xrightarrow{R} @^{\mathbf{a}}(\lambda^{\mathbf{b}[\mathbf{c}]\mathbf{d}[\mathbf{c}]\mathbf{e}}y.y^{\mathbf{f}}, z^{\mathbf{g}}) \xrightarrow{S} z^{\mathbf{a}[\mathbf{b}[\mathbf{c}]\mathbf{d}[\mathbf{c}]\mathbf{e}]\mathbf{f}[\mathbf{b}[\mathbf{c}]\mathbf{d}[\mathbf{c}]\mathbf{e}]\mathbf{g}}$$

   4.2 **Creation case II:** $(\lambda x.I)yz \xrightarrow{R} Iz \xrightarrow{S} z$. Then $\mathbf{c} \subseteq \mathbf{b}\lceil\mathbf{c}\rceil\mathbf{d}$:
   $$@^{\mathbf{a}}(@^{\mathbf{b}}(\lambda^{\mathbf{c}}x.\lambda^{\mathbf{d}}w.w^{\mathbf{e}}, y^{\mathbf{f}}), z^{\mathbf{g}}) \xrightarrow{R} @^{\mathbf{a}}(\lambda^{\mathbf{b}[\mathbf{c}]\mathbf{d}}w.y^{\mathbf{e}[\mathbf{c}]\mathbf{f}}, z^{\mathbf{g}}) \xrightarrow{S} z^{\mathbf{a}[\mathbf{b}[\mathbf{c}]\mathbf{d}]\mathbf{e}[\mathbf{c}]\mathbf{f}[\mathbf{b}[\mathbf{c}]\mathbf{d}]\mathbf{g}}$$

4.3 **Creation case III:** $(\lambda x.xy)I \xrightarrow{R} Iy \xrightarrow{S} y$. Then $\mathbf{b} \subseteq \mathbf{d}\lfloor\mathbf{b}\rfloor\mathbf{f}$:

$$@^{\mathbf{a}}(\lambda^{\mathbf{b}}x.@^{\mathbf{c}}(x^{\mathbf{d}}, y^{\mathbf{e}}), \lambda^{\mathbf{f}}z.z^{\mathbf{g}}) \xrightarrow{R} @^{\mathbf{a}\lceil\mathbf{b}\rceil\mathbf{c}}(\lambda^{\mathbf{d}\lfloor\mathbf{b}\rfloor\mathbf{f}}z.z^{\mathbf{g}}, y^{\mathbf{e}}) \xrightarrow{S} y^{\mathbf{a}\lceil\mathbf{b}\rceil\mathbf{c}\lceil\mathbf{d}\lfloor\mathbf{b}\rfloor\mathbf{f}\rceil\mathbf{g}\lfloor\mathbf{d}\lfloor\mathbf{b}\rfloor\mathbf{f}\rfloor\mathbf{e}}$$

5. **Contribution.** The name of a hredex ρR is a "subname" of the name of σS if and only if the family of ρR *contributes* to the family of σS in a semantical sense. This will be made more precise later.

6. **Confluence.** The Lévy labeled λ-calculus is confluent.

   For instance, the permutation diagram of Figure 6.2 is a (quite easy) illustration of the fact that the weak Church-Rosser property holds.

7. **Termination.** If labeled reduction is restricted to contracting redexes whose names are among a finite set of names, the resulting restricted system should be terminating. This property entails the Finite Family Developments theorem.

   The intuitive reason for this termination property to hold is the following. Let us say that a redex R is first-generation if it is present on the starting term, and $(n+1)$th-generation if the redexes that contribute to creating R are at most nth-generation. An infinite reduction sequence cannot contract only first-generation redexes since that would be an infinite development, contradicting the Finite Developments theorem. More in general, it can be seen that an infinite reduction sequence must contract nth-generation redexes, for arbitrarily large values of n. By the **Contribution** principle, newly created redexes include the names of all the redexes that have contributed to its creation. So, as evaluation proceeds, newer generations have larger and larger names. It follows that an infinite reduction sequence must involve redexes having an infinite number of names.

   As an illustration of this phenomenon consider the term $\Omega = (\lambda x.xx)\lambda x.xx$ and observe that the name of R is included in the name of RR, *i.e.* $\mathbf{b} \subseteq \mathbf{d}\lfloor\mathbf{b}\rfloor\mathbf{f}$:

   $$@^{\mathbf{a}}(\lambda^{\mathbf{b}}x.@^{\mathbf{c}}(x^{\mathbf{d}}, x^{\mathbf{e}}), \lambda^{\mathbf{f}}x.@^{\mathbf{g}}(x^{\mathbf{h}}, x^{\mathbf{i}})) \to @^{\mathbf{a}\lceil\mathbf{b}\rceil\mathbf{c}}(\lambda^{\mathbf{d}\lfloor\mathbf{b}\rfloor\mathbf{f}}x.@^{\mathbf{g}}(x^{\mathbf{h}}, x^{\mathbf{i}}), \lambda^{\mathbf{e}\lfloor\mathbf{b}\rfloor\mathbf{f}}x.@^{\mathbf{g}}(x^{\mathbf{h}}, x^{\mathbf{i}}))$$

   By appropriately renaming labels, this also shows that the name of RR is included in the name of RRR, the name of RRR is included in the name of $RRRR$, and so on. This confirms that the infinite reduction sequence $\Omega \to \Omega \to \ldots$ involves an infinite number of redex names.

8. **Reconstruction.** The reduction history of a term can be reconstructed from its labeling, modulo permutation equivalence, supposing that we start from an initially labeled term.

   For example, given the term $x^{\mathbf{a}}$ we know that it must be the starting term: its history must be empty. Any non-empty reduction yielding a variable x as its final result would have left a trace. That is, there would be other labels decorating x, indicating that some β-redexes were contracted before.

On the other hand, consider the two possible reductions $I(Iy) \to Iy$:

$$R : \underline{I(Iy)} \to Iy \quad \text{contracting the outermost redex}$$

$$S : I(\underline{Iy}) \to Iy \quad \text{contracting the innermost redex}$$

This is what Lévy calls a *syntactic accident*: two derivations happen to start and end on the same terms, but this is accidental. The **Reconstruction** property tells us that the labeled calculus is able to discriminate between computations that start and end on the same terms *by accident* and those that do it *by necessity*, by performing the same computational work. Carrying on with the example, if we start from an initially labeled source term, the labeled lifts R^ℓ and S^ℓ of the steps R and S yield different labeled target terms:

$$\begin{array}{ccc} @^{\mathbf{a}}(\lambda^{\mathbf{b}}x.x^{\mathbf{c}}, @^{\mathbf{d}}(\lambda^{\mathbf{e}}x.x^{\mathbf{f}}, y^{\mathbf{g}})) & \xrightarrow{R^\ell} & @^{\mathbf{a}\lceil\mathbf{b}\rceil\mathbf{c}\lfloor\mathbf{b}\rfloor\mathbf{d}}(\lambda^{\mathbf{e}}x.x^{\mathbf{f}}, y^{\mathbf{g}}) \\ \downarrow{\scriptstyle S^\ell} & & \\ @^{\mathbf{a}}(\lambda^{\mathbf{b}}x.x^{\mathbf{c}}, y^{\mathbf{d}\lceil\mathbf{e}\rceil\mathbf{f}\lfloor\mathbf{e}\rfloor\mathbf{g}}) & & \end{array}$$

From the labeled target term $@^{\mathbf{a}\lceil\mathbf{b}\rceil\mathbf{c}\lfloor\mathbf{b}\rfloor\mathbf{d}}(\lambda^{\mathbf{e}}x.x^{\mathbf{f}}, y^{\mathbf{g}})$ we can tell that it is the redex R which has been fired, even in the presence of a syntactic accident. Similarly, from the labeled term $@^{\mathbf{a}}(\lambda^{\mathbf{b}}x.x^{\mathbf{c}}, y^{\mathbf{d}\lceil\mathbf{e}\rceil\mathbf{f}\lfloor\mathbf{e}\rfloor\mathbf{g}})$ we can deduce its history, and conclude that it must be the single step derivation S. If we extend these derivations with their relative residuals R' and S', we obtain a permutation diagram ending on the same labeled term:

$$\begin{array}{ccc} @^{\mathbf{a}}(\lambda^{\mathbf{b}}x.x^{\mathbf{c}}, @^{\mathbf{d}}(\lambda^{\mathbf{e}}x.x^{\mathbf{f}}, y^{\mathbf{g}})) & \xrightarrow{R^\ell} & @^{\mathbf{a}\lceil\mathbf{b}\rceil\mathbf{c}\lfloor\mathbf{b}\rfloor\mathbf{d}}(\lambda^{\mathbf{e}}x.x^{\mathbf{f}}, y^{\mathbf{g}}) \\ \downarrow{\scriptstyle S^\ell} & & \downarrow{\scriptstyle S'^\ell} \\ @^{\mathbf{a}}(\lambda^{\mathbf{b}}x.x^{\mathbf{c}}, y^{\mathbf{d}\lceil\mathbf{e}\rceil\mathbf{f}\lfloor\mathbf{e}\rfloor\mathbf{g}}) & \xrightarrow{R'^\ell} & y^{\mathbf{a}\lceil\mathbf{b}\rceil\mathbf{c}\lfloor\mathbf{b}\rfloor\mathbf{d}\lceil\mathbf{e}\rceil\mathbf{f}\lfloor\mathbf{e}\rfloor\mathbf{g}} \end{array}$$

In fact the extended derivations RS' and SR' are permutation equivalent, and from the labeled target term $y^{\mathbf{a}\lceil\mathbf{b}\rceil\mathbf{c}\lfloor\mathbf{b}\rfloor\mathbf{d}\lceil\mathbf{e}\rceil\mathbf{f}\lfloor\mathbf{e}\rfloor\mathbf{g}}$ we can deduce that both redexes R and S have been fired. Remark that the order is irrelevant as we are interested in histories only modulo permutation equivalence.

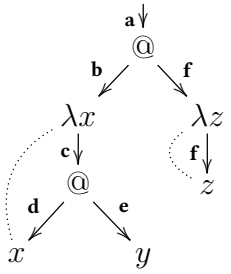
9. **Paths.** Redex names correspond to paths in the graph-representation of the starting term, connecting two nodes that may take part in an interaction.

   For example, let us recall the following labeled reduction:

$$@^{\mathbf{a}}(\lambda^{\mathbf{b}}x.@^{\mathbf{c}}(x^{\mathbf{d}}, y^{\mathbf{e}}), \lambda^{\mathbf{f}}z.z^{\mathbf{g}}) \xrightarrow{R} @^{\mathbf{a}\lceil\mathbf{b}\rceil\mathbf{c}}(\lambda^{\mathbf{d}\lfloor\mathbf{b}\rfloor\mathbf{f}}z.z^{\mathbf{g}}, y^{\mathbf{e}}) \xrightarrow{S} y^{\mathbf{a}\lceil\mathbf{b}\rceil\mathbf{c}\lceil\mathbf{d}\lfloor\mathbf{b}\rfloor\mathbf{f}\rceil\mathbf{g}\lfloor\mathbf{d}\lfloor\mathbf{b}\rfloor\mathbf{f}\rfloor\mathbf{e}}$$

   The starting term, seen as a graph, has the following shape. Note that each node has an incoming edge, and labels on a subterm decorate the corresponding incoming edge. By convention, nodes corresponding to bound variables are connected back to the binding

abstraction node:



The name of the first redex R is **b**. Note that, naturally, **b** is an edge connecting an application node and an abstraction node. The name of the second redex S is **d**⌊**b**⌋**f**. The insight of Asperti and Laneve [15] is that, in general, redex names correspond to paths in the graph of the starting term, connecting an application node and an abstraction node. In this case, start from the application node at the bottom. The path **d**⌊**b**⌋**f** can be read as follows:

- Follow the edge **d** forwards to x.
- Follow the edge connecting x back to its binder λx.
- Follow the edge **b** backwards to the application node at the top.
- Follow the edge **f** forwards to λz.

The presence of this path indicates the presence of a *virtual redex*, a potential interaction between the application node at the bottom and λz.

In the remainder of this section we will formulate a labeled variant of the LSC. Later, in Section 6.3 we prove that the labeled variant of the LSC verifies most of the properties in the Bestiary. As a means of giving some cohesion to the array of quite disparate properties that we have just listed, we will show that the labeled LSC without the gc rule forms a *Deterministic Family Structure* (DFS). Deterministic Family Structures are an axiomatic framework introduced by Glauert and Khasidashvili [61] to generalize Lévy's theory of optimal reductions. Showing that the LSC without gc is a DFS will essentially consist of ensuring that it enjoys properties 1–7 in the list. We will also discuss reasons that suggest that it is not possible to define a labeled variant of the full LSC (including the gc rule) that verifies all the properties above—or at least not without a fundamentally different approach. We will, nevertheless, deal with the gc rule to the best of our capabilities, and we will show that many of properties above can still be verified, including the nontrivial properties of **Confluence** and **Termination**.

In this this thesis we do not deal with the last two properties, **Reconstruction** and **Paths**, and in fact we consider these to be pending open problems. In the case of **Reconstruction**, a technical impediment is that we have not been able to define an extraction procedure such as the one that has been described in Section 6.1.1 for the λ-calculus. In Section 8.2 we will describe some of the difficulties we have found in our attempt to define an extraction procedure. In the case of the last property, **Paths**, we do not foresee any fundamental obstruction for adapting it to the LSC without gc.

In any case, we hope that the reader will agree that the labeled LSC we propose deserves to be regarded as a *Lévy labeled* LSC.

6.2.2 Residual Theory for the LSC

Recall from Def. 2.75 that the LSC is defined as the rewriting relation $\to_{\mathtt{LSC}}$ obtained from the union of the three following rewriting rules, closed under arbitrary contexts.

Distant beta	$(\lambda x.t)\mathtt{L}\, s$	$\mapsto_{\mathtt{db}}$	$t[x\backslash s]\mathtt{L}$	
Linear substitution	$\mathtt{C}\langle\!\langle x\rangle\!\rangle[x\backslash t]$	$\mapsto_{\mathtt{ls}}$	$\mathtt{C}\langle\!\langle t\rangle\!\rangle[x\backslash t]$	
Garbage collection	$t[x\backslash s]$	$\mapsto_{\mathtt{gc}}$	t	if $x \notin \mathsf{fv}(t)$

As a starting point in our quest to define a Lévy labeled variant of the LSC, let us restate the most basic of the properties we are after. We would like the labeled calculus to give a *name* to each redex, in such a way that:

- If a redex R' is a residual of a redex R, then R and R' have the same name.
- If a redex R creates a redex S, then the name of R is a subname of the name of S.

Let us remark that in making these statements we are already presupposing the existence of an *a priori* theory of residuals. For instance, it seems intuitively clear that the `ls` redex S is the ancestor of the `ls` redexes S_1, S_2, S'_1, and S'_2 in the reduction graph of Figure 6.3. It seems

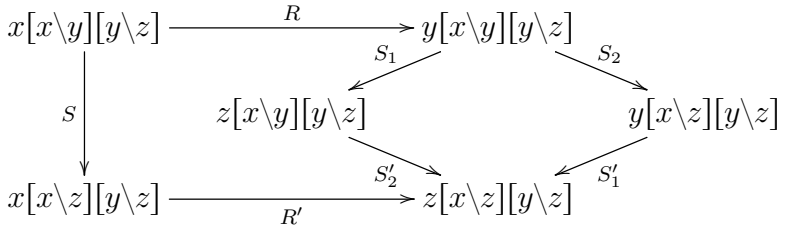


Figure 6.3: Reduction graph of $x[x\backslash y][y\backslash z]$

also clear that the `ls` step R creates the `db` step S in the following reduction:

$$x[x\backslash\lambda y.y]z \xrightarrow{R} (\lambda y.y)[x\backslash\lambda y.y]z \xrightarrow{S} y[y\backslash z][x\backslash\lambda y.y]$$

However, unlike in most other calculi, in the LSC steps interact *at a distance*. In fact the three rewriting rules involve some sort of non-local interaction. The `db` rule involves an interaction between an abstraction and an application that are separated by an arbitrary substitution context `L`. The `ls` rule involves an interaction between a variable that is bound to a substitution somewhere else in the term. Finally, the `gc` rule depends on the non-local condition that the variable bound by the substitution $[x\backslash s]$ does not occur anywhere in the body t. Adapting the standard techniques that are used to define residuals—*e.g.* in term rewriting systems—is not immediate.

Fortunately for us, in [6], Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi already provide a definition of residuals for the LSC, and they prove that it gives

rise to a quite well-behaved residual theory. Let us review the definitions and results of their work that will be relevant to our own.

Before proceeding, it is worth noting that in [6] residuals are defined using an auxiliary variant of the LSC that uses *labels*. The use of labels in the labeled calculus of [6] should not be confused with the Lévy labels that we are attempting to define. The purpose of labels in the labeled calculus of [6] is to provide an ancestor/descendant relation between the subterms of t and the subterms of s along a single rewriting step $t \to_{\mathsf{LSC}} s$. Lévy labels are a much more powerful formalism. In particular, Lévy labels give a name to every redex that is found along a derivation, including created redexes. To avoid confusion, we will depart from the nomenclature of Accattoli et al., and speak of *marks*, rather than labels, when referring to the labeled calculus of [6]. The nomenclature is also consistent with the *marked λ-calculus* that we used as an auxiliary tool to define residuals in the λ-calculus (*cf.* Def. 2.69).

Definition 6.1 (The marked LSC)**.** Consider a denumerable set of *marks* $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}, \ldots$. The set of *marked terms* is given by the following grammar:

$$\begin{array}{rcll} t, s, u, \ldots & ::= & x & \text{variable} \\ & \mid & x^{\mathfrak{a}} & \text{marked variable} \\ & \mid & \lambda x.t & \text{abstraction} \\ & \mid & \lambda x^{\mathfrak{a}}.t & \text{abstraction with marked variable} \\ & \mid & ts & \text{application} \\ & \mid & t[x\backslash s] & \text{substitution} \\ & \mid & t[x^{\mathfrak{a}}\backslash s] & \text{substitution with marked variable} \end{array}$$

The notations $\mathtt{L}$ for substitution contexts and $\mathtt{C}$ for arbitrary contexts are extended to allow marks. Similarly, the notion of free variables is extended to marked terms as expected, together with its notion of α-conversion. *Marked reduction* $\xrightarrow{\mathfrak{a}}$ on marked terms is defined as the contextual closure of the following rewriting rules:

$$\begin{array}{rcll} (\lambda x^{\mathfrak{a}}.t)\mathtt{L}\, s & \overset{\mathfrak{a}}{\mapsto}_{\mathtt{db}} & t[x\backslash s]\mathtt{L} & \\ \mathtt{C}\langle\!\langle x^{\mathfrak{a}} \rangle\!\rangle[x\backslash t] & \overset{\mathfrak{a}}{\mapsto}_{\mathtt{ls}} & \mathtt{C}\langle t\rangle[x\backslash t] & \\ t[x^{\mathfrak{a}}\backslash s] & \overset{\mathfrak{a}}{\mapsto}_{\mathtt{gc}} & t & \text{if } x \notin \mathsf{fv}(t) \end{array}$$

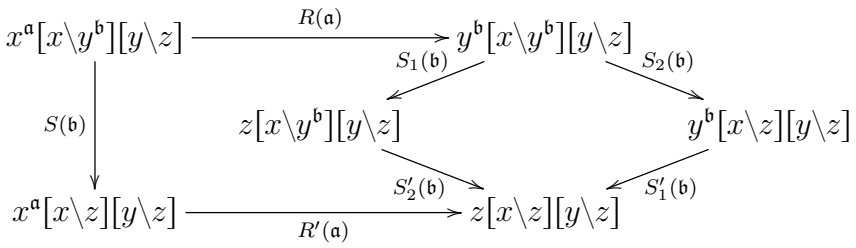
A *marked redex* is a redex R having a pattern of the form $(\lambda x^{\mathfrak{a}}.t)\mathtt{L}\, s$, $\mathtt{C}\langle\!\langle x^{\mathfrak{a}} \rangle\!\rangle[x\backslash t]$, or $t[x^{\mathfrak{a}}\backslash s]$, and $\mathfrak{a}$ is called the *mark of the redex* R. Note that a marked step $t \xrightarrow{\mathfrak{a}} s$ is decorated with the mark of the corresponding redex. On the other hand, *unmarked reduction* $\to$ on marked terms is defined as the contextual closure of the usual `db`, `ls` and `gc` rules—in this case, the redex is not marked but marks elsewhere in the term are allowed. The *anchor* of a redex (marked or not) is the variable possibly carrying its mark. If t is a marked term, t° is the term that results from erasing all the marks in t. If $t^\circ = s$, we say that t is a *variant* of s. In that case, we identify redexes of t and redexes of s via the obvious bijection.

Definition 6.2 (Residuals in the LSC)**.** Let R, S be two coinitial steps in the LSC:

$$R : t \to s \quad S : t \to u$$

Consider a marked variant t' of t having exactly one mark $\mathfrak{a}$ on the anchor of S. Let $R' : t' \to s'$ be the step corresponding to R via the obvious bijection. The set of *residuals of* S *after* R, written S/R, is the set of steps of the form $S' : s \to r$ for some term r, such that S' is marked with $\mathfrak{a}$ in the marked variant s' of s.

For example, the reduction graph of the term $x[x\backslash y][y\backslash z]$ (Figure 6.3) can be adapted to the marked LSC by marking the anchors of the redexes R and S, as below—we write $R(\mathfrak{a})$ to emphasize that $\mathfrak{a}$ is the mark of R:



Moreover, according to the definition of residual:

$$R/S = \{R'\} \quad S/R = \{S_1, S_2\} \quad S_1/S_2 = \{S'_1\} \quad S_2/S_1 = \{S'_1\} \quad R/R = \varnothing$$

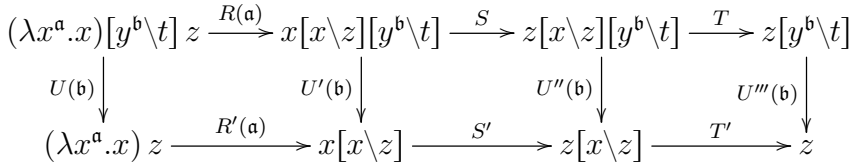
One should have in mind that, to calculate a set of residuals, for example S_1/S_2, we should start from a marked variant of the source term having a *single* marked redex, which is not the case for $y^{\mathfrak{b}}[x\backslash y^{\mathfrak{b}}][y\backslash z]$ in the diagram above. Recall also from Def. 2.31 that the residual relation can be extended to take the residuals of a step after a derivation, defining S/ρ by induction on ρ:

$$\begin{array}{rcl} S/\epsilon & \stackrel{\text{def}}{=} & \{S\} \\ S/R\rho & \stackrel{\text{def}}{=} & \{S'' \mid \exists S'.\ S' \in S/R \text{ and } S'' \in S'/\rho\} \end{array}$$

So in particular, in the diagram above we have:

$$S/RS_1 = \{S'_2\} \quad S/RS_2 = \{S'_1\}$$

For a different example of marked reduction, the following one involves a `db` step R, a `ls` step S, two `gc` steps T, U, and its residuals:



Recall from Section 2.2 that a step R *creates* S if the step S is not the residual of any S_0 after R. In this case, the `db` step R creates the `ls` step S, since x was originally bound by an abstraction but, after the `db` step, it becomes bound by a substitution, and is now susceptible of being substituted by z. Similarly, the `ls` step S creates the `gc` step T, as it exhausts the occurrences of x that are bound by the substitution $[x\backslash z]$, enabling the substitution to be garbage collected. Perhaps it is also interesting to note also that the newly created redexes S and T are not marked. In the case of the step T, there is no chance that it could be marked,

since the garbage collected substitution is created along the way, *i.e.* it comes from the right-hand side of a db step.

To conclude with this section, we restate two results that are already known to hold from [6].

Proposition 6.3. *The LSC forms an orthogonal axiomatic rewriting system.*

Proof. Recall from Def. 2.39 that an orthogonal axiomatic rewriting system in the sense of Melliès must verify four axioms:

1. **Autoerasure (AE).** That is, $R/R = \varnothing$ for every redex R.
   To prove this axiom, note that if t has a single marked redex $R(\mathfrak{a})$ and the redex $R : t \to s$ is fired, then s has no occurrences of $\mathfrak{a}$.

2. **Finite Residuals (FR).** That is, R/S is finite for every two coinitial redexes R, S.
   This axiom is immediate since the LSC is finitely branching, and the set R/S is a set of coinitial redexes, so it must be finite.

3. **Finite Developments (FD).** If $\mathcal{M}$ is a set of coinitial redexes, there are no infinite developments of $\mathcal{M}$.
   This axiom is Proposition 1 in [6]. It can be proved using the notion of *potential multiplicity*, similarly as in the proof that the λ-calculus verifies FD (Thm. 2.73).

4. **Semantic Orthogonality (SO).** For any two coinitial redexes R, S there exist complete developments ρ of R/S and σ of S/R such that ρ and σ are cofinal and, moreover, $R\sigma$ and $S\rho$ induce the same residual relation.
   This axiom is Proposition 2 in [6].

□

After presenting the Lévy labeled LSC, and using it as a tool, we will be able to give alternative proofs for **FD** and **SO**: termination of the labeled calculus restricted to bounded labels will be a generalization of **FD**, and confluence of the labeled calculus will be a generalization of **SO**. We also remind the reader that, as was discussed in Section 2.2, various results from [118] are automatically available in any orthogonal axiomatic rewriting system, in particular multisteps, residuals, permutation equivalence, and algebraic confluence.

Besides Prop. 6.3, there is a second result from the work by Accattoli et al. ([6]) that we should mention before going on, concerning redex creation in the LSC. Here we state an incomplete form of the result, for the sake of clarity. The fully-fledged result is stated and proved in the appendix (Prop. A.77).

Proposition 6.4 (Redex creation in the LSC — ♣ Prop. A.77)**.** *Let $t_1 \xrightarrow{R} t_2 \xrightarrow{S} t_3$ be a sequence of two redexes in the LSC such that R creates S. Then S is created in exactly one of seven possible ways. Here we provide only representative examples, see the appendix for the full statement and proof.*

1. `db` **creates** `db`. *For example,* $(\lambda x.(\lambda y.t)s)u \to (\lambda y.t)[x\backslash s]\, u \to t[y\backslash u][x\backslash s]$.
2. `db` **creates** `ls`. *For example,* $(\lambda x.xx)t \to (xx)[x\backslash t] \to (xt)[x\backslash t]$.
3. `db` **creates** `gc`. *For example,* $(\lambda x.y)t \to y[x\backslash t] \to y$.
4. `ls` **creates** `db` **upwards.** *For example,* $x[x\backslash\lambda y.t]s \to (\lambda y.t)[x\backslash\lambda y.t]s \to t[y\backslash s][x\backslash\lambda y.t]$.
5. `ls` **creates** `db` **downwards.** *For example,* $(xt)[x\backslash\lambda y.s] \to ((\lambda y.s)t)[x\backslash\lambda y.s] \to s[y\backslash t][x\backslash\lambda y.s]$.
6. `ls` **creates** `gc`. *For example,* $(yx)[x\backslash y] \to (yy)[x\backslash y] \to yy$.
7. `gc` **creates** `gc`. *For example,* $y[x\backslash z][z\backslash t] \to y[z\backslash t] \to y$.

Proof. The proof is by exhaustive case analysis on the three possible kinds of redexes that R and S might be (`db`, `ls`, or `gc`), and the position of the anchor of S in the term t_2. □

6.2.3 Definition of the Labeled LSC Without `gc`

For expository purposes, we start by giving a definition of a Lévy labeled variant of the LSC without the `gc` rule, and then discuss how to extend this definition to also contemplate the `gc` rule.

As a general convention, we use the symbol "ℓ" when naming constructions that correspond to labeled calculi, and the symbol "I" when naming constructions that only make sense in the calculus without `gc`. For example, $\mathcal{T}$ is the set of terms in the (unlabeled) LSC, $\mathcal{T}^{\ell}$ is the set of terms in the (full) labeled LSC, and $\mathcal{T}^{\ell I}$ is the set of terms in labeled LSC without `gc`[1].

Definition 6.5 (The Lévy labeled LSC without `gc`, LLSC^I)**.** Consider a denumerable set of *initial labels* $\mathcal{I} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots\}$. We assume the existence of a distinguished initial label $\bullet \in \mathcal{I}$. The set of *labels* $\mathcal{L}^I$ is defined by the following grammar:

$$\alpha, \beta, \gamma, \ldots ::= \mathbf{a} \mid \alpha\beta \mid \lceil\alpha\rceil \mid \lfloor\alpha\rfloor \mid \mathtt{db}(\alpha)$$

Labels are considered up to associativity of juxtaposition, *i.e.* for every $\alpha, \beta, \gamma \in \mathcal{L}$ we declare $(\alpha\beta)\gamma = \alpha(\beta\gamma)$ to hold. Labels that are not of the form $\alpha\beta$ are called *atomic*. The set of *labeled terms* $\mathcal{T}^{\ell I}$ is defined by the following grammar:

$$t, s, \ldots \quad ::= \quad x^{\alpha} \quad \mid \quad \lambda^{\alpha}x.t \quad \mid \quad @^{\alpha}(t, s) \quad \mid \quad t[x\backslash s]$$

Observe that there are labels over variables, abstractions, and applications, but not over substitutions. The *outermost atomic label* of a label α, written $\uparrow(\alpha)$, is defined as follows, by induction on the number of juxtapositions that take part in the construction of the label α:

$$\uparrow(\alpha) \stackrel{\text{def}}{=} \begin{cases} \uparrow(\alpha_1) & \text{if } \alpha = \alpha_1\alpha_2 \\ \alpha & \text{if } \alpha \text{ is atomic} \end{cases}$$

[1]The mnemonic for the symbol "I" is that the LSC corresponds to the full λ-calculus, while the LSC without `gc` corresponds to Church's λI-calculus.

For example $\uparrow([\mathbf{ab}]\mathbf{ac}) = [\mathbf{ab}]$. Similarly, the *innermost atomic label* of a label α, written $\downarrow(\alpha)$, is defined as follows:

$$\downarrow(\alpha) \stackrel{\text{def}}{=} \begin{cases} \downarrow(\alpha_2) & \text{if } \alpha = \alpha_1\alpha_2 \\ \alpha & \text{if } \alpha \text{ is atomic} \end{cases}$$

Note that $\uparrow(\alpha)$ and $\downarrow(\alpha)$ yield atomic labels, and that they are well-defined modulo associativity of juxtaposition. The *external label* of a term t, written $\ell(t)$, is defined as the label decorating its outermost node, ignoring substitutions:

$$\ell(x^\alpha) = \alpha \quad \ell(\lambda^\alpha x.t) = \alpha \quad \ell(@^\alpha(t,s)) = \alpha \quad \ell(t[x\backslash s]) = \ell(t)$$

The *outermost atomic label* of a term t, written $\uparrow(t)$ is defined as $\uparrow(\ell(t))$. For example:

$$\ell((\lambda^{\mathbf{abc}}x.x^{\mathbf{d}})[y\backslash y^{\mathbf{d}}]) = \mathbf{abc} \quad \text{and} \quad \uparrow((\lambda^{\mathbf{abc}}x.x^{\mathbf{d}})[y\backslash y^{\mathbf{d}}]) = \mathbf{a}$$

The syntax of contexts is extended to allow labeled terms, namely:

$$\mathtt{C} \;::=\; \Box \;\mid\; \lambda^\alpha x.\mathtt{C} \;\mid\; @^\alpha(\mathtt{C},t) \;\mid\; @^\alpha(t,\mathtt{C}) \;\mid\; \mathtt{C}[x\backslash t] \;\mid\; t[x\backslash\mathtt{C}]$$

and similarly for substitution contexts. An operation for *adding a label to a term*, written $\alpha : t$ is defined as follows by induction on t, by skipping substitutions:

$$\begin{array}{rcl@{\qquad}rcl} \alpha : x^\beta & \stackrel{\text{def}}{=} & x^{\alpha\beta} & \alpha : \lambda^\beta x.t & \stackrel{\text{def}}{=} & \lambda^{\alpha\beta}x.t \\ \alpha : @^\beta(t,s) & \stackrel{\text{def}}{=} & @^{\alpha\beta}(t,s) & \alpha : (t[x\backslash s]) & \stackrel{\text{def}}{=} & (\alpha : t)[x\backslash s] \end{array}$$

The *Lévy labeled LSC without* gc, LLSC^I, is defined as the rewriting system whose objects are the labeled terms $\mathcal{T}^{\ell I}$ and with the rewriting relation $\to_{\ell I}$ defined as the union of the following rules, closed under arbitrary contexts:

$$\begin{array}{rcl} @^\alpha((\lambda^\beta x.t)\mathtt{L}, s) & \mapsto_{\mathtt{db}} & \alpha[\mathtt{db}(\beta)] : t[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s]\mathtt{L} \\ \mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle[x\backslash t] & \mapsto_{\mathtt{ls}} & \mathtt{C}\langle\!\langle \alpha \bullet : t \rangle\!\rangle[x\backslash t] \end{array}$$

$$\to_{\ell I\mathtt{db}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathtt{db}} \rangle \quad \to_{\ell I\mathtt{ls}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathtt{ls}} \rangle \quad \to_{\ell I} \stackrel{\text{def}}{=} \to_{\ell I\mathtt{db}} \cup \to_{\ell I\mathtt{ls}}$$

Regarding the *names* of steps, the name of a db step like in the rule $\mapsto_{\mathtt{db}}$ is $\mathtt{db}(\beta)$, while the name of a ls step like in the rule $\mapsto_{\mathtt{ls}}$ is $\downarrow(\alpha) \bullet \uparrow(t)$. Sometimes we write $t \stackrel{\alpha}{\to}_{\ell I} s$ when $t \to_{\ell I} s$ and the name of the contracted redex is α.

In the following paragraphs we will try to understand the reasons motivating the design of the labeled system we have just defined. The main guiding principles are items 3. and 4. in the Bestiary of Section 6.2.1:

- **Copy:** residuals of a redex should have the same name as their ancestor.
- **Creation:** created redexes should include the name of all the redexes that contribute to their creation.

Forward propagation of labels

A term in a rewriting system has redexes, which represent possible *interactions* between parts of the term. When a redex R is fired in a Lévy labeled calculus, labels are propagated along the term according to precise rules. The informal idea is that labels on the left-hand side should propagate in such a way that the name of the redex R ends up "tainting" all those positions on the right-hand side in which there is a possibility of a new interaction due to the contraction of R. Let us give an informal account of how, and why, labels propagate in LLSC^I.

Suppose first that we fire a db redex R. The name of the db redex in a term like $@^\alpha((\lambda^\beta x.t)\mathtt{L}, s)$ is $\mathtt{db}(\beta)$. The informal idea is that β records the history indicating how the abstraction $\lambda^\beta x.t$ reached a position in the term in which it is able to interact with the application $@^\alpha(\ldots, s)$. Note that the list of substitutions L does not play any role regarding the interaction of the abstraction and the application: the substitutions are not able to aid nor interfere.

If we read the rewriting rule $\mapsto_{\mathtt{db}}$ of the LLSC^I forwards, we find out that the name $\mathtt{db}(\beta)$ of the contracted redex R is propagated to two places. First, the label $\mathtt{db}(\beta)$ is propagated to mark the root of the term t. This is because the root of the term t might be an abstraction, and firing R *exposes* the abstraction, leaving it on the root of the term, allowing it to, possibly, interact with an external application. This may allow a db **creates** db creation case, as in item 1. of Prop. 6.4. In order to comply with the **Creation** principle, the name of the fired db redex should be a sublabel of the created db redex. The following example illustrates how the name of the fired db redex ($\mathtt{db}(\mathbf{c})$) is indeed a sublabel of the created db redex ($\mathbf{b}\lceil\mathtt{db}(\mathbf{c})\rceil\mathbf{d}$):

$$\begin{array}{ll} @^{\mathbf{a}}(@^{\mathbf{b}}(\lambda^{\mathbf{c}}x.\lambda^{\mathbf{d}}y.x^{\mathbf{e}}, z^{\mathbf{f}}), z^{\mathbf{g}}) & \xrightarrow{\mathtt{db}(\mathbf{c})}_{\ell I} \quad @^{\mathbf{a}}((\lambda^{\mathbf{b}\lceil\mathtt{db}(\mathbf{c})\rceil\mathbf{d}}y.x^{\mathbf{e}})[x\backslash z^{\lfloor\mathtt{db}(\mathbf{c})\rfloor\mathbf{f}}], z^{\mathbf{g}}) \\ & \xrightarrow{\mathbf{b}\lceil\mathtt{db}(\mathbf{c})\rceil\mathbf{d}}_{\ell I} \quad x^{\mathbf{a}\lceil\mathtt{db}(\mathbf{b}\lceil\mathtt{db}(\mathbf{c})\rceil\mathbf{d})\rceil\mathbf{e}}[y\backslash z^{\lfloor\mathtt{db}(\mathbf{b}\lceil\mathtt{db}(\mathbf{c})\rceil\mathbf{d})\rfloor\mathbf{g}}][x\backslash z^{\mathbf{f}\lfloor\mathtt{db}(\mathbf{c})\rfloor}] \end{array}$$

On the other hand, the rule $\mapsto_{\mathtt{db}}$ of the LLSC^I also propagates the label $\mathtt{db}(\beta)$ to mark the argument of the substitution. This is because the argument s of the substitution might be replaced for an occurrence of x allowing a db **creates** ls situation as in item 2. of Prop. 6.4. As before, we would like to comply with the **Creation** principle. The following example illustrates how the name of the fired db redex ($\mathtt{db}(\mathbf{b})$) is indeed a sublabel of the created ls redex ($\mathbf{c} \bullet \lfloor\mathtt{db}(\mathbf{b})\rfloor$):

$$\begin{array}{ll} @^{\mathbf{a}}(\lambda^{\mathbf{b}}x.x^{\mathbf{c}}, y^{\mathbf{d}}) & \xrightarrow{\mathtt{db}(\mathbf{b})}_{\ell I} \quad x^{\mathbf{a}\lceil\mathtt{db}(\mathbf{b})\rceil\mathbf{c}}[x\backslash y^{\lfloor\mathtt{db}(\mathbf{b})\rfloor\mathbf{d}}] \\ & \xrightarrow{\mathbf{c}\bullet\lfloor\mathtt{db}(\mathbf{b})\rfloor}_{\ell I} \quad y^{\mathbf{a}\lceil\mathtt{db}(\mathbf{b})\rceil\mathbf{c}\bullet\lfloor\mathtt{db}(\mathbf{b})\rfloor\mathbf{d}} \end{array}$$

Note that in LSC the situation is subtler than in other calculi because the interaction between the variable x and the argument of the substitution is non-local, *i.e. at a distance*.

Consider now what happens if we fire a ls redex R. The name of the ls redex in a term like $\mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle[x\backslash t]$ is of the form $\downarrow(\alpha) \bullet \uparrow(t)$: it consists of two "halves". The first half ($\downarrow(\alpha)$) represents which copy of x we are contracting. For example, the two ls steps below should have different names and, indeed, their names are different: $\mathbf{a} \bullet \mathbf{c}$ and $\mathbf{b} \bullet \mathbf{c}$.

$$(x^{\mathbf{a}}x^{\mathbf{b}})[x\backslash y^{\mathbf{c}}] \begin{array}{ll} \xrightarrow{\mathbf{a}\bullet\mathbf{c}} & (y^{\mathbf{a}\bullet\mathbf{c}}x^{\mathbf{b}})[x\backslash y^{\mathbf{c}}] \\ \xrightarrow{\mathbf{b}\bullet\mathbf{c}} & (x^{\mathbf{a}}y^{\mathbf{b}\bullet\mathbf{c}})[x\backslash y^{\mathbf{c}}] \end{array}$$

The second half of the name $\uparrow(t)$ corresponds to the history of the argument t. Informally, the label $\uparrow(t)$ indicates how the term t came to be the argument of the substitution $[x\backslash\ldots]$.

By reading the rewriting rule $\mapsto_{\texttt{ls}}$ of the LLSC^I forwards, it can be seen that the name $\downarrow(\alpha)\bullet\uparrow(t)$ of the $\texttt{ls}$ redex appears on the root of the new copy of the term t. This is because the term t might be an abstraction, and firing the $\texttt{ls}$ redex R makes a copy of the abstraction. The new copy of t might possibly interact with an application, allowing a $\texttt{ls}$ **creates** $\texttt{db}$ creation case, as in items 4. and 5. of Prop. 6.4. As in the previous cases, we would like to comply with the **Creation** principle. The following example illustrates how the name of the fired $\texttt{ls}$ redex (**b** • **d**) is indeed a sublabel of the created $\texttt{db}$ redex ($\texttt{db}$(**b** • **d**)):

$$\begin{array}{rl}(@^{\mathbf{a}}(x^{\mathbf{b}},y^{\mathbf{c}}))[x\backslash\lambda^{\mathbf{d}}z.z^{\mathbf{e}}] & \xrightarrow{\mathbf{b}\bullet\mathbf{d}}_{\ell I}\ (@^{\mathbf{a}}(\lambda^{\mathbf{b}\bullet\mathbf{d}}z.z^{\mathbf{e}},y^{\mathbf{c}}))[x\backslash\lambda^{\mathbf{d}}z.z^{\mathbf{e}}]\\ & \xrightarrow{\texttt{db}(\mathbf{b}\bullet\mathbf{d})}_{\ell I}\ z^{\mathbf{a}\lceil\texttt{db}(\mathbf{b}\bullet\mathbf{d})\rceil\mathbf{e}}[z\backslash y^{\lfloor\texttt{db}(\mathbf{b}\bullet\mathbf{d})\rfloor\mathbf{c}}][x\backslash\lambda^{\mathbf{d}}z.z^{\mathbf{e}}]\end{array}$$

A point that should still be clarified is why the two "halves" of a $\texttt{ls}$ redex are atomic labels. That is, why the name of a $\texttt{ls}$ redex is $\downarrow(\alpha)\bullet\uparrow(t)$, rather than just $\alpha\bullet\ell(t)$. The reason is that we must comply with the **Copy** principle. To justify why we take $\downarrow(\alpha)$ rather than α for the first half, consider the following example:

$$\begin{array}{ccc}x^{\mathbf{a}}[x\backslash y^{\mathbf{b}}][y\backslash z^{\mathbf{c}}] & \xrightarrow{\mathbf{a}\bullet\mathbf{b}} & y^{\mathbf{a}\bullet\mathbf{b}}[x\backslash y^{\mathbf{b}}][y\backslash z^{\mathbf{c}}]\\ \downarrow{\scriptstyle \mathbf{b}\bullet\mathbf{c}} & & \downarrow{\scriptstyle \mathbf{b}\bullet\mathbf{c}}\\ x^{\mathbf{a}}[x\backslash z^{\mathbf{b}\bullet\mathbf{c}}][y\backslash z^{\mathbf{c}}] & & z^{\mathbf{a}\bullet\mathbf{b}\bullet\mathbf{c}}[x\backslash y^{\mathbf{b}}][y\backslash z^{\mathbf{c}}]\end{array}$$

The redex at the right-hand side of the diagram is one of the two residuals of the redex at the left-hand side, so by the **Copy** principle they should have the same name. If we were to take α, rather than $\downarrow(\alpha)$, for the first half of the name of a $\texttt{ls}$ redex, the name of the redex at the right-hand side would be **a** • **b** • **c** instead, violating the **Copy** principle.

To justify why we take $\uparrow(t)$ rather than $\ell(t)$ for the second half of the name of a $\texttt{ls}$-redex, consider the following (symmetric) example:

$$\begin{array}{ccc}x^{\mathbf{a}}[x\backslash y^{\mathbf{b}}][y\backslash z^{\mathbf{c}}] & \xrightarrow{\mathbf{b}\bullet\mathbf{c}} & x^{\mathbf{a}}[x\backslash z^{\mathbf{b}\bullet\mathbf{c}}][y\backslash z^{\mathbf{c}}]\\ \downarrow{\scriptstyle \mathbf{a}\bullet\mathbf{b}} & & \downarrow{\scriptstyle \mathbf{a}\bullet\mathbf{b}}\\ y^{\mathbf{a}\bullet\mathbf{b}}[x\backslash y^{\mathbf{b}}][y\backslash z^{\mathbf{c}}] & & z^{\mathbf{a}\bullet\mathbf{b}\bullet\mathbf{c}}[x\backslash z^{\mathbf{b}\bullet\mathbf{c}}][y\backslash z^{\mathbf{c}}]\end{array}$$

Here the redex at the right-hand side is the (unique) residual of the redex at the left-hand side, so by the **Copy** principle they should have the same name. If we were to take $\ell(t)$, rather than $\uparrow(t)$, for the second half of the name of a $\texttt{ls}$ redex, the name of the redex at the right-hand side would be **a** • **b** • **c** instead, violating the **Copy** principle.

The label constructors $\texttt{db}(-)$ and $\bullet$

An obvious difference between Lévy's labeled λ-calculus and the calculus LLSC^I that we have just proposed is the presence of labels of the form $\texttt{db}(\alpha)$ and the distinguished label ($\bullet$). The presence of these labels is not to be regarded as a fundamental feature of the labeled calculus, but rather as a technical convenience. Let us motivate their introduction.

The idea of labels is that they characterize *redex families*: two redexes should be assigned the same name if and only if they belong to the same family. According to the definition of LLSC^I, the name of a db redex in a term like $@^\alpha((\lambda^\beta x.t)\mathtt{L}, s)$ is $\mathtt{db}(\beta)$. For example, consider a situation like the following, corresponding to a `ls` **creates** db creation.

$$\begin{array}{rl}(@^{\mathbf{a}}(x^{\mathbf{b}}, y^{\mathbf{c}}))[x\backslash\lambda^{\mathbf{d}}z.z^{\mathbf{e}}] & \xrightarrow{\mathbf{b}\bullet\mathbf{d}}_{\ell I} \quad (@^{\mathbf{a}}(\lambda^{\mathbf{b}\bullet\mathbf{d}}z.z^{\mathbf{e}}, y^{\mathbf{c}}))[x\backslash\lambda^{\mathbf{d}}z.z^{\mathbf{e}}] \\ & \xrightarrow{\mathtt{db}(\mathbf{b}\bullet\mathbf{d})}_{\ell I} \quad z^{\mathbf{a}[\mathtt{db}(\mathbf{b}\bullet\mathbf{d})]\mathbf{e}}[z\backslash y^{\lfloor\mathtt{db}(\mathbf{b}\bullet\mathbf{d})\rfloor\mathbf{c}}][x\backslash\lambda^{\mathbf{d}}z.z^{\mathbf{e}}]\end{array}$$

Suppose that we had not introduced the $\mathtt{db}(-)$ constructor, and we had declared that the name of such a redex is just β. The name of the db redex above would then be just $\mathbf{b}\bullet\mathbf{d}$, and it would coincide with the name of the `ls` redex that contributed to its creation. This would contradict the principle that redexes in different families should have different names. Another reason to justify that the name of the `ls` redex should be *strictly* contained in the created db redex, is to comply with the **Termination** principle (item 7. in the Bestiary of Section 6.2.1). Recall that the **Termination** principle states that redexes of newer generations should have larger and larger names.

Regarding the distinguished initial label ($\bullet$), it is simply used as a marker to point out the places in which two labels have come into contact due to the contraction of a `ls` redex.

One may wonder if the addition of the constructors $\mathtt{db}(-)$ and $\bullet$ is strictly necessary to define a Lévy-like labeling for the LSC. It should be possible to formulate a variant of LLSC^I dispensing of both $\mathtt{db}(-)$ and $\bullet$ while essentially preserving all the good properties of the LLSC^I calculus, at the expense of treating redex names slightly more carefully. We have chosen to explicitly mark the places in which db and `ls` redexes take place, which simplifies the treatment of labels.

The distinguished label ($\bullet$) and associativity

As we have mentioned in the previous section, the distinguished label ($\bullet$) is used to point out the places in which two labels come into contact due to the contraction of a `ls` redex. We have chosen to make $\bullet$ an initial label, in such a way that $\mathbf{a}\bullet\mathbf{b}$ is a list of three labels: $\mathbf{a}$, $\bullet$, and $\mathbf{b}$. One may wonder if it would not be more appropriate to regard $\bullet$ as a binary constructor. To answer this question, consider the following example:

$$\begin{array}{ccc} x^{\mathbf{a}}[x\backslash y^{\mathbf{b}}][y\backslash z^{\mathbf{c}}] & \xrightarrow{\quad\mathbf{b}\bullet\mathbf{c}\quad} & x^{\mathbf{a}}[x\backslash z^{\mathbf{b}\bullet\mathbf{c}}][y\backslash z^{\mathbf{c}}] \\ \downarrow{\scriptstyle \mathbf{a}\bullet\mathbf{b}} & & \downarrow{\scriptstyle \mathbf{a}\bullet\mathbf{b}} \\ y^{\mathbf{a}\bullet\mathbf{b}}[x\backslash y^{\mathbf{b}}][y\backslash z^{\mathbf{c}}] & \xrightarrow{\mathbf{b}\bullet\mathbf{c}} z^{\mathbf{a}\bullet\mathbf{b}\bullet\mathbf{c}}[x\backslash y^{\mathbf{b}}][y\backslash z^{\mathbf{c}}] & \xrightarrow{\mathbf{b}\bullet\mathbf{c}} z^{\mathbf{a}\bullet\mathbf{b}\bullet\mathbf{c}}[x\backslash z^{\mathbf{b}\bullet\mathbf{c}}][y\backslash z^{\mathbf{c}}] \end{array}$$

Let t be the term at the bottom right of the diagram, *i.e.* $t = z^{\mathbf{a}\bullet\mathbf{b}\bullet\mathbf{c}}[x\backslash z^{\mathbf{b}\bullet\mathbf{c}}][y\backslash z^{\mathbf{c}}]$. Note that, if we contract y first and then x, the label decorating the leftmost copy z of t is $\mathbf{a}\bullet(\mathbf{b}\bullet\mathbf{c})$. On the other hand, if we contract x first and then the two copies of y, the corresponding label is $(\mathbf{a}\bullet\mathbf{b})\bullet\mathbf{c}$. The **Confluence** principle (item 6. in the Bestiary of Section 6.2.1) requires that the labeled calculus should be confluent. This basically means that the labels $\mathbf{a}\bullet(\mathbf{b}\bullet\mathbf{c})$ and $(\mathbf{a}\bullet\mathbf{b})\bullet\mathbf{c}$ should be considered equal. If we were to regard $\mathbf{a}\bullet\mathbf{b}$ as a new constructor, we

would have to work modulo associativity of $\bullet$, and also the associativity of $\bullet$ with respect to juxtaposition. That is, we would have to work modulo the following four equations:

$$\begin{array}{rcl} \alpha(\beta\gamma) & = & (\alpha\beta)\gamma \\ \alpha(\beta \bullet \gamma) & = & (\alpha\beta) \bullet \gamma \\ \alpha \bullet (\beta\gamma) & = & (\alpha \bullet \beta)\gamma \\ \alpha \bullet (\beta \bullet \gamma) & = & (\alpha \bullet \beta) \bullet \gamma \end{array}$$

Rather than doing this, we have chosen the arguably simpler route of working only modulo the first equation (associativity of juxtaposition), and regarding $\bullet$ as an initial label.

6.2.4 Definition of the Labeled LSC – Extension with gc

Definition 6.6 (The Lévy labeled LSC with gc, LLSC)**.** Similarly as in Def. 6.5, let $\mathcal{I} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots\}$ be a set of *initial labels*, and assume the existence of two distinguished labels $\bullet \in \mathcal{I}$ and $\otimes \in \mathcal{I}$. The set of labels $\mathcal{L}$ is again defined by the following grammar,

$$\alpha, \beta, \gamma, \ldots ::= \mathbf{a} \mid \alpha\beta \mid \lceil\alpha\rceil \mid \lfloor\alpha\rfloor \mid \mathtt{db}(\alpha)$$

Metavariables $\Omega, \Theta, \Psi, \ldots$ range over *finite sets* of initial labels. The set of *labeled terms* $\mathcal{T}^\ell$ is defined by the following grammar:

$$t, s, \ldots \quad ::= \quad x^\alpha \quad \mid \quad \lambda^\alpha_\Omega x.t \quad \mid \quad @^\alpha(t, s) \quad \mid \quad t[x\backslash s]_\Omega$$

The notions of *outermost atomic label* $\uparrow(\alpha)$ of a label α, *innermost atomic label* $\downarrow(\alpha)$ of a label α, *external label* $\ell(t)$ of a labeled term t, the operation of *adding a label to a term* $\alpha : t$, and the notions of *contexts* and *substitution contexts* are defined as in Def. 6.5.

The *Lévy labeled LSC with* gc, LLSC, is defined as the rewriting system whose objects are the labeled terms $\mathcal{T}^\ell$ and with the rewriting relation $\to_\ell$ defined as the union of the following rules, closed by arbitrary contexts.

$$\begin{array}{rcll} @^\alpha((\lambda^\beta_\Omega x.t)\mathtt{L}, s) & \mapsto_{\mathtt{db}} & \alpha\lceil\mathtt{db}(\beta)\rceil : t[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s]_\Omega\mathtt{L} & \\ \mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle[x\backslash t]_\Omega & \mapsto_{\mathtt{ls}} & \mathtt{C}\langle\!\langle \alpha \bullet \, : t \rangle\!\rangle[x\backslash t]_\Omega & \\ t[x\backslash s]_\Omega & \mapsto_{\mathtt{gc}} & t & \text{if } x \notin \mathsf{fv}(t) \end{array}$$

$$\to_{\ell\,\mathtt{db}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathtt{db}} \rangle \quad \to_{\ell\,\mathtt{ls}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathtt{ls}} \rangle \quad \to_{\ell\,\mathtt{gc}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathtt{gc}} \rangle \quad \to_\ell \stackrel{\text{def}}{=} \to_{\ell\,\mathtt{db}} \cup \to_{\ell\,\mathtt{ls}} \cup \to_{\ell\,\mathtt{gc}}$$

Regarding the *names* of steps, the name of a db step like in the rule $\mapsto_{\mathtt{db}}$ is $\mathtt{db}(\beta)$, the name of a ls step like in the rule $\mapsto_{\mathtt{ls}}$ is $\downarrow(\alpha) \bullet \uparrow(t)$, and the name of a gc step like in the rule $\mapsto_{\mathtt{gc}}$ is the *set of labels* $\{\mathbf{a} \bullet \uparrow(s) \mid \mathbf{a} \in \Omega\}$. As before, we write $t \xrightarrow{\alpha}_\ell s$ when $t \to_\ell s$ and the name of the contracted redex is α.

Observe that *redex names* $\mu, \nu, \xi, \ldots$ have three possibly shapes given by the grammar below, where α stands for an arbitrary label in $\mathcal{L}$, and ω, ω', etc. stand for atomic labels:

$$\mu ::= \underbrace{\mathtt{db}(\alpha)}_{\text{name of a db step}} \quad \mid \quad \underbrace{\omega \bullet \omega'}_{\text{name of a ls step}} \quad \mid \underbrace{\{\omega_1 \bullet \omega'_1, \ldots, \omega_n \bullet \omega'_n\}}_{\text{name of a gc step, } n \geqslant 1}$$

Usually we regard redex names as belonging to a separate sort, but occasionally we identify the names of db and ls steps with the corresponding underlying label—*e.g.* the redex name $\mathtt{db}(\mathbf{a})$ can also be thought as the label $\mathtt{db}(\mathbf{a})$.

The remainder of this subsection is devoted to motivating the definition of the labeling scheme introduced in Def. 6.6. We will also describe some difficulties involving the gc rule.

Motivation of the labeling: sets of labels (Ω) and dummy labels ($\otimes$)

Compared to the LLSC^I (without gc) the LLSC (with gc) incorporates three new elements:

1. Substitutions are decorated with a set of labels Ω.
2. Abstractions are decorated with a set of labels Ω.
3. There are gc steps, and the name of a gc step is a set of labels.

Observe that these elements do not interfere with the behavior of db and ls steps, and in particular it does not affect the names of db and ls steps. Let us motivate each of these new elements. Consider the following example reduction whose final step is a gc step:

$$(xx)[x\backslash y] \xrightarrow{R}_{\mathtt{ls}} (yx)[x\backslash y] \xrightarrow{S}_{\mathtt{ls}} (yy)[x\backslash y] \xrightarrow{T}_{\mathtt{gc}} yy$$

In this reduction sequence, the first two ls steps R and S contribute to the creation of the gc step T, which means that, in the labeled calculus, the names of R and S should be sublabels of the name of T. In fact, the order of the first two steps is irrelevant:

$$(xx)[x\backslash y] \xrightarrow{S'}_{\mathtt{ls}} (xy)[x\backslash y] \xrightarrow{R'}_{\mathtt{ls}} (yy)[x\backslash y] \xrightarrow{T}_{\mathtt{gc}} yy$$

The observation that the order is irrelevant is reflected in the fact that the name of the gc step is a *set* of labels.

Moreover, we note that the labeling scheme of the LLSC^I calculus, without gc, is not sufficient to give an appropriate name to the gc step. The main problem is that we lose track of the labels decorating each of the two copies of x, for example in this sequence of labeled steps in LLSC^I:

$$\begin{array}{lll} @^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{c}}))[x\backslash y^{\mathbf{d}}] & \xrightarrow{\mathbf{b}\bullet\mathbf{d}}_{\ell I\mathtt{ls}} & @^{\mathbf{a}}(y^{\mathbf{b}\bullet\mathbf{d}}, x^{\mathbf{c}})[x\backslash y^{\mathbf{d}}] \\ & \xrightarrow{\mathbf{c}\bullet\mathbf{d}}_{\ell I\mathtt{ls}} & @^{\mathbf{a}}(y^{\mathbf{b}\bullet\mathbf{d}}, y^{\mathbf{c}\bullet\mathbf{d}})[x\backslash y^{\mathbf{d}}] \end{array}$$

a hypothetical gc step of the form:

$$@^{\mathbf{a}}(y^{\mathbf{b}\bullet\mathbf{d}}, y^{\mathbf{c}\bullet\mathbf{d}})[x\backslash y^{\mathbf{d}}] \to_{\ell\,\mathtt{gc}} @^{\mathbf{a}}(y^{\mathbf{b}\bullet\mathbf{d}}, y^{\mathbf{c}\bullet\mathbf{d}})$$

should have a name including the labels $\mathbf{b}\bullet\mathbf{d}$ and $\mathbf{c}\bullet\mathbf{d}$. But, even though these labels do appear on the term (in this example), we have no way of knowing what relationship they have with the explicit substitution $[x\backslash y^{\mathbf{d}}]$.

The idea is that every substitution $t[x\backslash s]_\Omega$ should be decorated with a set of labels Ω, representing the initial labels that originally decorated each free occurrence of the variables x in t. For instance, in the LLSC the example above becomes:

$$\begin{array}{rll} @^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{c}}))[x\backslash y^{\mathbf{d}}]_{\{\mathbf{b},\mathbf{c}\}} & \xrightarrow{\mathbf{b}\bullet\mathbf{d}}_{\ell I \mathtt{ls}} & @^{\mathbf{a}}(y^{\mathbf{b}\bullet\mathbf{d}}, x^{\mathbf{c}})[x\backslash y^{\mathbf{d}}]_{\{\mathbf{b},\mathbf{c}\}} \\ & \xrightarrow{\mathbf{c}\bullet\mathbf{d}}_{\ell I \mathtt{ls}} & @^{\mathbf{a}}(y^{\mathbf{b}\bullet\mathbf{d}}, y^{\mathbf{c}\bullet\mathbf{d}})[x\backslash y^{\mathbf{d}}]_{\{\mathbf{b},\mathbf{c}\}} \end{array}$$

and as a consequence, the name of the gc step:

$$@^{\mathbf{a}}(y^{\mathbf{b}\bullet\mathbf{d}}, y^{\mathbf{c}\bullet\mathbf{d}})[x\backslash y^{\mathbf{d}}]_{\{\mathbf{b},\mathbf{c}\}} \to_{\ell\,\mathtt{gc}} @^{\mathbf{a}}(y^{\mathbf{b}\bullet\mathbf{d}}, y^{\mathbf{c}\bullet\mathbf{d}})$$

is, according to Def. 6.6, precisely the set $\{\mathbf{a}\bullet\uparrow(y^{\mathbf{d}}) \mid \mathbf{a}\in\{\mathbf{b},\mathbf{c}\}\}$ that is, $\{\mathbf{b}\bullet\mathbf{d}, \mathbf{c}\bullet\mathbf{d}\}$. Note that this set does not depend on the order in which the labeled variants of the steps R and S are fired, as can be seen in the diagram:

$$\begin{array}{ccc} @^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{c}}))[x\backslash y^{\mathbf{d}}]_{\{\mathbf{b},\mathbf{c}\}} & \xrightarrow{\mathbf{b}\bullet\mathbf{d}} & @^{\mathbf{a}}(y^{\mathbf{b}\bullet\mathbf{d}}, x^{\mathbf{c}}))[x\backslash y^{\mathbf{d}}]_{\{\mathbf{b},\mathbf{c}\}} \\ {\scriptstyle \mathbf{c}\bullet\mathbf{d}}\downarrow & & \downarrow{\scriptstyle \mathbf{c}\bullet\mathbf{d}} \\ @^{\mathbf{a}}(x^{\mathbf{b}}, y^{\mathbf{c}\bullet\mathbf{d}}))[x\backslash y^{\mathbf{d}}]_{\{\mathbf{b},\mathbf{c}\}} & \xrightarrow{\mathbf{b}\bullet\mathbf{d}} & @^{\mathbf{a}}(y^{\mathbf{b}\bullet\mathbf{d}}, y^{\mathbf{c}\bullet\mathbf{d}}))[x\backslash y^{\mathbf{d}}]_{\{\mathbf{b},\mathbf{c}\}} \end{array}$$

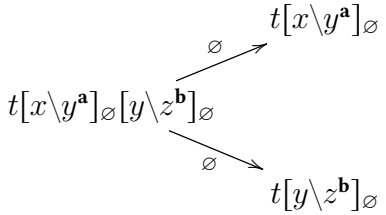
Later on, we will introduce an invariant characterizing *correctly labeled terms*. In a correctly labeled term, given any subterm $t[x\backslash s]_\Omega$ and any free occurrences of the form x^α in t, we will have that $\downarrow(\alpha)\in\Omega$.

To justify the presence of the decoration Ω over an abstraction $\lambda^\alpha_\Omega x.t$, note that a substitution $t[x\backslash s]_\Omega$ may be created as the result of firing a db step

$$@^\alpha((\lambda^\beta_\Omega x.t)\mathtt{L}, s) \to_{\ell\,\mathtt{db}}\ \alpha\lceil\mathtt{db}(\beta)\rceil : t[x\backslash\lfloor\mathtt{db}(\beta)\rfloor s]_\Omega\mathtt{L}$$

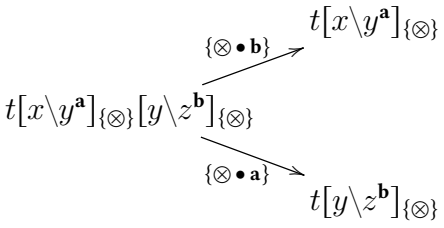
and the set Ω should appropriately record the set of initial labels originally decorating the free occurrences of x in t. This means that the invariant for correctly labeled terms should request that, given any subterm $\lambda^\alpha_\Omega x.t$ of a correctly labeled term for any free occurrences of the form x^α in t, we will have that $\downarrow(\alpha)\in\Omega$.

There is one more issue that we should mention: in a term of the form $\lambda^\alpha_\Omega x.t$ or of the form $t[x\backslash s]_\Omega$, the invariant for correctly labeled terms should not allow Ω to be the empty set. Note that if we allow Ω to be the empty set, the name of a gc step $t[x\backslash s]_\varnothing \to_{\ell\,\mathtt{gc}} t$ is the set of labels $\{\mathbf{a}\bullet\uparrow(t) \mid \mathbf{a}\in\varnothing\}$, that is, $\varnothing$. This is objectionable, because it may result in two gc steps that do not share the same origin but have the same name, for example, the name of the two steps below is $\varnothing$:



That is why the invariant for correctly labeled terms will forbid that Ω is empty. In the initial term, if t has no occurrences of x, we will decorate terms of the form $\lambda^\alpha_\Omega x.t$ and of the form

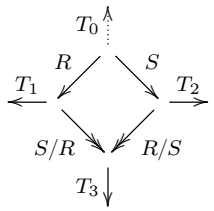
$t[x\backslash s]_\Omega$ with the set of labels $\Omega = \{\otimes\}$ where $\otimes$ is a distinguished *dummy* label. With this invariant, the names of the gc steps are different:



Failure of stability in the LSC with gc

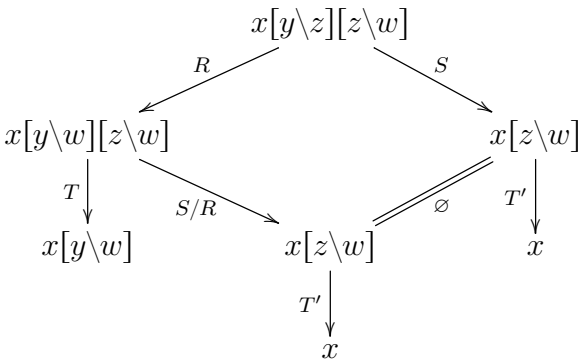
Stability is an abstract property of rewriting systems with residuals, stating that computational steps are created in an essentially unique way: if any two steps have a common residual, they must also have a common ancestor. This means that the presence of a computational step has a unique *cause*. The property of stability in the context of rewriting was originally studied by Jean-Jacques Lévy [109, 111], and inspired by Gérard Berry's notion of stability in denotational semantics [26].

Definition 6.7 (Stability)**.** An orthogonal axiomatic rewriting system (*cf.* Def. 2.39) verifies the *Stability* property if given steps R, S, T_1, T_2, T_3 such that $T_3 \in T_1/(S/R)$ and $T_3 \in T_2/(R/S)$, there exists a step T_0 such that $T_1 \in T_0/R$ and $T_2 \in T_0/S$. Graphically:



It is not difficult to see that the Stability property fails in the LSC, in presence of the gc rule.[2]

Remark 6.8 (Failure of Stability in the LSC). Consider the following diagram:

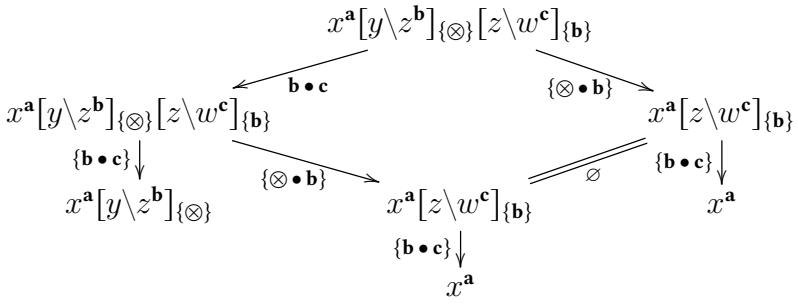


Note that R is a `ls` step and S, S/R, T, and T' are gc steps. Note also that $R/S = \varnothing$. Then T and T' do not have a common ancestor, which means that the LSC with gc does not have the Stability property.

[2]The Stability property does hold in the LSC without gc, as will be proved in Section 7.2.

The failure of Stability means that we cannot hope to fulfill all the principles of Lévy labels in the Bestiary of Section 6.2.1. In particular, if the **Initial** principle holds, we know that the names of R and S must be different. If the **Contribution** principle holds, we also know that the name of T should contain the name of R but not the name of S, while the name of T' should contain the name of S but not the name of R. Finally, from the **Copy** principle we conclude that the names of T and T' must coincide. From this we derive a contradiction.

In the labeled calculus LLSC of Def. 6.6, we have taken the design decision that the redex creation cases of the form "gc **creates** gc" (*cf.* Prop. 6.4) are not to be reflected in the labels. For instance, the example of Rem. 6.8 in the labeled calculus LLSC becomes:



Observe that the names of T and T' are both $\{\mathbf{b}\bullet\mathbf{c}\}$ and they include the name of R ($\mathbf{b}\bullet\mathbf{c}$) as a sublabel while, on the other hand, the name of S ($\{\otimes\bullet\mathbf{b}\}$) is unrelated with the name of T.

6.3 Properties of the LSC with Lévy Labels

This section is devoted to establishing various properties of the LLSC:

1. In Section 6.3.1 we prove basic properties of labeled reduction, including the invariant for correctly labeled terms.

2. In Section 6.3.2 we study permutation diagrams in the LLSC. In particular we prove that the LLSC is an orthogonal axiomatic rewriting system (Prop. 6.32).

3. In Section 6.3.3 we prove that the LLSC is *weakly* normalizing if the height of redex names is bounded (Prop. 6.45).

4. In Section 6.3.4 we build upon the previous result, and strengthen it to show that the LLSC is *strongly* normalizing if the height of redex names is bounded (Thm. 6.51).

5. In Section 6.3.5 we obtain as a corollary of previous results that the LLSC is confluent (Thm. 6.53).

Among these properties, we prove that the LLSC enjoys most of the desirable traits that we already listed for a Lévy labeled calculus in the Bestiary of Section 6.2.1. We summarize the status of each of these properties after finishing this section:

1. The **Lift** property is an easy observation. Given any (unlabeled) reduction sequence $\rho : t \twoheadrightarrow_{\mathsf{LSC}} s$ and any labeled variant $t' \in \mathcal{T}^{\ell}$ of t, there is a labeled variant $\rho' : t' \twoheadrightarrow_{\ell} s'$

of ρ. Moreover, in Lem. 6.23 we prove that all the labeled terms along the reduction ρ' preserve the invariant of being correctly labeled, provided that t' is correctly labeled.

2. The **Initial** property is proved in Lem. 6.19.
3. The **Copy** property is proved in Lem. 6.33.
4. The **Creation** property for the calculus *without* gc is proved in Prop. 6.41. We also show an example in which this property does not hold for the calculus with gc (Rem. 6.8).
5. The **Contribution** property for the calculus without gc is not proved in this section. We prove it in the next chapter (Prop. 7.12), when we show that the LSC without gc forms a Deterministic Family Structure (Thm. 7.13).
6. In Thm. 6.53 we give two alternative proofs of **Confluence**.
7. The **Termination** property is established in Thm. 6.51.
8. As we mentioned, we do not treat the **Reconstruction** or **Paths** properties.

6.3.1 Basic Properties

We begin by proving some basic properties of the labeled calculus LLSC as defined in Def. 6.6, including the invariant for *correctly labeled terms*.

Labels and contexts

Lemma 6.9 (Properties of labels and contexts)**.** *Operations on labels and contexts have the following properties:*

1. $\alpha : (\beta : t) = (\alpha\beta) : t$
2. *If* $\mathtt{L}$ *is a substitution context, then* $\alpha : (t\mathtt{L}) = (\alpha : t)\mathtt{L}$.
3. *If* $\mathtt{C}$ *is not a substitution context, then* $\alpha : \mathtt{C}\langle t\rangle = (\alpha : \mathtt{C})\langle t\rangle$.
4. $\uparrow (\alpha : t) = \uparrow (\alpha)$
5. $\uparrow (\mathtt{C}\langle\langle x^{\alpha}\rangle\rangle) = \uparrow (\mathtt{C}\langle\alpha \bullet : t\rangle)$
6. $\alpha : \mathtt{C}\langle\langle x^{\beta}\rangle\rangle$ *is of the form* $\mathtt{C}'\langle\langle x^{\beta'}\rangle\rangle$, *where* $\downarrow (\beta) = \downarrow (\beta')$ *and* $\alpha : \mathtt{C}\langle\beta \bullet : t\rangle = \mathtt{C}'\langle\beta' \bullet : t\rangle$.

Proof. The proofs are straightforward. Item 1. and 4. are by induction on t. Items 2., 3., and 5. are by induction on the context. Item 6. is easy by case analysis, depending on whether $\mathtt{C}$ is a substitution context or not, and using items 2. and 3. respectively. □

Lemma 6.10 (Adding labels is functorial)**.** *If* $t \xrightarrow{\mu}_{\ell} s$ *then* $(\alpha : t) \xrightarrow{\mu}_{\ell} (\alpha : s)$.

Proof. By induction on the context $\mathtt{C}$ under which the redex in t is contracted. The interesting case is the base case, when $\mathtt{C} = \Box$. Then we proceed by case analysis, depending on the kind of redex contracted.

1. `db` *step.*
$$\begin{array}{ccc} \alpha : @^{\beta}((\lambda^{\gamma}_{\Omega} x.t')\mathtt{L}, s') & \qquad & \alpha : \beta\lceil \mathtt{db}(\gamma)\rceil : t'[x\backslash\lfloor \mathtt{db}(\gamma)\rfloor : s']_{\Omega}\mathtt{L} \\ \| & & \| \\ @^{\alpha\beta}((\lambda^{\gamma}_{\Omega} x.t')\mathtt{L}, s') & \xrightarrow{\mathtt{db}(\gamma)} & \alpha\beta\lceil \mathtt{db}(\gamma)\rceil : t'[x\backslash\lfloor \mathtt{db}(\gamma)\rfloor : s']_{\Omega}\mathtt{L} \end{array}$$

2. `ls` *step.* Using Lem. 6.9, we have:
$$\begin{array}{ccc} \alpha : \mathtt{C}'\langle x^{\beta}\rangle[x\backslash t']_{\Omega} & \qquad & \alpha : \mathtt{C}'\langle \beta \bullet \ : t'\rangle[x\backslash t']_{\Omega} \\ \| & & \| \\ \mathtt{C}''\langle x^{\beta'}\rangle[x\backslash t']_{\Omega} & \xrightarrow{\downarrow(\beta)\bullet\uparrow(t')} & \mathtt{C}''\langle \beta' \bullet \ : t'\rangle[x\backslash t']_{\Omega} \end{array}$$

3. `gc` *step.*
$$\alpha : t'[x\backslash s']_{\Omega} \xrightarrow{\{\mathbf{a}\bullet\uparrow(s') \ \mid \ \mathbf{a}\in\Omega\}} \alpha : t'$$

□

Lemma 6.11 (Reduction preserves the outermost label)**.** *If* $t \to_{\ell} s$ *then* $\uparrow(t) = \uparrow(s)$.

Proof. By induction on the context $\mathtt{C}$ under which the redex in t is contracted. The interesting case is the base case, when $\mathtt{C} = \Box$. Then we proceed by case analysis, depending on the kind of redex contracted.

1. `db` *step.* Then:
$$\begin{array}{rcll} \uparrow(@^{\alpha}((\lambda^{\beta}_{\Omega} x.t')\mathtt{L}, s')) & = & \uparrow(\alpha) & \\ & = & \uparrow(\alpha\lceil \mathtt{db}(\beta)\rceil) & \\ & = & \uparrow(\alpha\lceil \mathtt{db}(\beta)\rceil : t'[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : s']_{\Omega}\mathtt{L}) & \text{by Lem. 6.9} \end{array}$$

2. `ls` *step.* Then $\uparrow(\mathtt{C}'\langle\!\langle x^{\alpha}\rangle\!\rangle) = \uparrow(\mathtt{C}'\langle\alpha \bullet \ : t'\rangle)$ by Lem. 6.9.

3. `gc` *step.* Then $\uparrow(t'[x\backslash s']_{\Omega}) = \uparrow(t')$.

□

Initially and correctly labeled terms

Recall that the **Initial** principle in the Bestiary of properties of Lévy labels given in Section 6.2.1 requests that, in an *initially* labeled term, two different redexes should have different names. As a consequence, if we have an unlabeled term and we want to decorate it with initial labels, each subterm (except for substitutions) should be decorated with a different initial label. For example, xx should be initially labeled as $@^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{c}})$ or $@^{\mathbf{c}}(x^{\mathbf{a}}, x^{\mathbf{b}})$ rather than $@^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{b}})$ or $@^{\mathbf{a}}(x^{\mathbf{a}}, x^{\mathbf{a}})$.

Moreover, *binders* in the LLSC, that is, abstractions and explicit substitutions, are decorated with a set of labels Ω. As we have discussed in Section 6.2.4, the set Ω associated to

a subterm binding a variable x should start being the set of initial labels decorating the free occurrences of x, or $\{\otimes\}$ if there are no free occurrences of x. For example, $\lambda^{\mathbf{a}}_{\{\mathbf{c},\mathbf{d}\}}x.@^{\mathbf{b}}(x^{\mathbf{c}}, x^{\mathbf{d}})$ and $@^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{c}})[y\backslash z^{\mathbf{d}}]_{\{\otimes\}}$ are initially labeled terms.

The property that a term is initially labeled is very restrictive, and the rewriting relation $(\to_\ell)$ of the LLSC does not preserve the invariant that a term is initially labeled. For example, in the following `ls` step:

$$x^{\mathbf{a}}[x\backslash\lambda^{\mathbf{b}}_{\{\mathbf{c}\}}y.y^{\mathbf{c}}]_{\{\mathbf{a}\}} \to_{\ell\,\mathtt{ls}} (\lambda^{\mathbf{a}\bullet\mathbf{b}}_{\{\mathbf{c}\}}y.y^{\mathbf{c}})[x\backslash\lambda^{\mathbf{b}}_{\{\mathbf{c}\}}y.y^{\mathbf{c}}]_{\{\mathbf{a}\}}$$

the left-hand side is initially labeled, while on the right-hand side: (1) there is a subterm decorated with a label which is not an initial label ($\mathbf{a}\bullet\mathbf{b}$), (2) there are two subterms decorated with the same initial label (the two copies of $y^{\mathbf{c}}$), and (3) the set of labels $\{\mathbf{a}\}$ on the explicit substitution does not correspond to a free occurrence of x.

In this section, we give a precise definition of *initially labeled terms*, and we define the invariant for *correctly labeled terms*, in such a way that all initially labeled terms are correctly labeled, and the rewriting relation $(\to_{\mathsf{LSC}})$ preserves correctly labeled terms.

Definition 6.12 (Leaf labels)**.** Let $t \in \mathcal{T}^\ell$. The multiset of *leaf labels* of t, written $\mathsf{vl}_x(t)$, is the multiset of atomic labels of the form $\downarrow(\alpha)$ for each free occurrence of x^α in t. Formally:

$$\begin{array}{rcll}
\mathsf{vl}_x(x^\alpha) & \stackrel{\text{def}}{=} & \{\downarrow(\alpha)\} & \\
\mathsf{vl}_x(y^\alpha) & \stackrel{\text{def}}{=} & \varnothing & \text{if } x \neq y \\
\mathsf{vl}_x(\lambda^\alpha_\Omega y.t) & \stackrel{\text{def}}{=} & \mathsf{vl}_x(t) & \text{if } x \neq y \\
\mathsf{vl}_x(@^\alpha(t,s)) & \stackrel{\text{def}}{=} & \mathsf{vl}_x(t) \uplus \mathsf{vl}_x(s) & \\
\mathsf{vl}_x(t[y\backslash s]_\Omega) & \stackrel{\text{def}}{=} & \mathsf{vl}_x(t) \uplus \mathsf{vl}_x(s) & \text{if } x \neq y
\end{array}$$

We also extend this operation to contexts by defining $\mathsf{vl}_x(\Box) \stackrel{\text{def}}{=} \varnothing$. Note that $\mathsf{vl}_x(\mathtt{C}\langle t\rangle) = \mathsf{vl}_x(\mathtt{C}) \uplus \mathsf{vl}_x(t)$ if $\mathtt{C}$ does not bind x. Occasionally we treat multisets as sets when the multiplicity of labels is not relevant.

Lemma 6.13 (Properties of leaf labels)**.** *Leaf labels have the following properties:*

1. $\mathsf{vl}_x(\alpha : t) = \mathsf{vl}_x(t)$
2. *If* $t \to_\ell s$ *then* $\mathsf{vl}_x(t) \supseteq \mathsf{vl}_x(s)$ *for any variable* x *(where "$\supseteq$" denotes the inclusion of* sets*).*

Proof. Item 1. is straightforward by induction on t. Item 2. is by induction on t. The interesting case is when we have a step:

1. `db` *step.* $@^\alpha((\lambda^\beta_\Omega y.t')\mathtt{L}, t_2) \xrightarrow{\mathtt{db}(\beta)}_{\ell\,\mathtt{db}} \alpha\lceil\mathtt{db}(\beta)\rceil : t'[y\backslash\lfloor\mathtt{db}(\beta)\rfloor : t_2]_\Omega\mathtt{L}$. Then:

$$\begin{array}{rcll}
\mathsf{vl}_x(@^\alpha((\lambda^\beta_\Omega y.t')\mathtt{L}, t_2)) & = & \mathsf{vl}_x(t') \uplus \mathsf{vl}_x(t_2) \uplus \mathsf{vl}_x(\mathtt{L}) & \\
& = & \mathsf{vl}_x(\alpha\lceil\mathtt{db}(\beta)\rceil : t') \uplus \mathsf{vl}_x(\lfloor\mathtt{db}(\beta)\rfloor : t_2) \uplus \mathsf{vl}_x(\mathtt{L}) & \text{by item 1.} \\
& = & \mathsf{vl}_x(\alpha\lceil\mathtt{db}(\beta)\rceil : t'[y\backslash\lfloor\mathtt{db}(\beta)\rfloor : t_2]_\Omega\mathtt{L}) &
\end{array}$$

2. `ls` *step.* $\mathtt{C}\langle\!\langle y^{\alpha}\rangle\!\rangle[y\backslash t_2]_{\Omega} \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t_2)}_{\ell\,\mathtt{ls}} \mathtt{C}\langle\alpha\bullet : t_2\rangle[y\backslash t_2]_{\Omega}$. Then:

$$\begin{array}{rcll} \mathsf{vl}_x(\mathtt{C}\langle\!\langle y^{\alpha}\rangle\!\rangle[y\backslash t_2]_{\Omega}) & = & \mathsf{vl}_x(\mathtt{C}) \uplus \mathsf{vl}_x(t_2) & \\ & = & \mathsf{vl}_x(\mathtt{C}) \uplus \mathsf{vl}_x(t_2) \uplus \mathsf{vl}_x(t_2) & \text{(set equality)} \\ & = & \mathsf{vl}_x(\mathtt{C}) \uplus \mathsf{vl}_x(\alpha\bullet : t_2) \uplus \mathsf{vl}_x(t_2) & \text{by item. 1} \\ & = & \mathsf{vl}_x(\mathtt{C}\langle\alpha\bullet : t_2\rangle[y\backslash t_2]_{\Omega}) & \end{array}$$

3. `gc` *step.* $t_1[y\backslash t_2]_{\Omega} \xrightarrow{\{\mathbf{a}\bullet\uparrow(t_2)\ |\ \mathbf{a}\in\Omega\}}_{\ell\,\mathtt{gc}} t_1$. Then $\mathsf{vl}_x(t_1[y\backslash t_2]_{\Omega}) = \mathsf{vl}_x(t_1) \uplus \mathsf{vl}_x(t_2) \supseteq \mathsf{vl}_x(t_1)$.

□

Definition 6.14 (Initially labeled terms)**.** A term $t \in \mathcal{T}^{\ell}$ is *initially labeled*, written $\textsc{init}(t)$, if:

1. For every subterm of s, the external label $\ell(s)$ is initial and $\ell(s) \notin \{\bullet, \otimes\}$.
2. For every pair of subterms s_1, s_2 at different positions, $\ell(s_1) \neq \ell(s_2)$.
3. For every subterm of t that is a binder, *i.e.* of the form $(\lambda^{\mathbf{a}}_{\Omega} x.s)$, or of the form $s[x\backslash u]_{\Omega}$, we have $\Omega = \begin{cases} \{\otimes\} & \text{if } \mathsf{vl}_x(s) = \varnothing \\ \mathsf{vl}_x(s) & \text{otherwise} \end{cases}$

Remark 6.15. Given an unlabeled term t, there always exists an initially labeled variant t^{ℓ} of t.

Example 6.16 (Initially labeled terms)**.** *The labeled term* $(\lambda^{\mathbf{a}}_{\{\mathbf{c}\}} x.@^{\mathbf{b}}(x^{\mathbf{c}}, y^{\mathbf{d}}))[y\backslash z^{\mathbf{e}}]_{\{\mathbf{e}\}}$ *is an initially labeled variant of* $(\lambda x.xy)[y\backslash z]$.

The labeled terms $\lambda^{\mathbf{a}}_{\{\mathbf{c},\mathbf{d}\}} x.@^{\mathbf{b}}(x^{\mathbf{c}}, x^{\mathbf{d}})$ *and* $y^{\mathbf{a}}[x\backslash z^{\mathbf{b}}]_{\{\otimes\}}$ *are initially labeled.*

The labeled terms $x^{\bullet}$ *and* $x^{\otimes}$ *are not initially labeled because the distinguished initial labels* $\bullet$ *and* $\otimes$ *cannot decorate subterms.*

The labeled terms $@^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{b}})$ *and* $@^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{a}})$ *are not initially labeled because different subterms should have different labels.*

The labeled terms $\lambda^{\mathbf{a}}_{\{\mathbf{b},\mathbf{c}\}} x.x^{\mathbf{b}}$ *and* $y^{\mathbf{a}}[x\backslash z^{\mathbf{b}}]_{\{\otimes\}}$ *are not initially labeled because sets of labels over binders should coincide with the set of leaf labels of the bound variable.*

The following easy lemma (Lem. 6.18) states that the names of the steps in an initially labeled term have a very particular shape.

Definition 6.17 (Initial redex names)**.** A redex name μ is said to be *initial* according to the following definition by cases:

1. A `db` redex name is initial if it is of the form $\mathtt{db}(\mathbf{a})$ with $\mathbf{a} \in \mathcal{I}$.
2. A `ls` redex name is initial if it is of the form $\mathbf{a}\bullet\mathbf{b}$ with $\mathbf{a}, \mathbf{b} \in \mathcal{I}$.
3. A `gc` redex name is initial if it is of the form $\{\otimes\bullet\mathbf{a}\}$.

Lemma 6.18 (Redexes in initially labeled terms are initial)**.** *Let* $t \in \mathcal{T}^{\ell}$ *be initially labeled. Let* μ *be the name of some redex in* t*. Then* μ *is initial.*

Proof. By cases.

1. `db` *redex.* $t = @^{\mathbf{a}}((\lambda^{\mathbf{b}}_{\Omega} x.t')\mathtt{L}, s')$; name: $\mathtt{db}(\mathbf{b})$.
2. `ls` *redex.* $t = \mathtt{C}\langle\!\langle x^{\mathbf{a}} \rangle\!\rangle [x \backslash t']_{\Omega}$; name: $\mathbf{a} \bullet \uparrow (t')$. Note that $\uparrow (t') \in \mathcal{I}$ given that t' is an initially labeled term.
3. `gc` *redex.* $t = t'[x \backslash s']_{\Omega}$; name: $\{\mathbf{a} \bullet \uparrow (s') \mid \mathbf{a} \in \Omega\}$. As before, $\uparrow (s') \in \mathcal{I}$ since s' is an initially labeled term. Moreover, since a `gc` step applies, $\mathsf{vl}_x(t') = \varnothing$, hence $\Omega = \{\otimes\}$.

□

The following lemma proves the **Initial** property from the Bestiary of Section 6.2.1:

Lemma 6.19 (Initial property**).** *Let $t \in \mathcal{T}^{\ell}$ be initially labeled. If $R : t \xrightarrow{\mu}_{\ell} s$ and $S : t \xrightarrow{\nu}_{\ell} u$ are different steps, then $\mu \neq \nu$.*

Proof. If R is a `db` step, its name is of the form $\mu = \mathtt{db}(\mathbf{a})$, where $\mathbf{a}$ is the label decorating the λ. Suppose that $\nu = \mathtt{db}(\mathbf{a})$. Then S is a `db` step contracting the same λ, hence $R = S$.

If R is a `ls` step, its name is of the form $\mu = \mathbf{a} \bullet \mathbf{b}$, where $\mathbf{a}$ is the label decorating the contracted variable. Suppose that $\nu = \mathbf{a} \bullet \mathbf{b}$. Then S is a `ls` step contracting the same variable, hence $R = S$.

If R is a `gc` step, its name is of the form $\mu = \{\mathbf{a} \bullet \uparrow (\mathbf{a}) \mid \mathbf{a} \in \Omega\}$, where $\mathbf{a}$ is the external label of the term s in the substitution $t[x \backslash s]_{\Omega}$ erased by R. Suppose that $\nu = \{\mathbf{a} \bullet \uparrow (\mathbf{a}) \mid \mathbf{a} \in \Omega\}$. Then S is a `gc` step erasing the same substitution, hence $R = S$. □

Next we define the notion of *correctly labeled terms*. To do so, we also define an auxiliary predicate that states whether a term is *good*. Roughly speaking, a term is *good* if the distinguished initial label $\bullet$ only appears as a result of applying the $\rightarrow_{\ell\,\mathtt{ls}}$ rule, and the distinguished initial label $\otimes$ only appears decorating the sets Ω of the binders $\lambda^{\alpha}_{\Omega} x.t$ and $t[x \backslash s]_{\Omega}$ when $x \notin \mathsf{fv}(t)$.

Definition 6.20 (Correctly labeled terms)**.** A label $\alpha \in \mathcal{L}$ is *good*, written $\checkmark(\alpha)$ if it verifies the following inductive definition:

$$\begin{array}{rcl}
\mathbf{a} \notin \{\bullet, \otimes\} & \Longrightarrow & \checkmark(\mathbf{a}) \\
\checkmark(\alpha) \wedge \checkmark(\beta) & \Longrightarrow & \checkmark(\alpha\,\beta) \\
\checkmark(\alpha) \wedge \checkmark(\beta) & \Longrightarrow & \checkmark(\alpha \bullet \beta) \\
\checkmark(\alpha) & \Longrightarrow & \checkmark(\lceil \alpha \rceil) \\
\checkmark(\alpha) & \Longrightarrow & \checkmark(\lfloor \alpha \rfloor) \\
\checkmark(\alpha) & \Longrightarrow & \checkmark(\mathtt{db}(\alpha))
\end{array}$$

A set of initial variables Ω is good, written $\checkmark(\Omega)$, if it is non-empty, it contains no occurrences of $\bullet$, and it does not contain occurrences of $\otimes$ unless it is precisely $\{\otimes\}$. Formally:

$$\checkmark(\Omega) \quad \overset{\text{def}}{\Longleftrightarrow} \quad (\Omega \neq \varnothing) \wedge (\bullet \notin \Omega) \wedge (\otimes \notin \Omega \vee \Omega = \{\otimes\})$$

A term $t \in \mathcal{T}^\ell$ is good, written $\checkmark(t)$, if every label and set of labels is good. More precisely:

$$\begin{array}{lcl} \checkmark(\alpha) & \Longrightarrow & \checkmark(x^\alpha) \\ \checkmark(\alpha) \wedge \checkmark(\Omega) \wedge \checkmark(t) & \Longrightarrow & \checkmark(\lambda^\alpha_\Omega x.t) \\ \checkmark(\alpha) \wedge \checkmark(t) \wedge \checkmark(s) & \Longrightarrow & \checkmark(@^\alpha(t,s)) \\ \checkmark(t) \wedge \checkmark(s) \wedge \checkmark(\Omega) & \Longrightarrow & \checkmark(t[x\backslash s]_\Omega) \end{array}$$

We also extend the notion of *goodness* to contexts, by declaring $\checkmark(\Box)$ to hold. Note that $\checkmark(\mathtt{C}\langle t\rangle)$ holds if and only if $\checkmark(\mathtt{C})$ and $\checkmark(t)$ hold.

A term $t \in \mathcal{T}^\ell$ is said to be *correctly labeled* if and only if all of the following conditions hold:

1. *Good:* $\checkmark(t)$ holds.
2. *Correct abstractions:* for any subterm $\lambda^\alpha_\Omega x.t'$ we have $\mathsf{vl}_x(t') \subseteq \Omega$.
3. *Correct substitutions:* for any subterm $t'[x\backslash s']_\Omega$ we have $\mathsf{vl}_x(t') \subseteq \Omega$.

For points 2, and 3, note that the inclusions are set-theoretical, *i.e.* we only care about the underlying set of the multiset $\mathsf{vl}_x(t)$.

Example 6.21 (Correctly labeled terms)**.** *The labeled term* $\lambda^{\mathbf{a}\bullet\mathbf{b}}_{\{\lceil \mathtt{db}(\mathbf{d})\rceil, \lfloor \mathtt{db}(\mathbf{e})\rfloor\}} x.x^{\mathbf{c}\lceil \mathtt{db}(\mathbf{d})\rceil}$ *is a correctly labeled variant of* $\lambda x.x$*.*

The labeled terms $x^{\mathbf{a}\bullet}$ *and* $y^{\mathbf{a}\otimes\mathbf{b}}$ *are not correctly labeled because* $\mathbf{a}\bullet$ *and* $\mathbf{a}\otimes\mathbf{b}$ *are not good. The labeled term* $\lambda^{\mathbf{a}}_{\{\otimes\}} x.x^{\mathbf{bcd}}$ *is not correctly labeled because* $\{\mathbf{d}\}$ *is not a subset of* $\{\otimes\}$*.*

The definition of initially and correctly labeled terms is also extended to derivations. A derivation $\rho : t \twoheadrightarrow_\ell s$ is said to be initially (resp. correctly) labeled if t is initially (resp. correctly) labeled. By Rem. 6.15, any derivation $\rho : t \twoheadrightarrow_{\mathsf{LSC}} s$ in the unlabeled LSC has an initially labeled variant $\rho' : t' \twoheadrightarrow_\ell s'$.

Next we show that the notion of correctly labeled terms is indeed invariant by the rewriting relation $(\to_\ell)$.

Remark 6.22 (Initially labeled terms are correctly labeled). Initially labeled terms are correctly labeled.

Lemma 6.23 (Reduction preserves correctly labeled terms)**.** *Let* $t \in \mathcal{T}^\ell$ *be a correctly labeled term and* $t \to_\ell s$*. Then* s *is correctly labeled.*

Proof. In the proof of this lemma we use the fact that if $\checkmark(t)$ and $\checkmark(\alpha)$ then $\checkmark(\alpha : t)$ and $\checkmark(\alpha\bullet : t)$, which can be easily proved by induction on t. The proof proceeds by induction on t. The interesting cases are when there is a step at the root of the term:

1. $\mathtt{db}$ *step.* $@^\alpha((\lambda^\beta_\Omega x.t')\mathtt{L}, t_2) \xrightarrow{\mathtt{db}(\beta)}_{\ell\,\mathtt{db}} \alpha\lceil \mathtt{db}(\beta)\rceil : t'[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : t_2]_\Omega\mathtt{L}$.
   1.1 *Good:* since the invariant holds for t we have: $\checkmark(\alpha)$, $\checkmark(\beta)$, $\checkmark(\Omega)$, $\checkmark(t')$, $\checkmark(t_2)$, and $\checkmark(\mathtt{L})$ which implies $\checkmark(\alpha\lceil \mathtt{db}(\beta)\rceil)$, $\checkmark(\lfloor \mathtt{db}(\beta)\rfloor)$. Moreover, $\checkmark(\alpha\lceil \mathtt{db}(\beta)\rceil : t'[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : t_2]_\Omega\mathtt{L})$.

1.2 *Correct abstractions:* immediate by the invariant on t.

1.3 *Correct substitutions:* for substitution nodes in t' and t_2, it is immediate by the invariant on t. For substitution nodes in $\mathtt{L}$, it is also immediate by using Lem. 6.13, since $\mathtt{L}$ cannot bind any variable in t_2. Finally, for the substitution $[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : t_2]_\Omega$ we use the fact that t has correct abstractions and Lem. 6.13 to conclude that $\mathsf{vl}_x(\alpha\lceil \mathtt{db}(\beta)\rceil : t') = \mathsf{vl}_x(t') \subseteq \Omega$.

2. $\mathtt{ls}$ *step.* $\mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle[x\backslash t_2]_\Omega \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t_2)}_{\ell\ \mathtt{ls}} \mathtt{C}\langle\alpha\bullet : t_2\rangle[x\backslash t_2]_\alpha$.

2.1 *Good:* by the invariant on t we have that $\checkmark(\mathtt{C})$, $\checkmark(\alpha)$, $\checkmark(t_2)$, and $\checkmark(\Omega)$. So $\checkmark(\mathtt{C}\langle\alpha\bullet : t_2\rangle[x\backslash t_2]_\Omega)$.

2.2 *Correct abstractions:* abstractions internal to t_2 or internal to $\mathtt{C}$ are correct by the invariant on t. The only non-trivial case is that of abstraction nodes in the path to the hole of $\mathtt{C}$. Let $\mathtt{C}$ be of the form $\mathtt{C}_1\langle\lambda^\beta_\Theta y.\mathtt{C}_2\rangle$. Then $\mathsf{vl}_y(\mathtt{C}_2\langle\!\langle x^\alpha\rangle\!\rangle) = \mathsf{vl}_y(\mathtt{C}_2\langle\alpha : t_2\rangle)$ since $x \neq y$, and t_2 cannot have free occurrences of y. We conclude by the fact that $\mathsf{vl}_y(\mathtt{C}_2\langle x^\alpha\rangle) \subseteq \Theta$, as the invariant holds for t.

2.3 *Correct substitutions:* the only non-trivial case is for substitutions lying in the path to the hole of $\mathtt{C}$. Let $\mathtt{C}$ be of the form $\mathtt{C}_1\langle\mathtt{C}_2[y\backslash s]_\Theta\rangle$. Then $\mathsf{vl}_y(\mathtt{C}_2\langle\!\langle x^\alpha\rangle\!\rangle) = \mathsf{vl}_y(\mathtt{C}_2\langle\alpha : t_2\rangle)$. We conclude similarly as in the previous item.

3. $\mathtt{gc}$ *step.* $t_1[x\backslash t_2]_\Omega \xrightarrow{\{\mathbf{a}\bullet\uparrow(t_2)\ \mid\ \mathbf{a}\in\Omega\}}_{\ell\ \mathtt{gc}} t_1$ with $x \notin \mathsf{fv}(t_1)$. This case it is immediate by the fact that the invariant holds for t.

□

Definition 6.24 (Initially reachable terms)**.** A term t is said to be *initially reachable* if there exists an initially labeled term t_0 such that $t_0 \twoheadrightarrow_\ell t$.

Remark 6.25 (Initially reachable terms are correctly labeled)*.* Initially reachable terms are correctly labeled by the fact that reduction preserves correct labelling (Lem. 6.23).

Labeling morphisms

Sometimes it is useful to *rename* labels. For example, the following derivation in the LLSC:

$$(x^{\mathbf{a}}\, x^{\mathbf{b}})[x\backslash z^{\mathbf{c}}]_{\{\mathbf{a},\mathbf{b}\}} \xrightarrow{\mathbf{a}\bullet\mathbf{c}}_\ell (z^{\mathbf{a}\bullet\mathbf{c}}\, x^{\mathbf{b}})[x\backslash z^{\mathbf{c}}]_{\{\mathbf{a},\mathbf{b}\}}$$

May be renamed by mapping the label $\mathbf{a}$ to $\mathbf{d}\bullet\mathbf{e}$, the label $\mathbf{b}$ to $\mathbf{d}\bullet\mathbf{e}$, and the label $\mathbf{c}$ to $\mathbf{f}\bullet\mathbf{g}$, obtaining:

$$(x^{\mathbf{d}\bullet\mathbf{e}}\, x^{\mathbf{d}\bullet\mathbf{e}})[x\backslash z^{\mathbf{f}\bullet\mathbf{g}}]_{\{\mathbf{d}\}} \xrightarrow{\mathbf{e}\bullet\mathbf{f}}_\ell (z^{\mathbf{d}\bullet\mathbf{e}\bullet\mathbf{f}\bullet\mathbf{g}}\, x^{\mathbf{d}\bullet\mathbf{e}})[x\backslash z^{\mathbf{f}\bullet\mathbf{g}}]_{\{\mathbf{d}\}}$$

Note that the set $\{\mathbf{a},\mathbf{b}\}$ collapses to the set $\{\mathbf{d}\}$ and that the name of the $\mathtt{ls}$ redex is not $\mathbf{d}\bullet\mathbf{e}\bullet\mathbf{f}\bullet\mathbf{g}$ but rather $\mathbf{e}\bullet\mathbf{f}$. This mechanism can be formalized with the following notion of labeling morphism.

Definition 6.26 (Labeling morphism)**.** A *labeling morphism* ϕ is a function $\phi : \mathcal{L} \to \mathcal{L}$ homomorphic on all label constructors, except for initial labels:

$$\begin{array}{rclrclrcl} \phi(\bullet) & = & \bullet & \phi(\otimes) & = & \otimes & \phi(\mathtt{db}(\alpha)) & = & \mathtt{db}(\phi(\alpha)) \\ \phi(\lceil\alpha\rceil) & = & \lceil\phi(\alpha)\rceil & \phi(\lfloor\alpha\rfloor) & = & \lfloor\phi(\alpha)\rfloor & \phi(\alpha\,\beta) & = & \phi(\alpha)\,\phi(\beta) \end{array}$$

If Ω is a set of labels, we write $\phi(\Omega)$ to stand for $\{\downarrow(\phi(\Omega)) \mid \alpha \in \Omega\}$. The domain of labeling morphisms is extended so that they may be applied on terms as follows:

$$\begin{array}{rclrcl} \phi(x^\alpha) & = & x^{\phi(\alpha)} & \phi(\lambda^\alpha_\Omega x.t) & = & \lambda^{\phi(\alpha)}_{\phi(\Omega)} x.\phi(t) \\ \phi(@^\alpha(t,s)) & = & @^{\phi(\alpha)}(\phi(t),\phi(s)) & \phi(t[x\backslash s]_\Omega) & = & \phi(t)[x\backslash\phi(s)]_{\phi(\Omega)} \end{array}$$

Labeling morphisms may also be applied on contexts, by declaring that $\phi(\Box) = \Box$, and on redex names, as follows:

$$\begin{array}{rcl} \phi(\mathtt{db}(\alpha)) & = & \mathtt{db}(\phi(\alpha)) \\ \phi(\alpha \bullet \beta) & = & \downarrow(\phi(\alpha)) \bullet \uparrow(\phi(\beta)) \\ \phi(\{\mathbf{a} \bullet \beta \mid \mathbf{a} \in \Omega\}) & = & \{\mathbf{a} \bullet \uparrow(\phi(\beta)) \mid \mathbf{a} \in \phi(\Omega)\} \end{array}$$

Remark 6.27. A labeling morphism is uniquely determined by its value on the set of initial labels $\mathcal{I}$.

Lemma 6.28. *If ϕ is a labeling morphism, the following hold for any label $\alpha \in \mathcal{L}$ and any term $t \in \mathcal{T}^\ell$:*

1. $\downarrow(\phi(\alpha)) = \downarrow(\phi(\downarrow(\alpha)))$
2. $\uparrow(\phi(\alpha)) = \uparrow(\phi(\uparrow(\alpha)))$
3. $\uparrow(\phi(t)) = \uparrow(\phi(\uparrow(t)))$

Proof. Items 1. and 2. are straightforward by induction on α. Item 3. is a consequence of item 2. □

Proposition 6.29 (Labeling morphisms are functorial)**.** *Let ϕ be a label morphism. Then for each step $R : t \xrightarrow{\mu}_\ell s$ there is a step $\phi(R) : \phi(t) \xrightarrow{\phi(\mu)}_\ell \phi(s)$.*

Proof. By induction on the context $\mathtt{C}$ under which the redex in t is contracted: The interesting case is when there is a step at the root of the term.

1. $\mathtt{db}$ *step*

$$\begin{array}{ccc} @^\alpha((\lambda^\beta_\Omega x.t)\mathtt{L}, s) & \xrightarrow{\mathtt{db}(\beta)} & \alpha\lceil\mathtt{db}(\beta)\rceil : t[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s]_\Omega\mathtt{L} \\ \downarrow{\scriptstyle\phi(-)} & & \downarrow{\scriptstyle\phi(-)} \\ @^{\phi(\alpha)}((\lambda^{\phi(\beta)}_{\phi(\Omega)} x.\phi(t))\phi(\mathtt{L}), \phi(s)) & \xrightarrow{\mathtt{db}(\phi(\beta))} & \phi(\alpha\lceil\mathtt{db}(\beta)\rceil) : \phi(t)[x\backslash\phi(\lfloor\mathtt{db}(\beta)\rfloor) : \phi(s)]_{\phi(\Omega)}\phi(\mathtt{L}) \end{array}$$

2. `ls` *step*

$$\begin{array}{ccc} \mathtt{C}\langle\!\langle x^{\alpha}\rangle\!\rangle[x\backslash t]_{\Omega} & \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t)} & \mathtt{C}\langle\alpha\bullet : t\rangle[x\backslash t]_{\Omega} \\ \phi(-)\downarrow & & \phi(-)\downarrow \\ \phi(\mathtt{C})\langle\!\langle x^{\phi(\alpha)}\rangle\!\rangle[x\backslash\phi(t)]_{\phi(\Omega)} & \xrightarrow{\downarrow(\phi(\alpha))\bullet\uparrow(\phi(t))} & \phi(\mathtt{C})\langle\phi(\alpha)\bullet : \phi(t)\rangle[x\backslash\phi(t)]_{\phi(\Omega)} \end{array}$$

Note that $\phi(\downarrow(\alpha)\bullet\uparrow(t)) = \downarrow(\phi(\downarrow(\alpha)))\bullet\uparrow(\phi(\uparrow(t))) = \downarrow(\phi(\alpha))\bullet\uparrow(\phi(t))$ by Lem. 6.28.

3. `gc` *step* Let $x \notin \mathsf{fv}(t)$ and:

$$\begin{array}{ccc} t[x\backslash s]_{\Omega} & \xrightarrow{\{\mathbf{a}\bullet\uparrow(s)\ \mid\ \mathbf{a}\in\Omega\}} & t \\ \phi(-)\downarrow & & \phi(-)\downarrow \\ \phi(t)[x\backslash\phi(s)]_{\phi(\Omega)} & \xrightarrow{\{\mathbf{a}\bullet\uparrow(\phi(s))\ \mid\ \mathbf{a}\in\phi(\Omega)\}} & \phi(t) \end{array}$$

Note that $\phi(\{\mathbf{a}\bullet\uparrow(s) \mid \mathbf{a}\in\Omega\}) = \{\mathbf{a}\bullet\uparrow(\phi(\uparrow(s))) \mid \mathbf{a}\in\uparrow(\Omega)\} = \{\mathbf{a}\bullet\uparrow(\phi(s)) \mid \mathbf{a}\in\uparrow(\Omega)\}$ by Lem. 6.28.

□

As a consequence of the previous proposition, labeling morphisms can be applied on derivations, setting $\phi(R_1 \dots R_n) = \phi(R_1) \dots \phi(R_n)$.

6.3.2 Orthogonality

In this section we show that the LLSC is confluent. Actually, we prove the much stronger property that the LLSC forms an orthogonal axiomatic rewriting system in the sense of Melliès, as defined in Def. 2.39.

We begin by showing that the LLSC is weakly Church–Rosser. Recall from Def. 2.18 that an abstract rewriting system is weakly Church–Rosser if every peak $\leftarrow\!\!\rightarrow$ formed by exactly two steps can be closed with zero or more steps $\twoheadrightarrow\!\!\twoheadleftarrow$. In the labeled calculus a stronger result can be established, which we call *strong permutation*. It states that every peak $\overset{\mu}{\leftarrow}\!\!\overset{\nu}{\rightarrow}$, where μ and ν are the names of the steps, can be closed with zero or more steps *of the same name*, that is $\overset{\nu}{\twoheadrightarrow}\!\!\overset{\mu}{\twoheadleftarrow}$.

Proposition 6.30 (Strong permutation — ♣ Prop. A.79)**.** *Let $R : t \xrightarrow{\mu}_{\ell} s$ and $S : t \xrightarrow{\nu}_{\ell} u$ be steps in the LLSC. Then there exists a term r and two derivations $\sigma : s \xrightarrow{\nu}\!\!\!\!\rightarrow_{\ell} r$ and $\rho : u \xrightarrow{\mu}\!\!\!\!\rightarrow_{\ell} r$. Diagrammatically:*

$$\begin{array}{ccc} t & \xrightarrow{\mu} & s \\ \nu\downarrow & & \nu\downarrow \\ u & \xrightarrow[\mu]{} & r \end{array}$$

The proof is constructive, and:

1. *If R is a* `db` *step, σ consists of exactly one step.*

2. *If R is a* `ls` *step, σ may consist of one or two steps.*

3. *If R is a* `gc` *step, σ may consist of zero or one steps.*

And symmetrically for S and ρ.

Proof. By exhaustive case analysis. See the appendix for the detailed proof. Below we show three examples that illustrate some interesting situations:

1. Nested `db` *steps.*

$$\begin{array}{ccc} @^{\mathbf{a}}(\lambda^{\mathbf{b}}_{\{\mathbf{c}\}}x.x^{\mathbf{c}}, @^{\mathbf{d}}(\lambda^{\mathbf{e}}_{\{\mathbf{f}\}}y.y^{\mathbf{f}}, z^{\mathbf{g}})) & \xrightarrow{\mathtt{db}(\mathbf{b})} & x^{\mathbf{a}\lceil \mathtt{db}(\mathbf{b})\rceil^{\mathbf{c}}}[x\backslash @^{\lfloor \mathtt{db}(\mathbf{b})\rfloor^{\mathbf{d}}}(\lambda^{\mathbf{e}}_{\{\mathbf{f}\}}y.y^{\mathbf{f}}, z^{\mathbf{g}})]_{\mathbf{c}} \\ {\scriptstyle \mathtt{db}(\mathbf{e})}\downarrow & & {\scriptstyle \mathtt{db}(\mathbf{e})}\downarrow \\ @^{\mathbf{a}}(\lambda^{\mathbf{b}}_{\{\mathbf{c}\}}x.x^{\mathbf{c}}, y^{\mathbf{d}\lceil \mathtt{db}(\mathbf{e})\rceil^{\mathbf{f}}}[y\backslash z^{\lfloor \mathtt{db}(\mathbf{e})\rfloor^{\mathbf{g}}}]_{\{\mathbf{f}\}}) & \xrightarrow{\mathtt{db}(\mathbf{b})} & x^{\mathbf{a}\lceil \mathtt{db}(\mathbf{b})\rceil^{\mathbf{c}}}[x\backslash y^{\lfloor \mathtt{db}(\mathbf{b})\rfloor^{\mathbf{d}}\lceil \mathtt{db}(\mathbf{e})\rceil^{\mathbf{f}}}[y\backslash z^{\lfloor \mathtt{db}(\mathbf{e})\rfloor^{\mathbf{g}}}]_{\{\mathbf{f}\}}]_{\mathbf{c}} \end{array}$$

2. Duplication of a `ls` *step by a* `ls` *step.*

$$\begin{array}{ccc} x^{\mathbf{a}}[x\backslash y^{\mathbf{b}}]_{\{\mathbf{a}\}}[y\backslash z^{\mathbf{c}}]_{\{\mathbf{b}\}} & \xrightarrow{\mathbf{a}\bullet\mathbf{b}} & y^{\mathbf{a}\bullet\mathbf{b}}[x\backslash y^{\mathbf{b}}]_{\{\mathbf{a}\}}[y\backslash z^{\mathbf{c}}]_{\{\mathbf{b}\}} \\ & & {\scriptstyle \mathbf{b}\bullet\mathbf{c}}\downarrow \\ {\scriptstyle \mathbf{b}\bullet\mathbf{c}}\downarrow & & z^{\mathbf{a}\bullet\mathbf{b}\bullet\mathbf{c}}[x\backslash y^{\mathbf{b}}]_{\{\mathbf{a}\}}[y\backslash z^{\mathbf{c}}]_{\{\mathbf{b}\}} \\ & & {\scriptstyle \mathbf{b}\bullet\mathbf{c}}\downarrow \\ x^{\mathbf{a}}[x\backslash z^{\mathbf{b}\bullet\mathbf{c}}]_{\{\mathbf{a}\}}[y\backslash z^{\mathbf{c}}]_{\{\mathbf{b}\}} & \xrightarrow{\mathbf{a}\bullet\mathbf{b}} & z^{\mathbf{a}\bullet\mathbf{b}\bullet\mathbf{c}}[x\backslash z^{\mathbf{b}\bullet\mathbf{c}}]_{\{\mathbf{a}\}}[y\backslash z^{\mathbf{c}}]_{\{\mathbf{b}\}} \end{array}$$

Note that, if there is duplication, there are exactly two ways to close the diagram, depending on the order in which the copies of the duplicated steps are contracted.

3. Erasure of a `ls` *step by a* `gc` *step.*

$$\begin{array}{ccc} x^{\mathbf{a}}[w\backslash y^{\mathbf{b}}]_{\{\otimes\}}[y\backslash z^{\mathbf{c}}]_{\{\mathbf{b}\}} & \xrightarrow{\{\otimes\bullet\mathbf{b}\}} & x^{\mathbf{a}}[y\backslash z^{\mathbf{c}}]_{\{\mathbf{b}\}} \\ {\scriptstyle \mathbf{b}\bullet\mathbf{c}}\downarrow & & \| \\ x^{\mathbf{a}}[w\backslash z^{\mathbf{b}\bullet\mathbf{c}}]_{\{\otimes\}}[y\backslash z^{\mathbf{c}}]_{\{\mathbf{b}\}} & \xrightarrow{\{\otimes\bullet\mathbf{b}\}} & x^{\mathbf{a}}[y\backslash z^{\mathbf{c}}]_{\{\mathbf{b}\}} \end{array}$$

□

Definition 6.31 (Residuals for the LLSC)**.** Recall that the LSC with its usual residual relation forms an orthogonal axiomatic rewriting system (Prop. 6.3). The LLSC is provided with a residual relation by relying on the residual relation of the LSC as follows.

If $t \in \mathcal{T}^{\ell}$ is a labeled term, let us write $|t| \in \mathcal{T}$ for the term without labels that results from erasing all labels from t. Similarly, if $R : t \to_{\ell} s$ is a labeled step in the LLSC, let us write $|R| : |t| \to_{\mathsf{LSC}} |s|$ for the corresponding step in the LSC, via the obvious bijection.

Let $R : t \to_{\ell} s$ be a labeled step and consider two labeled steps $S_1 : t \to_{\ell} u$ and $S_2 : s \to_{\ell} r$. We declare the residual relation $S_1 \langle R \rangle S_2$ to hold in the LLSC if and only if the usual relation $|S_1| \langle |R| \rangle |S_2|$ holds in the LSC.

Proposition 6.32 (Orthogonality)**.** *The LLSC forms an orthogonal axiomatic rewriting system.*

Proof. It can be checked that the LLSC provided with the residual relation of Def. 6.31 forms an orthogonal axiomatic rewriting system. The first three axioms: **Autoerasure**, **Finite Residuals**, and **Finite Developments** (FD), are immediate consequences of the fact that the LSC is an orthogonal axiomatic rewriting system. For example, to prove FD, suppose that there is an infinite development $t_1 \to_\ell t_2 \to_\ell \dots$ of a set of coinitial steps $\mathcal{M}$ in the LLSC. Then $|t_1| \to_{\mathsf{LSC}} |t_2| \to_{\mathsf{LSC}} \dots$ is an infinite development of the set $\{|R| \mid R \in \mathcal{M}\}$ in the LSC, contradicting the fact that the LSC enjoys the FD property.

The **Semantic Orthogonality** (SO) axiom is a consequence of Strong permutation (Prop. 6.30). Strictly speaking, SO can be checked by exhaustively inspecting all the diagrams constructed in the proof of Prop. 6.30 and checking that they are indeed permutation tiles (*cf.* Def. 2.37). □

The following lemma proves the **Copy** property from the Bestiary of Section 6.2.1, and a weak form of the converse implication:

Lemma 6.33 (Copy property**).** *Let R_1, S, and R_2 be steps such that $\mathsf{src}(R_1) = \mathsf{src}(S)$ and $\mathsf{src}(R_2) = \mathsf{tgt}(S)$. Then:*

1. *If $R_1 \langle S \rangle R_2$ in the LLSC, then R_1 and R_2 have the same name.*
2. *If $\mathsf{src}(R_1)$ is initially labeled and R_1 and R_2 have the same name, then $R_1 \langle S \rangle R_2$.*

Proof. Item 1. is an immediate consequence of Strong permutation (Prop. 6.30). Namely, if we consider the peak formed by S and R_1, Prop. 6.30 ensures that the step R_2 used to close the diagram has the same name as R_1.

We omit the technical proof of item 2. Define the *anchor label* of a step as follows. Given a `db` step named $\mathtt{db}(\mathbf{b})$, its anchor label is the label decorating the lambda, that is, **b**. Given a `ls` step named $\mathbf{a} \bullet \mathbf{b}$, its anchor label is the label decorating the contracted variable, that is, **a**. Given a `gc` step named $\{\mathbf{a}_1 \bullet \mathbf{b}, \dots, \mathbf{a}_n \bullet \mathbf{b}\}$, its anchor label is the label decorating the erased substitution, that is, **b**. Let $S : t \to_\ell s$ and recall that t is initially labeled so there is a single occurrence of the anchor label of R_1 in t. Consider three cases, depending on whether the number of residuals $\#(R_1/S)$ is 0, 1, or 2.

0. If $R_1/S = \varnothing$ then S is a `gc` step erasing the contracted substitution. Then S erases the unique occurrence of the anchor label of R_1, that is, the anchor label of R_1 does not appear anywhere in in s, contradicting the fact that R_1 and R_2 have the same name. So this case is impossible.

1. If $R_1/S = \{R_2'\}$ then S does not erase or duplicate of the anchor label of R_1. This means that there is a unique occurrence of the anchor label of R_1 in s, and this implies that $R_2 = R_2'$.

2. If $R_1/S = \{R_{21}, R_{22}\}$ then S makes exactly two copies of the anchor label of R_1, so there are exactly two occurrences of the anchor label of R_1 in s, and this implies that $R_2 \in \{R_{21}, R_{22}\}$.

□

Remark 6.34. The converse of the **Copy** property does not hold if the term is not initially labeled, and even if it is initially reachable. For example, the source of R_1 and S below is initially reachable but not initially labeled, and even though R_1 and R_2 have the same name, it is not the case that $R_1 \langle S \rangle R_2$:

$$\begin{array}{ccccc} (x^{\mathbf{a}}x^{\mathbf{b}})[x\backslash y^{\mathbf{c}}]_{\{\mathbf{a},\mathbf{b}\}}[y\backslash z^{\mathbf{d}}]_{\{\mathbf{c}\}} & \xrightarrow{\mathbf{a}\bullet\mathbf{c}} & (y^{\mathbf{a}\bullet\mathbf{c}}x^{\mathbf{b}})[x\backslash y^{\mathbf{c}}]_{\{\mathbf{a},\mathbf{b}\}}[y\backslash z^{\mathbf{d}}]_{\{\mathbf{c}\}} & \xrightarrow{S(\mathbf{b}\bullet\mathbf{c})} & (y^{\mathbf{a}\bullet\mathbf{c}}y^{\mathbf{b}\bullet\mathbf{c}})[x\backslash y^{\mathbf{c}}]_{\{\mathbf{a},\mathbf{b}\}}[y\backslash z^{\mathbf{d}}]_{\{\mathbf{c}\}} \\ & & \downarrow{\scriptstyle R_1(\mathbf{c}\bullet\mathbf{d})} & & \downarrow{\scriptstyle R_2(\mathbf{c}\bullet\mathbf{d})} \\ & & (z^{\mathbf{a}\bullet\mathbf{c}\bullet\mathbf{d}}x^{\mathbf{b}})[x\backslash y^{\mathbf{c}}]_{\{\mathbf{a},\mathbf{b}\}}[y\backslash z^{\mathbf{d}}]_{\{\mathbf{c}\}} & & (y^{\mathbf{a}\bullet\mathbf{c}}z^{\mathbf{b}\bullet\mathbf{c}\bullet\mathbf{d}})[x\backslash y^{\mathbf{c}}]_{\{\mathbf{a},\mathbf{b}\}}[y\backslash z^{\mathbf{d}}]_{\{\mathbf{c}\}} \end{array}$$

One important corollary of Orthogonality is that labeling is consistent with permutation equivalence.

Proposition 6.35 (Permutation equivalent derivations yield the same labellings)**.** *Let ρ_1 and ρ_2 be permutation equivalent derivations,* i.e. $\rho_1 \equiv \rho_2$*. Let ρ_1^ℓ and ρ_2^ℓ be labeled variants of ρ_1 and ρ_2 respectively such that* $\mathsf{src}(\rho_1^\ell) = \mathsf{src}(\rho_2^\ell)$*,* i.e. *they start on the same labeled term. Then* $\mathsf{tgt}(\rho_1^\ell) = \mathsf{tgt}(\rho_2^\ell)$*,* i.e. *they end on the same labeled term.*

Proof. Recall from Def. 2.40 that $\equiv$ is the reflexive, symmetric and transitive closure of the one-step permutation axiom $\equiv^1$. We proceed by induction on the derivation that $\rho_1 \equiv \rho_2$. The reflexivity, symmetry and transitivity cases are immediate. The only interesting case is the axiom, *i.e.* when:

$$\rho_1 = \tau_1\, R\, \sigma\, \tau_2 \equiv^1 \tau_1\, S\, \rho\, \tau_2 = \rho_2$$

where ρ is a complete development of R/S and σ is a complete development of S/R. Consider the labeled variants τ_1^ℓ, R^ℓ, S^ℓ, σ^ℓ, ρ^ℓ, τ_2^ℓ, and $\tau_2^{\ell\ell}$ of τ_1, R, S, σ, ρ, τ_2, and τ_2 respectively, such that:

- $\tau_1^\ell\, R^\ell\, \sigma^\ell\, \tau_2^\ell$ is a labelled variant of $\tau_1\, R\, \sigma\, \tau_2$ whose source is t^ℓ,
- $\tau_1^\ell\, S^\ell\, \rho^\ell\, \tau_2^{\ell\ell}$ is a labelled variant of $\tau_1\, S\, \rho\, \tau_2$ whose source is t^ℓ.

Then we know that $R^\ell\, \sigma^\ell$ has the same source as $S^\ell\, \rho^\ell$, and we claim that they have the same target. This is a consequence of the strong permutation property (Prop. 6.30), observing that every diagram in the proof of Prop. 6.30 is closed with the relative residuals of R and S. Since $R^\ell\, \sigma^\ell$ and $S^\ell\, \rho^\ell$ have the same target, then $\tau_2^\ell = \tau_2^{\ell\ell}$, so we conclude that ρ_1^ℓ and ρ_2^ℓ have the same target, as required. □

6.3.3 Weak Normalization for Bounded Reduction

Consider the following infinite derivation in the LLSC, starting from an initially labeled variant of the non-terminating term $\Omega = (\lambda x.xx)(\lambda x.xx)$ and lifting the reduction $\Omega \twoheadrightarrow_{\mathsf{LSC}} \Omega \twoheadrightarrow_{\mathsf{LSC}} \ldots$ to the labeled calculus:

$$\begin{array}{ll} & @^{\mathbf{a}}(\lambda^{\mathbf{b}}_{\{\mathbf{d},\mathbf{e}\}}x.@^{\mathbf{c}}(x^{\mathbf{d}}, x^{\mathbf{e}}), \lambda^{\mathbf{f}}_{\{\mathbf{h},\mathbf{i}\}}x.@^{\mathbf{g}}(x^{\mathbf{h}}, x^{\mathbf{i}})) \\ \twoheadrightarrow_\ell & @^{\mathbf{a}[\mathtt{db}(\mathbf{b})]\mathbf{c}}(\lambda^{\mathbf{d}\bullet\mathbf{a}[\mathtt{db}(\mathbf{b})]\mathbf{f}}_{\{\mathbf{h},\mathbf{i}\}}x.@^{\mathbf{g}}(x^{\mathbf{h}}, x^{\mathbf{i}}), \lambda^{\mathbf{e}\bullet\mathbf{a}[\mathtt{db}(\mathbf{b})]\mathbf{f}}_{\{\mathbf{h},\mathbf{i}\}}x.@^{\mathbf{g}}(x^{\mathbf{h}}, x^{\mathbf{i}})) \\ \twoheadrightarrow_\ell & @^{\mathbf{a}[\mathtt{db}(\mathbf{b})]\mathbf{c}[\mathtt{db}(\mathbf{d}\bullet\mathbf{a}[\mathtt{db}(\mathbf{b})]\mathbf{f})]\mathbf{g}}(\lambda^{\mathbf{h}\bullet[\mathtt{db}(\mathbf{d}\bullet\mathbf{a}[\mathtt{db}(\mathbf{b})]\mathbf{f})]\mathbf{e}\bullet\mathbf{a}[\mathtt{db}(\mathbf{b})]\mathbf{f}}_{\{\mathbf{h},\mathbf{i}\}}x.\ldots, \ldots) \\ \twoheadrightarrow_\ell & \ldots \end{array}$$

One can see that the names of the db steps become progressively larger in size. More precisely, the name of each db step is strictly contained in the name of the following db step, as evidenced by the underlining:

$$\begin{aligned}
&\mathtt{db}(\mathbf{b})\\
&\mathtt{db}(\mathbf{d} \bullet \mathbf{a}\lfloor \underline{\mathtt{db}(\mathbf{b})} \rfloor \mathbf{f})\\
&\mathtt{db}(\mathbf{h} \bullet \lfloor \underline{\mathtt{db}(\mathbf{d} \bullet \mathbf{a}\lfloor \mathtt{db}(\mathbf{b}) \rfloor \mathbf{f})} \rfloor \mathbf{e} \bullet \mathbf{a}\lfloor \mathtt{db}(\mathbf{b}) \rfloor \mathbf{f})\\
&\dots
\end{aligned}$$

This means that the names of the db steps become not merely larger but also *deeper* in height, if labels are seen as trees. Informally speaking, this is because the LLSC is designed to verify the **Creation** principle in the Bestiary of properties given in Section 6.2.1. Recall that the **Creation** principle states that whenever a step R creates a second step S, the name of R is a sublabel of the name of S. In this case, each db redex contributes to the creation of the following one, and as a consequence the name of each db step is a sublabel of the name of the following one.

In this section we show that if the rewriting relation of the LLSC is restricted so that the *height* of the names of steps is bounded, the resulting rewriting relation turns out to be weakly normalizing. In the following section (Section 6.3.4) we prove that it actually turns out to be strongly normalizing. Note that the strong normalization result explains and generalizes the example given above: it means that whenever we have an infinite labeled reduction sequence $t_1 \to_\ell t_2 \to_\ell \dots$ the names of the steps must be labels whose height cannot be bounded by any integer.

Below we introduce the auxiliary calculi LLSC^P and LLSC^{PI}, which only allow contracting a step R if the name of R verifies a given predicate P. We also introduce the notion of *bounded predicate*.

Definition 6.36 (The P-restricted LLSC)**.** Let P be a predicate on redex names. We define two calculi, LLSC^P and LLSC^{PI}. The set of terms is $\mathcal{T}^\ell$ in both cases. The reduction relation $\to_{\ell\, P}$ is defined as in the LLSC, restricted to contracting only steps whose names verify the predicate P. The reduction relation $\to_{\ell\, PI}$ is defined similarly, but restricted to contracting db and ls-steps:

$$\begin{aligned}
t \xrightarrow{\mu}_{\ell\, P} s &\iff^{\text{def}} t \xrightarrow{\mu}_\ell s \ \wedge\ P(\mu) \text{ holds}\\
t \xrightarrow{\mu}_{\ell\, PI} s &\iff^{\text{def}} (t \xrightarrow{\mu}_{\ell\, \mathtt{db}} s \ \vee\ t \xrightarrow{\mu}_{\ell\, \mathtt{ls}} s) \ \wedge\ P(\mu) \text{ holds}
\end{aligned}$$

Note that since there are no gc-steps, the name of a step in the LLSC^{PI} can always be understood as a label. We write $t \to_{\ell\, P} s$ if $t \xrightarrow{\mu}_{\ell\, P} s$ holds for some redex name μ, and similarly for $\to_{\ell\, PI}$.

Definition 6.37 (Height of labels and redex names)**.** We define the height of a label as follows:

$$\begin{aligned}
h(\mathbf{a}) &\stackrel{\text{def}}{=} 1\\
h(\lceil\alpha\rceil) = h(\lfloor\alpha\rfloor) = h(\mathtt{db}(\alpha)) &\stackrel{\text{def}}{=} 1 + h(\alpha)\\
h(\alpha\beta) &\stackrel{\text{def}}{=} \max\{h(\alpha), h(\beta)\}
\end{aligned}$$

Similarly, we define the height of a redex name as follows:

$$\begin{array}{lrcll} \texttt{db} \text{ redex:} & h(\texttt{db}(\alpha)) & \stackrel{\text{def}}{=} & 1 + h(\alpha) & \\ \texttt{ls} \text{ redex:} & h(\alpha \bullet \beta) & \stackrel{\text{def}}{=} & \max\{h(\alpha), h(\beta)\} & \text{where } \alpha, \beta \text{ are atomic} \\ \texttt{gc} \text{ redex:} & h(\{\mathbf{a} \bullet \beta \mid \mathbf{a} \in \Omega\}) & \stackrel{\text{def}}{=} & \max\{h(\mathbf{a} \bullet \beta) \mid \mathbf{a} \in \Omega\} & \text{where } \beta \text{ is atomic} \end{array}$$

Note that in the case of `db` and `ls` redexes the height of the redex name coincides with its height if seen as a label.

Definition 6.38 (Bounded predicate)**.** A predicate P on redex names is said to be *bounded* if and only if there exists a bound $H \in \mathbb{N}$ such that for every redex name μ, if $P(\mu)$ holds then $h(\mu) < H$.

The result of weak normalization from this section should be seen as a stepping stone for the stronger result of strong normalization in the next section. In particular, for the time being, we can dispense of `gc` steps and work exclusively with the LLSC^{PI}. Our current goal is then to prove that the rewriting relation $\xrightarrow{\mu}_{\ell\, PI}$ is weakly normalizing if P is bounded. This is the content of Prop. 6.45 below. We sketch the structure of the proof:

1. In Prop. 6.41, we show that the LLSC without `gc` verifies the **Creation** principle in the Bestiary of Section 6.2.1.
2. In Lem. 6.43, we show that among the steps from a LLSC^{PI} term, there is at least one *non-duplicating* step.
3. In Lem. 6.44, we show that contracting a non-duplicating step has the following effect:
   - 3.1 Every other step is preserved (*i.e.* it has exactly one residual).
   - 3.2 The created redexes have *deeper* names than the contracted redex, *i.e.* the height of the label increases.

These results will allow us to define a measure on terms that always decreases when contracting a non-duplicating redex.

Creation principle

The following relation of *name contribution* corresponds to the informal notion that the name of a redex is "contained" in the name of another redex.

Definition 6.39 (Name contribution)**.** A redex name μ is said to *directly contribute* to a redex name ν, written $\mu \overset{\text{Name}}{\hookrightarrow}_1 \nu$, if one of the three following cases holds:

$$\begin{array}{rcll} \texttt{db}(\beta) & \overset{\text{Name}}{\hookrightarrow}_1 & \texttt{db}(\alpha \lceil \texttt{db}(\beta) \rceil \gamma) & \\ \texttt{db}(\beta) & \overset{\text{Name}}{\hookrightarrow}_1 & \alpha \bullet \lfloor \texttt{db}(\beta) \rfloor & \text{where } \alpha \text{ is any atomic label} \\ \downarrow(\alpha) \bullet \uparrow(\beta) & \overset{\text{Name}}{\hookrightarrow}_1 & \texttt{db}(\alpha \bullet \beta) & \end{array}$$

A redex name μ is said to (indirectly) *contribute* to a redex name ν, written $\mu \overset{\text{Name}}{\hookrightarrow} \nu$, if $\mu (\overset{\text{Name}}{\hookrightarrow}_1)^* \nu$.

Remark 6.40. If $\mu \overset{\text{Name}}{\hookrightarrow_1} \nu$ then $h(\mu) < h(\nu)$.

Proposition 6.41 (Creation property for the LLSC without gc**).** *Let $t \xrightarrow{\mu}_\ell s \xrightarrow{\nu}_\ell u$ be a sequence of two steps, each of which may be a* db *step or a* ls *step but not a* gc *step. If the first step creates the second one, then $\mu \overset{\text{Name}}{\hookrightarrow_1} \nu$.*

Proof. Recall that the notion of residual in the LLSC is defined in terms of the notion of residual in the LSC (Def. 6.31), *i.e.* the residual relation $S_1 \langle R \rangle S_2$ holds for three given labeled steps if and only if $|S_1| \langle |R| \rangle |S_2|$ holds for the underlying unlabeled variants. Recall also that in the LSC there are seven creation cases (Prop. 6.4), three of which involve gc steps. So it suffices to analyze the remaining four creation cases:

1. db *creates* db. The situation is:

$$\begin{array}{rl} & @^\gamma(@^\delta((\lambda^\beta_\Omega x.(\lambda^\varepsilon_\Theta y.t)\mathrm{L}_1)\mathrm{L}_2, s)\mathrm{L}_3, u) \\ \xrightarrow{\mathtt{db}(\beta)}_\ell & @^\gamma((\lambda^{\delta\lceil \mathtt{db}(\beta)\rceil\varepsilon}_\Theta y.t)\mathrm{L}_1[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : s]_\Omega \mathrm{L}_2\mathrm{L}_3, u) \end{array}$$

   So indeed $\mathtt{db}(\beta) \overset{\text{Name}}{\hookrightarrow_1} \mathtt{db}(\delta\lceil \mathtt{db}(\beta)\rceil\varepsilon)$.

2. db *creates* ls. The situation is

$$@^\gamma((\lambda^\beta_\Omega x.\mathtt{C}\langle\!\langle x^\delta \rangle\!\rangle)\mathrm{L}, t) \xrightarrow{\mathtt{db}(\beta)}_\ell \gamma\lceil \mathtt{db}(\beta)\rceil : \mathtt{C}\langle\!\langle x^\delta \rangle\!\rangle[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : t]_\Omega \mathrm{L}$$

   By Lem. 6.9, $\gamma\lceil \mathtt{db}(\beta)\rceil : \mathtt{C}\langle\!\langle x^\delta \rangle\!\rangle$ is of the form $\mathtt{C}'\langle\!\langle x^{\delta'} \rangle\!\rangle$ with $\downarrow(\delta) = \downarrow(\delta')$. Moreover, by Lem. 6.9, $\uparrow(\lfloor \mathtt{db}(\beta)\rfloor : t) = \uparrow(\lfloor \mathtt{db}(\beta)\rfloor) = \lfloor \mathtt{db}(\beta)\rfloor$. So we conclude that $\mathtt{db}(\beta) \overset{\text{Name}}{\hookrightarrow_1} \downarrow(\delta) \bullet \lfloor \mathtt{db}(\beta)\rfloor$, as required.

3. ls *creates* db *upwards.* The situation is:

$$@^\beta(x^\gamma \mathrm{L}_1[x\backslash(\lambda^\delta_\Theta y.s)\mathrm{L}_2]_\Omega \mathrm{L}_3, t) \xrightarrow{\downarrow(\gamma)\bullet\uparrow(\delta)}_{\ell\,\mathtt{ls}} @^\beta((\lambda^{\gamma\bullet\delta}_\Theta y.s)\mathrm{L}_2\mathrm{L}_1[x\backslash(\lambda^\delta_\Theta y.s)\mathrm{L}_2]_\Omega \mathrm{L}_3, t)$$

   So indeed $\downarrow(\gamma) \bullet \uparrow(\delta) \overset{\text{Name}}{\hookrightarrow_1} \mathtt{db}(\gamma \bullet \delta)$.

4. ls *creates* db *downwards.* Similar to the previous case.

□

Non-duplicating steps

Definition 6.42 (Non-duplicating step)**.** Given any axiomatic rewriting system with a notion of residual, a step $R : t \to s$ is said to be *non-duplicating* if any coinitial step $S : t \to u$ has at most one residual after R, that is, $\#(S/R) \leqslant 1$.

Lemma 6.43 (Existence of non-duplicating $\to_{\ell\, PI}$-steps)**.** *Let $t \in \mathcal{T}^\ell$ not in $\to_{\ell\, PI}$-normal form. Then t has at least one non-duplicating $\to_{\ell\, PI}$-redex.*

Proof. Since t is not in $\to_{\ell\, PI}$ -normal form, it has at least one $\to_{\ell\, PI}$ -redex. Let $R : t \to_{\ell\, PI} s$ be the step whose anchor lies more to the right. Recall that the anchor of a $\mathtt{db}$ step is the variable bound by the λ, and the anchor of a $\mathtt{ls}$ step is the variable affected by the substitution. Let $S : t \to_{\ell\, PI} u$ be any step coinitial to R, and let us check that $\#(S/R) \leqslant 1$. If R is a $\mathtt{db}$ step, then trivially $\#(S/R) \leqslant 1$. If R is a $\mathtt{ls}$ step, suppose that $\#(S/R) > 1$. Then the step R is of the form $\mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x^\alpha \rangle\!\rangle [x\backslash r]\rangle \to \mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle \alpha \bullet : r \rangle\!\rangle [x\backslash r]\rangle$ and the anchor of S must lie inside r. This contradicts the fact that R is the step whose anchor lies more to the right. □

Lemma 6.44 (Effect of contracting a non-duplicating step)**.** *Let* $\mathsf{names}_{PI}(t)$ *denote the multiset of names of* $\to_{\ell\ PI}$ *-steps of* t*. Let* $R : t \xrightarrow{\alpha}_{\ell\ PI} s$ *be a non-duplicating step. Then there exist multisets of labels* $\mathfrak{m}$ *and* $\mathfrak{n}$ *such that:*

$$\begin{array}{rcl} \mathsf{names}_{PI}(t) & = & \mathfrak{m} \uplus \{\alpha\} \\ \mathsf{names}_{PI}(s) & = & \mathfrak{m} \uplus \mathfrak{n} \end{array}$$

and moreover $h(\alpha) < h(\beta)$ *for every label* $\beta \in \mathfrak{n}$*. Note that* α *is the name of the contracted step and* $\mathfrak{n}$ *are the names of the created steps.*

Proof. Since α is the name of a step of t, we can write $\mathsf{names}_{PI}(t)$ as $\mathsf{names}_{PI}(t) = \mathfrak{m} \uplus \{\alpha\}$. Since R is a non-duplicating step and it is not a $\mathtt{gc}$ step, given any step $S : t \to_{\ell\, PI} u$ such that $R \neq S$ we have that S has a single residual, that is, S/R is a singleton. Moreover, by the **Copy** property (Lem. 6.33), S/R has the same name as S. So we have that $\mathsf{names}_{PI}(t) = \mathfrak{m} \uplus \mathfrak{n}$, where $\mathfrak{n}$ is the multiset of names of the redexes created in this step. Recall that the name of the contracted redex contributes to the names of the created redexes (Prop. 6.41). That is, $\alpha \overset{\text{Name}}{\hookrightarrow}_1 \beta$ for every $\beta \in \mathfrak{n}$. This implies, in particular, that $h(\alpha) < h(\beta)$ for every $\beta \in \mathfrak{n}$. □

Weak normalization for bounded reduction without $\mathtt{gc}$

We are now able to prove the main result of this subsection. The argument for weak normalization relies on the extension of a well-founded ordering $(X, >)$ to a well-founded ordering $(>)$ over multisets of elements of X, as described in Thm. 2.29.

Proposition 6.45 (Bounded reduction is weakly normalizing)**.** *If* P *is a bounded predicate, then* $\to_{\ell\, PI}$ *is* WN*.*

Proof. Let $H \in \mathbb{N}$ be a bound for the bounded predicate P. Consider the following measure, which takes a multiset of labels and yields a multiset of integers, $\#(\mathfrak{m}) \overset{\text{def}}{=} \{H - h(\alpha) \mid \alpha \in \mathfrak{m}\}$. This measure can be extended to work over terms, by declaring $\#(t) \overset{\text{def}}{=} \#(\mathsf{names}_{PI}(t))$. Note that $\#(t)$ is finite and its elements are non-negative integers. The proof relies on the following claim:

- **Claim.** If t is not in $\to_{\ell\, PI}$ -normal form, there is a step $t \to_{\ell\, PI} s$ such that $\#(t) > \#(s)$. *Proof of the claim.* Let $t \in \mathcal{T}^\ell$ be a term not in $\to_{\ell\, PI}$ -normal form. By Lem. 6.43, it has at least one non-duplicating redex $R : t \to_{\ell\, PI} s$. By Lem. 6.44, that there exist multisets $\mathfrak{m}$ and $\mathfrak{n}$ such that $\mathsf{names}_{PI}(t) = \mathfrak{m} \uplus \{\alpha\}$ and $\mathsf{names}_{PI}(s) = \mathfrak{m} \uplus \mathfrak{n}$, where moreover $h(\alpha) < h(\beta)$ for all $\beta \in \mathfrak{n}$. It suffices to check that $\#(t) > \#(s)$.

We begin by observing that $\#(t)(H - h(\alpha)) > \#(s)(H - h(\alpha))$. To see this, consider any label that has the same height as α. It cannot belong to $\mathfrak{n}$, since all labels in $\mathfrak{n}$ have greater height than α. Then $\#(\mathfrak{n})(H - h(\alpha)) = 0$ and we have:

$$\begin{array}{rcl} \#(t)(H - h(\alpha)) & = & \#(\mathfrak{m} \uplus \{\alpha\})(H - h(\alpha)) \\ & > & \#(\mathfrak{m})(H - h(\alpha)) \\ & = & \#(\mathfrak{m} \uplus \mathfrak{n})(H - h(\alpha)) \\ & = & \#(s)(H - h(\alpha)) \end{array}$$

This observation implies, in particular, that $\#(t) \neq \#(s)$. Now consider a value $x \in \mathbb{Z}_{\geqslant 0}$ such that $\#(s)(x) > \#(t)(x)$. Since $\#(s)(x) > 0$, there is a label β such that $x = H - h(\beta)$. Clearly $h(\beta) \neq h(\alpha)$, since we have already proved $\#(t)(H - h(\alpha)) > \#(s)(H - h(\alpha))$, which would contradict the fact that $\#(t)(x) < \#(s)(x)$. Therefore $\#(\{\alpha\})(x) = \#(\{\alpha\})(H - \beta) = 0$.

Moreover, there must be a label $\beta' \in \mathfrak{n}$ of the same height as β. By contradiction, suppose that all labels of the same height as β were in $\mathfrak{m}$. Then $\#(\mathfrak{n})(H - \beta) = 0$, which implies:

$$\begin{array}{rcl} \#(t)(H - h(\beta)) & = & \#(\mathfrak{m} \uplus \{\beta\})(H - h(\beta)) \\ & = & \#(\mathfrak{m})(H - h(\beta)) \\ & = & \#(\mathfrak{m} \uplus \mathfrak{n})(H - h(\beta)) \\ & = & \#(s)(H - h(\beta)) \end{array}$$

This also contradicts the fact that $\#(t)(x) < \#(s)(x)$. Then $x = H - h(\beta')$ with $\beta' \in \mathfrak{n}$. Finally, we want to show that there is a non-negative integer $y \in \mathbb{Z}_{\geqslant 0}$ such that $y > x$ and $\mathfrak{m}(y) > \mathfrak{m}(x)$. Let $y := H - h(\alpha)$. In fact, since $\beta' \in \mathfrak{n}$ is of greater height than α, we have $y = H - h(\alpha) > H - h(\beta') = x$. Moreover, as we have already observed, $\#(t)(y) = \#(t)(H - h(\alpha)) > \#(s)(H - h(\alpha)) = \#(s)(y)$. This concludes the proof of the claim.

By repeatedly applying the claim, it is immediate to conclude that $\to_{\ell\, PI}$ is WN. □

6.3.4 Strong Normalization for Bounded Reduction

In this section we build upon, and strengthen, the normalization result of Prop. 6.45, by showing that the full LLSC (with gc) is *strongly* normalizing as long as reduction is restricted so that the height of redex names is bounded. The structure of the proof is as follows:

1. First we show that $\to_{\ell\, PI}$ is increasing (Lem. 6.48). Recall from Def. 2.21 that a rewriting relation $\to \subseteq X^2$ is increasing if there exists a function $f : X \to \mathbb{N}$ such that $x \to y$ implies $f(x) < f(y)$.

2. Using the previous property, we conclude in Lem. 6.49 that $\to_{\ell\, PI}$ is strongly normalizing if P is a bounded predicate.

3. Finally, using a technical lemma to postpone gc steps (Lem. 6.50), we obtain the main result (Thm. 6.51), which states that the reduction relation for the full LLSC is SN, as long as redex names have bounded height.

Strong normalization for bounded reduction without gc

Definition 6.46 (Measure of a labeled term)**.** We define the *size of a label* as follows:

$$\begin{array}{rcl} \|\mathbf{a}\| & \stackrel{\text{def}}{=} & 1 \\ \|\lceil\alpha\rceil\| = \|\lfloor\alpha\rfloor\| = \|\mathtt{db}(\alpha)\| & \stackrel{\text{def}}{=} & 1 + \|\alpha\| \\ \|\alpha\,\beta\| & \stackrel{\text{def}}{=} & \|\alpha\| + \|\beta\| \end{array}$$

Given a labeled term, its *measure* $\|t\|$ is defined as the sum of the sizes of all its labels:

$$\begin{array}{rcl} \|x^\alpha\| & \stackrel{\text{def}}{=} & \|\alpha\| \\ \|\lambda^\alpha_\Omega x.t\| & \stackrel{\text{def}}{=} & \|\alpha\| + \|t\| \\ \|@^\alpha(t,s)\| & \stackrel{\text{def}}{=} & \|\alpha\| + \|t\| + \|s\| \\ \|t[x\backslash s]_\Omega\| & \stackrel{\text{def}}{=} & \|t\| + \|s\| \end{array}$$

The measure of a context $\|\mathtt{C}\|$ is defined similarly, by declaring $\|\Box\| \stackrel{\text{def}}{=} 0$.

Lemma 6.47 (Properties of the measure of a term)**.** *The measure of a labeled term has the following properties:*

1. $\|\mathtt{C}\langle t\rangle\| = \|\mathtt{C}\| + \|t\|$
2. $\|\alpha : t\| = \|\alpha\| + \|t\|$

Proof. Both items are straightforward, by induction on $\mathtt{C}$ and t respectively. □

Lemma 6.48 (Labeled reduction without gc is increasing)**.** *If* $t \to_{\ell\, PI} s$ *then* $\|t\| < \|s\|$.

Proof. By induction on the context $\mathtt{C}$ under which the $\to_{\ell\ PI}$ redex in t is contracted. The inductive cases are easy by induction. The interesting case is when there is a step at the root:

1. db *step.* $t = @^\alpha((\lambda^\beta_\Omega x.t')\mathtt{L}, s') \xrightarrow{\mathtt{db}(\beta)}_{\ell\, PI} \alpha\lceil\mathtt{db}(\beta)\rceil : t'[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s']_\Omega\mathtt{L} = s$. Then:

$$\begin{array}{rcll} \|t\| & = & \|\alpha\| + \|\beta\| + \|t'\| + \|\mathtt{L}\| + \|s'\| & \text{by Lem. 6.47} \\ & < & \|\alpha\| + 2\|\beta\| + \|t'\| + \|\mathtt{L}\| + \|s'\| + 4 & \\ & = & \|s\| & \text{by Lem. 6.47} \end{array}$$

2. ls *step.* $t = \mathtt{C}\langle\!\langle x^\alpha\rangle\!\rangle[x\backslash t']_\Omega \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t')}_{\ell\, PI} \mathtt{C}\langle\alpha\bullet : t'\rangle[x\backslash t']_\Omega$. Then:

$$\begin{array}{rcll} \|t\| & = & \|\mathtt{C}\| + \|x^\alpha\| + \|t'\| & \text{by Lem. 6.47} \\ & = & \|\mathtt{C}\| + \|\alpha\| + \|t'\| & \\ & < & \|\mathtt{C}\| + \|\alpha\| + \|t'\| + \|t'\| + 1 & \\ & = & \|\mathtt{C}\| + \|\alpha\bullet\| + \|t'\| + \|t'\| & \\ & = & \|\mathtt{C}\| + \|\alpha\bullet : t'\| + \|t'\| & \text{by Lem. 6.47} \\ & = & \|s\| & \end{array}$$

□

Lemma 6.49 (Bounded reduction in the calculus without gc is strongly normalizing)**.** *Let P be a bounded predicate. Then $\to_{\ell PI}$ is* SN*.*

Proof. Note that we already know that $\to_{\ell PI}$ is weakly Church–Rosser, weakly normalizing, and increasing:

1. **WCR.** This is a consequence of Strong permutation (Prop. 6.30). Recall that in Prop. 6.30 all peaks $\overset{\mu}{\leftarrow}\overset{\nu}{\rightarrow}$ are closed with steps with the same name $\overset{\nu}{\rightarrow\!\!\!\!\rightarrow}\overset{\mu}{\leftarrow\!\!\!\!\leftarrow}$. So if we suppose that $P(\mu)$ and $P(\nu)$ hold for the names of the steps in the peak, the steps closing the diagram are also related by $\to_{\ell PI}$.

2. **WN.** This has been shown in Prop. 6.45.

3. **Inc.** This has been shown in Lem. 6.48.

Moreover, Klop-Nederpelt's Lemma (Lem. 2.22), asserts that WCR $\wedge$ WN $\wedge$ Inc $\implies$ SN, which concludes the proof. □

Strong normalization for bounded reduction with gc

In order to extend the strong normalization result to the full calculus, including the gc rule, we need the following technical lemma that allows to *postpone* gc steps.

Lemma 6.50 (Postponement of gc in the LLSC-calculus — ♣ Lem. A.80)**.** *Let $\rho : t \twoheadrightarrow_\ell s$ be a reduction sequence. Then there exists a term u and a reduction sequence $\sigma : t \twoheadrightarrow_{\ell\,\mathtt{db}\cup\mathtt{ls}} u \twoheadrightarrow_{\ell\,\mathtt{gc}} s$.*

Moreover, let $\#_\mu(\rho)$ denote the number of redexes named μ that are contracted along the reduction sequence ρ. Then:

1. *The number of* db *and* ls *redexes is preserved:*

$$\#_\mu(\rho) = \#_\mu(\sigma) \text{ if } \mu \text{ is the name of a } \mathtt{db} \text{ or } \mathtt{ls} \text{ redex}$$

2. *The number of* gc *redexes may increase:*

$$\#_\mu(\rho) \leqslant \#_\mu(\sigma) \text{ if } \mu \text{ is the name of a } \mathtt{gc} \text{ redex}$$

3. *The reduction σ contracts the same names as ρ:*

$$\#_\mu(\sigma) > 0 \implies \#_\mu(\rho) > 0 \text{ for any redex name } \mu$$

Proof. See the appendix for the full proof. □

The following is the main result of this section:

Theorem 6.51 (Bounded reduction in LLSC^P is strongly normalizing)**.** *Let P be a bounded predicate. Then labeled reduction $\to_{\ell P}$ is* SN*.*

Proof. Suppose given an infinite reduction sequence $\rho : t_0 \xrightarrow{\mu_1}_{\ell\ P} t_1 \dots \xrightarrow{\mu_n}_{\ell\ P} t_n \dots$. We show how to construct a second infinite reduction sequence: $\sigma : s_0 \xrightarrow{\nu_1}_{\ell\, P} s_1 \dots \xrightarrow{\nu_n}_{\ell\, P} s_n \dots$ consisting only of `db` and `ls` steps. This construction contradict the fact that LLSC^{PI} (without `gc`) is strongly normalizing for bounded families of labels, as already shown in Lem. 6.49.

First, note that there cannot be an infinite reduction sequence $t \twoheadrightarrow_{\ell\ P} \dots$ consisting only of `gc` steps, since $\to_{\ell\,\mathtt{gc}}$ is SN (as it strictly decreases the size of the term). To alleviate notation, given $i \leqslant j$, let $t_i \xrightarrow{\mu}\!\!\!\!\twoheadrightarrow_{\ell\ } t_j$ stand for $t_i \xrightarrow{\mu_{i+1}}_{\ell} t_{i+1} \dots \xrightarrow{\mu_j}_{\ell} t_j$. To construct σ, proceed by induction on n, with the following invariant. At the n-th step, for $n \geqslant 1$, we will have built:

- a reduction sequence $s_0 \xrightarrow{\nu}\!\!\!\!\twoheadrightarrow_{\ell\ P} s_n$ of length exactly n;
- consisting of only `db` and `ls` steps;
- and such that $t_0 \twoheadrightarrow_{\ell\ \mathtt{db}\cup\mathtt{ls}} s_n \twoheadrightarrow_{\ell\,\mathtt{gc}} t_{k_n}$ for some $k_n \geqslant 0$.

We prove the base case and inductive step:

- *Base case,* $n = 1$. By the previous remark, there cannot be an infinite sequence of `gc` steps, so we can choose $k_1 > 0$ such that $t_0 \xrightarrow{\mu}\!\!\!\!\twoheadrightarrow_{\ell\ \mathtt{gc}} \xrightarrow{\mu_{k_1}}_{\ell\,\mathtt{db}\cup\mathtt{ls}} t_{k_1}$. By postponement (Lem. 6.50) there exists a term s_1 such that $t_0 \xrightarrow{\mu_{k_1}}_{\ell\,\mathtt{db}\cup\mathtt{ls}} s_1 \twoheadrightarrow_{\ell\,\mathtt{gc}} t_{k_1}$. Take $s_0 := t_0$ and $\nu_1 := \mu_{k_1}$.
- *Induction, "*$n \implies n+1$*".* We already have $t_0 \twoheadrightarrow_{\ell\ \mathtt{db}\cup\mathtt{ls}} s_n \twoheadrightarrow_{\ell\,\mathtt{gc}} t_{k_n}$. As before, since there cannot be an infinite sequence of `gc` steps, we can choose $k_{n+1} > k_n$ such that $s_n \twoheadrightarrow_{\ell\,\mathtt{gc}} t_{k_n} \twoheadrightarrow_{\ell\,\mathtt{gc}} \xrightarrow{\mu_{k_{n+1}}}_{\ell\,\mathtt{db}\cup\mathtt{ls}} t_{k_{n+1}}$. By postponement, there exists a term s_{n+1} such that $s_n \xrightarrow{\mu_{k_{n+1}}}_{\ell\,\mathtt{db}\cup\mathtt{ls}} s_{n+1} \twoheadrightarrow_{\ell\ \mathtt{gc}} t_{k_{n+1}}$. Take $\nu_{n+1} := \mu_{k_{n+1}}$.

Finally, since $\nu_n = \mu_{k_n}$, it is clear that $P(\nu_n)$ must hold for all n. □

6.3.5 Confluence

In this section we give two proofs that the LLSC is confluent. Both proofs are consequences of previous facts that we have already established. The first proof is based on purely syntactical methods, while the second one relies on residual theory.

Lemma 6.52 (Bounded reduction in the LLSC^P is confluent)**.** *Let P be a bounded predicate. Then the rewriting relation $\to_{\ell\, P}$ is Church–Rosser.*

Proof. By Newman's Lemma (Lem. 2.20), is suffices to show that $\to_{\ell\ P}$ is SN (*cf.* Thm. 6.51) and WCR (*cf.* Prop. 6.30). □

Theorem 6.53 (The LLSC is confluent)**.** *The rewriting relation $\to_\ell$ is Church–Rosser.*

Proof.

First proof. If $\rho : t \twoheadrightarrow_{\ell\ } s$ and $\sigma : t \twoheadrightarrow_{\ell\ } u$, define $P(\mu)$ to hold iff μ is the name of redex contracted in ρ or σ. Since the number of such labels is finite, P is bounded and by the previous result (Lem. 6.52) we conclude.

Second proof. We have proved that the LLSC is an orthogonal axiomatic rewriting system (Prop. 6.32). Moreover, orthogonal axiomatic rewriting systems enjoy algebraic confluence (Coro. 2.56), which is a strong form of confluence. □

Chapter 7

Applications of the Labeled Linear Substitution Calculus

7.1 Introduction

In his PhD thesis [109], Jean-Jacques Lévy defined a notion of *redex family* for the λ-calculus, with the goal of understanding what an optimal implementation of the λ-calculus would look like. In a straightforward implementation of the λ-calculus, each step in the implementation corresponds to a single β-step. If the implementation has some non-trivial mechanism of *sharing*, however, each step in the implementation may correspond to the simultaneous contraction of many β-steps. Lévy proved that if an implementation of the λ-calculus contracts, in each step, a maximal set of β-steps belonging to the same redex family, in such a way that at least one β-step is needed, then the implementation is optimal. Later, in [78], Lévy and Gérard Huet studied standardization and normalization in the setting of orthogonal term rewriting systems. In particular, they showed that if a reduction strategy repeatedly contracts a *needed* step, the strategy reaches a result whenever possible, *i.e.* it is normalizing.

In their work [61], John Glauert and Zurab Khasidashvili generalized the results of optimality and normalization to any abstract rewriting system, provided that it comes equipped with well-behaved notions of *residuals* and *redex families*. The abstract axiomatic structure encapsulating all the desired properties is called a *Deterministic Family Structure* in [61].

In the previous chapter, we have defined a Lévy labeled calculus LLSC (Section 6.2) and we have proved that it enjoys a number of properties (Section 6.3). In this chapter, the Lévy labeled calculus LLSC is used as a tool to prove properties about the usual, unlabeled, LSC. In particular, the labeled calculus LLSC is used to show that the unlabeled LSC forms a Deterministic Family Structure, and consequently to obtain optimality, standardization, and normalization results. Most of the results concern the LSC without gc.

7.1.1 Our Work

This chapter is the result of collaboration with Eduardo Bonelli and it is structured as follows. We highlight in boldface what we consider to be the main contributions:

1. In the previous chapter (Rem. 6.8), we have already seen that the LSC with the gc rule does not enjoy the property known as *redex stability*. In Section 7.2, using the LLSC as a tool, we show that, however, **the LSC without the** gc **rule does enjoy redex stability** (Prop. 7.1).

2. In Section 7.3 we recall Glauert and Khasidashvili's notion of Deterministic Family Structure (DFS). Then, using the LLSC as a tool, we prove that **the LSC without** gc **forms a Deterministic Family Structure** (Thm. 7.13).

   When the properties of Deterministic Family Structures are translated into the language of Lévy labels, the statement that the LSC forms a DFS can essentially be summarized as the statement that the LLSC verifies the properties 1–7 in the Bestiary of Chapter 6 Section 6.2.1. The property of *Termination* for the LLSC corresponds to the property usually known as *Generalized Finite Developments* or *Finite Family Developments*. The property of *Contribution* (Prop. 7.12) is not immediate and relies on Finite Family Developments.

3. In Section 7.4 we recall Lévy's *optimality* result for the λ-calculus (Thm. 7.17), we also review Glauert and Khasidashvili's abstract optimality result (Thm. 7.24), and, as a corollary, we derive an optimality result for the LSC without gc.

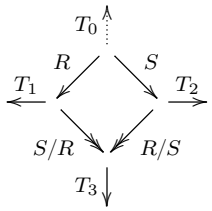
4. In Section 7.5 we recall the problem of *standardization*, and we propose a **standardization procedure for Deterministic Family Structures** (Prop. 7.39), inspired on a standardization result by Klop. As a corollary, we obtain a standardization result for the LSC without gc (Coro. 7.43).

5. In Section 7.6 we recall the notion of *normalization*, and we prove a **normalization result for Deterministic Family Structures** (Prop. 7.54), giving sufficient conditions under which a reduction strategy is normalizing. As a corollary, we conclude that, in the LSC without gc the call-by-name strategy (Coro. 7.56) and a variant of the call-by-need strategy (Coro. 7.59) are normalizing.

Moreover, in Section 8.2 in the Conclusion (Chapter 8), we discuss an open problem regarding the definition of an *extraction procedure* for the LSC. We propose an extraction procedure and we state two unproved conjectures about it.

7.2 Stability

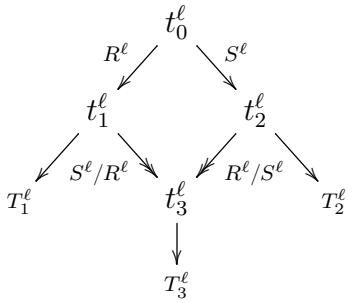
Recall that an orthogonal axiomatic rewriting system is said to verify the Stability property (Def. 6.7) if any two steps that have a common residual also have a common ancestor. Graphically:



In Rem. 6.8 we observed that the LSC with the gc rule does not have the Stability property. On the other hand:

Proposition 7.1. *The LLSC without* gc *has the Stability property.*

Proof. Let R, S, T_1, T_2, and T_3 be five steps such that $T_3 \in T_1/(S/R) \cap T_2/(R/S)$. Consider an initially labeled variant t_0^ℓ of the term at the peak of the diagram, *i.e.* the source of R and S. Consider also all the corresponding labeled variants R^ℓ, S^ℓ, T_1^ℓ, T_2^ℓ, and T_3^ℓ of R, S, T_1, T_2, and T_3 respectively:



Since T_3^ℓ is a residual of both T_1^ℓ and T_2^ℓ, by the **Copy** property (Lem. 6.33), we have that T_1^ℓ, T_2^ℓ and T_3^ℓ have the same name. We consider two cases, depending on whether T_1^ℓ and T_2^ℓ have an ancestor in t_0^ℓ or not.

1. *If T_1^ℓ has an ancestor in t_0^ℓ.* Then there is a step T_0^ℓ such that $T_1^\ell \in T_0^\ell/R$, so T_0^ℓ has the same name as T_1^ℓ by the **Copy** property (Lem. 6.33). Moreover, since T_0^ℓ and T_2^ℓ have the same name and t_0^ℓ is initially labeled, we have that $T_2^\ell \in \hat{T}_0^\ell/R$ using the converse of the **Copy** property (Lem. 6.33).

2. *If T_2^ℓ has an ancestor in t_0^ℓ.* Analogous to the previous case.

3. *If T_1^ℓ and T_2^ℓ do not have an ancestor in t_0^ℓ.* We argue that this case is impossible. Since T_1^ℓ has no ancestor, by definition, it is a created redex. By the **Creation** property (Prop. 6.41) the name of R^ℓ directly contributes to the name of T_1^ℓ, so there are three possibilities:

$$\begin{array}{ccc} \mathtt{db}(\mathbf{b}) & \overset{\mathsf{Name}}{\hookrightarrow_1} & \mathtt{db}(\mathbf{a}\lceil\mathtt{db}(\mathbf{b})\rceil\mathbf{c}) \\ \mathtt{db}(\mathbf{b}) & \overset{\mathsf{Name}}{\hookrightarrow_1} & \mathbf{a} \bullet \lfloor\mathtt{db}(\mathbf{b})\rfloor \\ \mathbf{a} \bullet \mathbf{b} & \overset{\mathsf{Name}}{\hookrightarrow_1} & \mathtt{db}(\mathbf{a} \bullet \mathbf{b}) \end{array}$$

   Note that since **a**, **b**, and **c** are initial labels, the name of T_1^ℓ uniquely determines the name of R^ℓ. Symmetrically, T_2^ℓ is created by S^ℓ, and the name of S^ℓ is uniquely determined by the name of T_2^ℓ.

   Finally, since T_1^ℓ and T_2^ℓ both have the same name, and they uniquely determine the name of their ancestors, the names of R^ℓ and S^ℓ must coincide. Moreover t_0^ℓ is initially labeled, so by the **Initial** property (Lem. 6.19) we have that $R^\ell = S^\ell$, and in particular $R = S$, which is a contradiction.

□

7.3 Redex Families

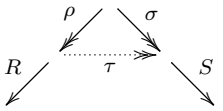
In this section we study *redex families* in the LSC without gc. First, we review the definition of Deterministic Family Structure (DFS), proposed by Glauert and Khasidashvili in [61]. A Deterministic Family Structure is an abstract rewriting system that verifies a number of particular axioms. These axioms essentially request that there is a well-behaved notion of redex family, which allows one to state and prove a generalized version of Lévy's optimality result in a framework that abstracts away the low level details of Lévy labels. Second, we prove that the LSC without gc forms a DFS (Thm. 7.13). This construction relies crucially on the labeled LSC that we have defined in previous sections (Def. 6.6).

We begin by recalling some notation and definitions:

- An axiomatic rewriting system has the *unique ancestor* (UA) property (Def. 2.31) if a step has at most one ancestor, that is, whenever $R_1 \langle S \rangle R$ and $R_2 \langle S \rangle R$ then $R_1 = R_2$.
- An axiomatic rewriting system has the *acyclicity* property (Def. 2.31) if two steps cannot erase each other, that is, whenever $R \neq S$ and $R/S = \varnothing$ then $S/R \neq \varnothing$.
- In an abstract rewriting system, a *redex with history* or *hredex* for short (see Section 6.1.1) is a non-empty derivation. We are usually interested in the last step of the hredex, so hredexes are typically written as of the form ρR where ρ is a possibly empty derivation and R is a composable step. The set of hredexes whose source is an object x is denoted by $\mathsf{Hist}(x)$.
- In an orthogonal axiomatic rewriting system we write $\rho \equiv \sigma$ whenever ρ and σ are permutation equivalent derivations (Def. 2.40).

We also give a formal definition of *copy*:

Definition 7.2 (Copy relation)**.** Let ρR and σS be coinitial hredexes in any orthogonal axiomatic rewriting system. We say that σS is a *copy* of ρR, written $\rho R \leqslant \sigma S$ if there exists a derivation τ such that $\rho\tau \equiv \sigma$ and $R \langle \tau \rangle S$. Graphically:



Definition 7.3 (Deterministic Family Structure)**.** A *Deterministic Residual Structure* (DRS) is an orthogonal axiomatic rewriting system (*cf.* Def. 2.39) that moreover verifies the unique ancestor (UA) and acyclicity properties.

A *Deterministic Family Structure* (DFS) is a triple $\langle \mathcal{A}, \simeq, \hookrightarrow \rangle$, where $\mathcal{A}$ is a Deterministic Residual Structure, $\simeq$ is an equivalence relation between coinitial hredexes whose equivalence classes are called *families*, and $\hookrightarrow$ is a binary relation of *contribution* between coinitial families. Two families are declared to be coinitial if their representatives are coinitial. The family of an hredex ρR is written $\mathsf{Fam}_{\simeq}(\rho R)$. Moreover, the following axioms hold:

1. Initial. If R, S are different coinitial steps, then $\mathsf{Fam}_{\simeq}(R) \neq \mathsf{Fam}_{\simeq}(S)$.
2. Copy. The inclusion $(\leqslant) \subseteq (\simeq)$ holds.
3. Finite Family Developments (FFD). Any derivation that contracts hredexes of a finite number of families is finite. More precisely, there cannot be an infinite derivation $R_1 R_2 \dots R_n \dots$ such that the set $\{\mathsf{Fam}_{\simeq}(R_1 \dots R_n) \mid n \in \mathbb{N}\}$ is finite.
4. Creation. If ρR is an hredex and R creates S, then $\mathsf{Fam}_{\simeq}(\rho R) \hookrightarrow \mathsf{Fam}_{\simeq}(\rho R S)$.
5. Contribution. Given any two coinitial families $\phi_1, \phi_2 \in \mathsf{Hist}(t)/\simeq$, the relation $\phi_1 \hookrightarrow \phi_2$ holds if and only if for every hredex $\sigma S \in \phi_2$, there is an hredex $\rho R \in \phi_1$ such that ρR is a prefix of σ (*i.e.* $\sigma = \rho R \sigma'$).

Note that the formal requirements imposed by the definition of a DFS correspond to the informal principles 2–7 in the Bestiary of Chapter 6 Section 6.2.1. While the Bestiary states these principles using redex *names*, the notion of DFS states them using the more abstract notion of family.

The axioms Initial, Copy, and Creation correspond to the principles 2, 3, and 4, respectively, in the Bestiary. Practically speaking, once one has defined an appropriate Lévy labeling for a calculus, the proof that these axioms are fulfilled should be a technical but direct proof. The axioms Finite Family Developments and Contribution correspond to the principles 5 and 7, respectively, in the Bestiary, and their proof is typically non-trivial.

Let us briefly recapitulate notation. So far we have introduced various axiomatic structures to deal with rewriting systems abstractly. The following table summarizes the relation between these structures, from most general to most restrictive:

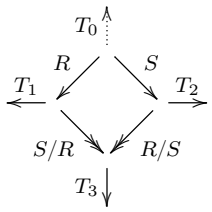
Abstract rewriting system	(ARS)	Def. 2.2	Objects and steps.
Axiomatic rewriting system	(AxRS)	Def. 2.30	ARS + residuals.
Orthogonal axiomatic rewriting system	(OAxRS)	Def. 2.39	AxRS + AE + FR + FD + SO.
Deterministic Residual Structure	(DRS)	Def. 7.3	OAxRS + UA + acyclicity.
Deterministic Family Structure	(DFS)	Def. 7.3	DRS + redex families.

The LSC with gc is not a Deterministic Family Structure

We have already shown in Rem. 6.8 that the full LSC (with gc) does not enjoy the Stability property. A consequence of this fact is that the LSC with gc does not form a Deterministic Family Structure. To see this it suffices to prove the following proposition, which can already be found in Glauert and Khasidashvili's work [61, Lemma 4.1].

Proposition 7.4. *If $\langle \mathcal{A}, \simeq, \hookrightarrow \rangle$ is a DFS then $\mathcal{A}$ has the Stability property.*

Proof. Consider a diagram as in the definition of Stability:



That is, we have that $R \neq S$, where $T_3 \in T_1/(S/R)$ and $T_3 \in T_2/(R/S)$. Let us show that T_1 and T_2 have a common ancestor T_0. Note that $R\,T_1 \leqslant (R \sqcup S)\,T_3$ and $S\,T_2 \leqslant (R \sqcup S)\,T_3$ by definition of the copy relation. By the Copy axiom we have that $R\,T_1$ and $S\,T_2$ are in the same family:

$$R\,T_1 \simeq (R \sqcup S)\,T_3 \simeq S\,T_2$$

We consider three cases, depending on whether $\mathsf{Fam}_{\simeq}(R)$ contributes to $\mathsf{Fam}_{\simeq}(RT_1)$, or $\mathsf{Fam}_{\simeq}(S)$ contributes to $\mathsf{Fam}_{\simeq}(ST_2)$, or none of these happens.

1. **If $\mathsf{Fam}_{\simeq}(R) \hookrightarrow \mathsf{Fam}_{\simeq}(RT_1)$.** We argue that this case is impossible. By the Contribution axiom, since $\mathsf{Fam}_{\simeq}(R) \hookrightarrow \mathsf{Fam}_{\simeq}(RT_1)$ and $S\,T_2 \in \mathsf{Fam}_{\simeq}(RT_1)$, we must have that $S \in \mathsf{Fam}_{\simeq}(R)$. This means that $R \simeq S$, which contradicts the Initial axiom, since R and S are different coinitial steps.

2. **If $\mathsf{Fam}_{\simeq}(S) \hookrightarrow \mathsf{Fam}_{\simeq}(ST_2)$.** This case is impossible, by a symmetric argument as in the first case.

3. **If $\neg(\mathsf{Fam}_{\simeq}(R) \hookrightarrow \mathsf{Fam}_{\simeq}(RT_1))$ and $\neg(\mathsf{Fam}_{\simeq}(S) \hookrightarrow \mathsf{Fam}_{\simeq}(ST_2))$.** Then by the contrapositive of the Creation axiom, we have that R does not create T_1 so it has an ancestor, *i.e.* there is a step T_0 such that $T_1 \in T_0/R$. Similarly, T_2 has an ancestor before S, *i.e.* there is a step T_0' such that $T_2 \in T_0'/S$. Note that T_0 and T_0' are both ancestors of T_3, so by the unique ancestor property we have that $T_0 = T_0'$, which concludes the proof.

□

The LSC without gc forms a Deterministic Family Structure

This section is devoted to proving that the LSC without gc forms a Deterministic Family Structure. By definition, a DFS is a triple $\langle \mathcal{A}, \simeq, \hookrightarrow \rangle$ where $\mathcal{A}$ is a Deterministic Residual Structure, so we start by showing that it forms a DRS.

Proposition 7.5 (LSC is a DRS)**.** *The LSC with* gc *forms a Deterministic Residual Structure.*

Proof. In [8], the LSC has already been shown to form an orthogonal axiomatic rewriting system. (We also give a proof of this fact in Prop. 6.3). It remains to be checked that the LSC has the unique ancestor and acyclicity properties:

1. *Unique ancestor* (UA). Let $R_1 \langle S \rangle R$ and $R_2 \langle S \rangle R$ in the LSC. Let $R_1^\ell, R_2^\ell, R^\ell, S^\ell$ be labeled variants of R_1, R_2, R, S respectively, such that the source of R_1^ℓ, R_2^ℓ, and S^ℓ coincides and is initially labeled, and such that moreover the residual relations $R_1^\ell \langle S^\ell \rangle R^\ell$ and $R_2^\ell \langle S^\ell \rangle R^\ell$ hold in the LLSC.

   By the **Copy** property (Lem. 6.33), R_1^ℓ and R^ℓ have the same name. Similarly, R_2^ℓ and R^ℓ have the same name, so in fact R_1^ℓ and R_2^ℓ have the same name. Then by Lem. 6.19 $R_1^\ell = R_2^\ell$, so $R_1 = R_2$, as required.

2. *Acyclicity.* Let R and S be different steps such that $R/S = \varnothing$. Since only gc steps may erase other steps, S must be a gc step of the form $S : \mathtt{C}\langle t[x\backslash s]\rangle \to_{\mathsf{LSC}} \mathtt{C}\langle t\rangle$ and the anchor of the redex contracted by R lies inside s, *i.e.* R is a step of the form $R : \mathtt{C}\langle t[x\backslash s]\rangle \to_{\mathsf{LSC}} \mathtt{C}\langle t[x\backslash s']\rangle$. Then S/R is the singleton $\{S'\}$ where $S' : \mathtt{C}\langle t[x\backslash s']\rangle \to_{\mathsf{LSC}} \mathtt{C}\langle t\rangle$. So in fact $S/R \neq \varnothing$.

□

To show that the LSC without gc forms a DFS, we need some preliminary lemmas. Recall that $(\overset{\mathsf{Name}}{\hookrightarrow})$ stands for the relation of *name contribution* defined in Def. 6.39.

Lemma 7.6. *Let ϕ be a labeling morphism and let μ and ν be (non-*gc*) redex names. Then $\mu \overset{\mathsf{Name}}{\hookrightarrow} \nu$ implies $\phi(\mu) \overset{\mathsf{Name}}{\hookrightarrow} \phi(\nu)$.*

Proof. Recall that $\overset{\mathsf{Name}}{\hookrightarrow}$ is the transitive closure of $\overset{\mathsf{Name}}{\hookrightarrow}_1$, so it suffices to check that the property holds for one step of $\overset{\mathsf{Name}}{\hookrightarrow}_1$. By cases on the rules defining the relation $\overset{\mathsf{Name}}{\hookrightarrow}_1$:

1. If $\mu = \mathtt{db}(\beta) \overset{\mathsf{Name}}{\hookrightarrow}_1 \mathtt{db}(\alpha \lceil \mathtt{db}(\beta) \rceil \gamma) = \nu$ then

$$\phi(\mu) = \mathtt{db}(\phi(\beta)) \overset{\mathsf{Name}}{\hookrightarrow}_1 \mathtt{db}(\phi(\alpha) \lceil \mathtt{db}(\phi(\beta)) \rceil \phi(\gamma)) = \phi(\nu)$$

2. If $\mu = \mathtt{db}(\beta) \overset{\mathsf{Name}}{\hookrightarrow}_1 \alpha \bullet \lfloor \mathtt{db}(\beta) \rfloor = \nu$ where α is an atomic label, then

$$\phi(\mu) = \mathtt{db}(\phi(\beta)) \overset{\mathsf{Name}}{\hookrightarrow}_1 \downarrow (\phi(\alpha)) \bullet \lfloor \mathtt{db}(\phi(\beta)) \rfloor = \phi(\nu)$$

3. If $\mu = \beta \bullet \gamma \overset{\mathsf{Name}}{\hookrightarrow}_1 \mathtt{db}(\alpha\beta \bullet \gamma\delta) = \nu$ where β and γ are atomic labels then

$$\phi(\mu) = \downarrow (\phi(\beta)) \bullet \uparrow (\phi(\gamma)) \overset{\mathsf{Name}}{\hookrightarrow}_1 \mathtt{db}(\phi(\alpha)\phi(\beta) \bullet \phi(\gamma)\phi(\delta)) = \phi(\nu)$$

□

The LSC without gc will be shown to form a DFS with the following notions of redex family and contribution.

Definition 7.7 (Redex families in the LSC without gc)**.** Let ρR and σS be coinitial hredexes in the LSC without gc. Let $\rho^\ell R^\ell$ and $\sigma^\ell S^\ell$ be initially labeled variants of ρR and σS respectively, starting on the same initially labeled term. Let μ be the name of R^ℓ and let ν be the name of S^ℓ. Then:

- **Family relation.** We define $\rho R \overset{\mathsf{Fam}}{\simeq} \sigma S$ to hold if and only if $\mu = \nu$.
- **Contribution relation.** We define $\rho R \overset{\mathsf{Fam}}{\hookrightarrow} \sigma S$ to hold if and only if $\mu \overset{\mathsf{Name}}{\hookrightarrow} \nu$.

Example 7.8 (Redex families and contribution)**.** *In the following diagram:*

$$\begin{array}{ccccc} ((\lambda x.x)\, y)[y\backslash z] & \xrightarrow{R} & x[x\backslash y][y\backslash z] & \xrightarrow{S} & y[x\backslash y][y\backslash z] \\ \downarrow{\scriptstyle T_1} & & \downarrow{\scriptstyle T_2} & & \\ ((\lambda x.x)\, z)[y\backslash z] & & x[x\backslash z][y\backslash z] & & \end{array}$$

We have that $T_1 \overset{\mathsf{Fam}}{\simeq} RT_2$ *and that* $R \overset{\mathsf{Fam}}{\hookrightarrow} RS$*. This can be justified starting from an initially labeled variant of* $((\lambda x.x)\, y)[y\backslash z]$ *and noting that the names of* T_1 *and* RT_2 *are both* $\mathbf{d} \bullet \mathbf{e}$*, and that the name of* R *contributes to the name of* RS*, that is,* $\mathtt{db}(\mathbf{b}) \overset{\mathsf{Name}}{\hookrightarrow} \mathbf{c} \bullet \lfloor \mathtt{db}(\mathbf{b}) \rfloor$*:*

$$\begin{array}{ccccc} @^{\mathbf{a}}(\lambda^{\mathbf{b}} x.x^{\mathbf{c}}, y^{\mathbf{d}})[y\backslash z^{\mathbf{e}}] & \xrightarrow{\mathtt{db}(\mathbf{b})} & x^{\mathbf{a}\lceil \mathtt{db}(\mathbf{b})\rceil \mathbf{c}}[x\backslash y^{\lfloor \mathtt{db}(\mathbf{b})\rfloor \mathbf{d}}][y\backslash z^{\mathbf{e}}] & \xrightarrow{\mathbf{c}\bullet\lfloor \mathtt{db}(\mathbf{b})\rfloor} & y^{\mathbf{a}\lceil \mathtt{db}(\mathbf{b})\rceil \mathbf{c}\bullet\lfloor \mathtt{db}(\mathbf{b})\rfloor \mathbf{d}}[x\backslash y^{\lfloor \mathtt{db}(\mathbf{b})\rfloor \mathbf{d}}][y\backslash z^{\mathbf{e}}] \\ \downarrow{\scriptstyle \mathbf{d}\bullet\mathbf{e}} & & \downarrow{\scriptstyle \mathbf{d}\bullet\mathbf{e}} & & \\ @^{\mathbf{a}}(\lambda^{\mathbf{b}} x.x^{\mathbf{c}}, z^{\mathbf{d}\bullet\mathbf{e}})[y\backslash z^{\mathbf{e}}] & & x^{\mathbf{a}\lceil \mathtt{db}(\mathbf{b})\rceil \mathbf{c}}[x\backslash z^{\lfloor \mathtt{db}(\mathbf{b})\rfloor \mathbf{d}\bullet\mathbf{e}}][y\backslash z^{\mathbf{e}}] & & \end{array}$$

Proposition 7.9 (Redex families are well-defined)**.** *The relations* $(\overset{\mathsf{Fam}}{\simeq})$ *and* $(\overset{\mathsf{Fam}}{\hookrightarrow})$ *are well-defined, in the sense that they do not depend on the choice of the initial labeling.*

Proof. Let ρR and σS be coinitial hredexes in the LSC without gc. Let $\rho_1^\ell R_1^\ell$ and $\sigma_1^\ell S_1^\ell$ be initially labeled variants of ρR and σS starting on the same initially labeled term t_1^ℓ, and let $\rho_2^\ell R_2^\ell$ and $\sigma_2^\ell S_2^\ell$ be initially labeled variants of ρR and σS starting on a possibly different initially labeled term t_2^ℓ. Let $\mu_1, \nu_1, \mu_2, \nu_2$ be the names of $R_1^\ell, S_1^\ell, R_2^\ell, S_2^\ell$ respectively. To show that $(\overset{\mathsf{Fam}}{\simeq})$ and $(\overset{\mathsf{Fam}}{\hookrightarrow})$ are well-defined it suffices to prove that:

1. $(\mu_1 = \nu_1) \iff (\mu_2 = \nu_2)$
2. $(\mu_1 \overset{\mathsf{Name}}{\hookrightarrow} \nu_1) \iff (\mu_2 \overset{\mathsf{Name}}{\hookrightarrow} \nu_2)$

In both cases, to prove the equivalence it suffices to show the implication in one direction, since the other one is symmetric.

Note that each subterm of t_1^ℓ is labeled with a different initial label, so there is a labeling morphism $\phi : \mathcal{L} \to \mathcal{L}$ such that $\phi(t_1^\ell) = t_2^\ell$. Since labeling morphisms are functorial (Prop. 6.29) we have that $\phi(\rho_1^\ell R_1^\ell) = \rho_2^\ell R_2^\ell$ and similarly $\phi(\sigma_1^\ell S_1^\ell) = \sigma_2^\ell S_2^\ell$. This means that $\phi(\mu_1) = \phi(R_1^\ell) = R_2^\ell = \mu_2$ and $\phi(\nu_1) = \phi(S_1^\ell) = S_2^\ell = \nu_2$.

Now, if $\mu_1 = \nu_1$ then $\mu_2 = \phi(\mu_1) = \phi(\nu_1) = \nu_2$, which proves that $(\overset{\mathsf{Fam}}{\simeq})$ is well-defined.

On the other hand, if $\mu_1 \overset{\mathsf{Name}}{\hookrightarrow} \nu_1$ then using Lem. 7.6 we have that $\mu_2 = \phi(\mu_1) \overset{\mathsf{Name}}{\hookrightarrow} \phi(\nu_1) = \nu_2$, which proves that $(\overset{\mathsf{Fam}}{\hookrightarrow})$ is well-defined. □

It is immediate to check that the family relation $(\overset{\mathsf{Fam}}{\simeq})$ is an equivalence relation. This reduces to the fact that equality of redex names is in turn an equivalence relation. As in the abstract definition of a DFS, equivalence classes of $\overset{\mathsf{Fam}}{\simeq}$ are called *families*, and $\mathsf{Fam}_{\simeq}(\rho R)$ stands for the family of ρR.

Definition 7.10 (Family contribution relation)**.** Given two coinitial families ϕ_1, ϕ_2, we say that ϕ_1 contributes to ϕ_2, and write $\phi_1 \overset{\mathsf{Fam}}{\hookrightarrow} \phi_2$ if and only if given $\rho R \in \phi_1$ and $\sigma S \in \phi_2$ the condition $\rho R \overset{\mathsf{Fam}}{\hookrightarrow} \sigma S$ holds.

Note that, by abuse of notation, we write $\overset{\mathsf{Fam}}{\hookrightarrow}$ both for the contribution relation between hredexes and for the contribution relation between families.

Proposition 7.11 (Contribution is well-defined)**.** *The contribution relation $\overset{\mathsf{Fam}}{\hookrightarrow}$ between families is well-defined, in the sense that it does not depend on the choice of representative.*

Proof. Let t be a term and let $\phi, \phi' \in \mathsf{Hist}(t)/\overset{\mathsf{Fam}}{\simeq}$ be two families. Let us show that the condition $\phi \overset{\mathsf{Fam}}{\hookrightarrow} \phi'$ does not depend upon the choice of the representative of the equivalence classes of ϕ and ϕ'. Indeed, let $\rho_1 R_1, \rho_2 R_2 \in \phi$, and let $\sigma_1 S_1, \sigma_2 S_2 \in \phi'$. Let us show that $\rho_1 R_1 \overset{\mathsf{Fam}}{\hookrightarrow} \sigma_1 S_1$ if and only if $\rho_2 R_2 \overset{\mathsf{Fam}}{\hookrightarrow} \sigma_2 S_2$.

Let t^ℓ be an initially labelled variant of the source term t, and consider labelled variants $\rho_1^\ell R_1^\ell$, $\rho_2^\ell R_2^\ell$, $\sigma_1^\ell S_1^\ell$, and $\sigma_2^\ell S_2^\ell$ of $\rho_1 R_1$, $\rho_2 R_2$, $\sigma_1 S_1$, and $\sigma_2 S_2$ respectively. Moreover, let μ_1, μ_2, ν_1, and ν_2 be the names of R_1^ℓ, R_2^ℓ, S_1^ℓ, and S_2^ℓ respectively.

Then, by definition of being in the same family, $\rho_1 R_1 \overset{\mathsf{Fam}}{\simeq} \rho_2 R_2$ means that $\mu_1 = \mu_2$. Similarly, $\sigma_1 S_1 \overset{\mathsf{Fam}}{\simeq} \sigma_2 S_2$ means that $\nu_1 = \nu_2$. Then:

$$
\begin{array}{lll}
 & \rho_1 R_1 \overset{\mathsf{Fam}}{\hookrightarrow} \sigma_1 S_1 & \\
\text{if and only if} & \mu_1 \overset{\mathsf{Name}}{\hookrightarrow} \nu_1 & \text{by definition of } \overset{\mathsf{Fam}}{\hookrightarrow} \\
\text{if and only if} & \mu_2 \overset{\mathsf{Name}}{\hookrightarrow} \nu_2 & \text{since } \mu_1 = \mu_2 \text{ and } \nu_1 = \nu_2 \\
\text{if and only if} & \rho_2 R_2 \overset{\mathsf{Fam}}{\hookrightarrow} \sigma_2 S_2 & \text{by definition of } \overset{\mathsf{Fam}}{\hookrightarrow}
\end{array}
$$

Hence $\overset{\mathsf{Fam}}{\hookrightarrow}$ is well defined on the set of families. □

In the following proposition we state and prove the Contribution axiom for the LSC without gc. The proof relies on various quite technical lemmas whose statement and proof can be found in Section A.4.1 of the appendix.

Proposition 7.12 (Contribution axiom for the LSC without gc)**.** *Let $\phi_1, \phi_2 \in \mathsf{Hist}(t)/\overset{\mathsf{Fam}}{\simeq}$ be coinitial families in the LSC without* gc*. Then the following propositions are logically equivalent:*

1. **Syntactic contribution.** $\phi_1 \overset{\mathsf{Fam}}{\hookrightarrow} \phi_2$.
2. **Semantic contribution.** *For every hredex $\sigma S \in \phi_2$, there is an hredex $\rho R \in \phi_1$ such that ρR is a prefix of σ.*

Proof. Let us show each direction of the implication. We refer to the implication $(1 \implies 2)$ as *correctness* and to the implication $(2 \implies 1)$ as *completeness*.

($\Rightarrow$) **Correctness.** Let $\sigma S \in \phi_2$ be an hredex. Consider an initially labelled variant t_0^ℓ of t, and the labelled variant $\sigma^\ell S^\ell$ of σS whose source is t_0^ℓ. Let $t_1^\ell = \mathsf{tgt}(\sigma^\ell) = \mathsf{src}(S^\ell)$. Moreover, let $\tau T \in \phi_1$, and consider the labelled variant $\tau^\ell T^\ell$ of τT whose source is t_0^ℓ.

Let ν be the name of S^ℓ, and let μ be the name of T^ℓ. Since $\phi_1 \overset{\mathsf{Fam}}{\hookrightarrow} \phi_2$ we have, by definition, that $\mu \overset{\mathsf{Name}}{\hookrightarrow} \nu$. It can be seen that names contributing to a step must occur in

the source (by Lem. A.83 in Section A.4.1 of the appendix). This means that there must exist a label α decorating a subterm of t_1^ℓ such that μ is a sublabel of α. By Lem. A.94 (in Section A.4.1 of the appendix), this entails that there must exist a step in σ^ℓ whose name is μ. This means that $\sigma^\ell = \rho^\ell R^\ell \upsilon^\ell$ where the name of R^ℓ is μ, hence $\rho R \overset{\mathsf{Fam}}{\simeq} \tau T$ and so $\sigma = \rho R \upsilon$ where $\rho R \in \phi_1$, as wanted.

($\Leftarrow$) **Completeness.** Let us show that $\phi_1 \overset{\mathsf{Fam}}{\hookrightarrow} \phi_2$. Let $\sigma S \in \phi_2$ be an hredex. Consider an initially labelled variant t_0^ℓ of t, and consider the labelled variant $\sigma^\ell S^\ell$ whose source is t_0^ℓ. Let ν be the name of S^ℓ. Let P be the predicate on redex names such that $P(\mu)$ holds if and only if $\mu \overset{\mathsf{Name}}{\hookrightarrow} \nu$. Observe that P is a bounded predicate, since by Rem. 6.40 we have that $h(\mu) < h(\nu)$ for every μ such that $P(\mu)$ holds. Hence labeled reduction in the calculus restricted to P is strongly normalizing (Thm. 6.51). Consider a maximal derivation ρ^ℓ starting from t_0^ℓ and contracting redexes whose names verify the predicate P; then ρ^ℓ must be finite as we have just argued. Since the LLSC is an orthogonal axiomatic rewriting system (Prop. 6.32), by algebraic confluence (Coro. 2.56) we may close the diagram formed by ρ^ℓ and σ^ℓ with labelled variants of the relative projections ρ/σ and σ/ρ. The situation is:

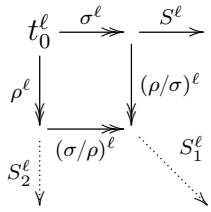
$$\begin{array}{ccc} t_0^\ell & \overset{\sigma^\ell}{\twoheadrightarrow} & \overset{S^\ell}{\rightarrow} \\ {\scriptstyle \rho^\ell}\downarrow & & \downarrow{\scriptstyle (\rho/\sigma)^\ell} \\ & \underset{(\sigma/\rho)^\ell}{\twoheadrightarrow} & \end{array}$$

Note that, by definition of the residual relation, any step contracted along ρ/σ must be the residual of some step in ρ. Moreover, we know that residuals of redexes have the same name as their ancestor (Lem. 6.33), so given any step T^ℓ that is contracted along $(\rho/\sigma)^\ell$ its name ξ is also the name of a step T_0^ℓ that is contracted along ρ^ℓ. Hence ξ must verify the predicate P, which means that $\xi \overset{\mathsf{Name}}{\hookrightarrow} \nu$. In particular $\xi \neq \nu$, since the relation $\overset{\mathsf{Name}}{\hookrightarrow}$ is a strict partial order. Then by Lem. A.95 (in Section A.4.1 of the appendix) there is a residual $S_1 \in S/(\rho/\sigma)$ and the name of its corresponding labelled variant S_1^ℓ is also ν.

We need an auxiliary claim:

Claim: the names of the redexes contracted along $(\sigma/\rho)^\ell$ do not contribute to ν.
Proof of the claim. By contradiction, suppose that $(\sigma/\rho)^\ell$ is of the form $\tau_1^\ell T^\ell \tau_2^\ell$ where the name of T^ℓ is ξ and it contributes to ν, that is $\xi \overset{\mathsf{Name}}{\hookrightarrow} \nu$. Without loss of generality, let T^ℓ be the first such step. Then the names of the redexes contracted along τ_1^ℓ do not contribute to ξ, because if τ_1^ℓ contracts a redex T'^ℓ whose name is $\xi' \overset{\mathsf{Name}}{\hookrightarrow} \xi$, then by transitivity of $\overset{\mathsf{Name}}{\hookrightarrow}$ we have $\xi' \overset{\mathsf{Name}}{\hookrightarrow} \nu$, contradicting the hypothesis that T is the first redex with that property. By Lem. A.96 (in Section A.4.1 of the appendix) this means that T^ℓ must have an ancestor T_0^ℓ, that is a step T_0 such that $T \in T_0/(\sigma/\rho)$ and such that the name of T_0^ℓ is also ξ. Thus we obtain a derivation $\rho^\ell T_0^\ell$ where the name of T_0 verifies P. This contradicts the hypothesis that ρ^ℓ was a maximal derivation contracting only redexes that verify P, which concludes the proof of the claim.

Now since redexes contracted along $(\sigma/\rho)^\ell$ do not contribute to the name of S_1^ℓ, we may again apply Lem. A.96 and obtain that there exists an ancestor S_2^ℓ, *i.e.* a step S_2 such that $S_1 \in S_2/(\sigma/\rho)$ and such that the name of S_2^ℓ is also ν. The situation is as follows:



To conclude the proof, note that $\rho S_2 \overset{\mathsf{Fam}}{\simeq} \sigma S$ since S_2^ℓ and S^ℓ have the same name, namely ν. So $\rho S_2 \in \phi_2$ since $\sigma S \in \phi_2$. By hypothesis, this implies that there exists an hredex $\rho_1 R \in \phi_1$ such that ρ can be written as of the form $\rho_1 R \rho_2$. Consider the labelled variant $\rho_1^\ell R^\ell$ of $\rho_1 R$ whose source is t_0^ℓ. The step R^ℓ is one of the redexes in ρ^ℓ. By construction, the names of all the steps contracted along ρ^ℓ verify the predicate P. In particular, if we let μ stand for the name of R^ℓ, we have that $P(\mu)$ holds, *i.e.* that $\mu \overset{\mathsf{Name}}{\hookrightarrow} \nu$. This, by definition, means that $\rho_1 R \overset{\mathsf{Fam}}{\hookrightarrow} \rho S_2$, and this in turn means that $\phi_1 \overset{\mathsf{Fam}}{\hookrightarrow} \phi_2$, as required.

□

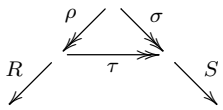
Finally, we are able to prove the main theorem of this section.

Theorem 7.13 (The LSC without gc is a DFS)**.** *The triple* $(\mathcal{A}, \overset{\mathsf{Fam}}{\simeq}, \overset{\mathsf{Fam}}{\hookrightarrow})$ *forms a Deterministic Family Structure, where* $\mathcal{A}$ *is the DRS constructed in Prop. 7.5,* $\overset{\mathsf{Fam}}{\simeq}$ *is the "same family" relation between coinitial hredexes (Def. 7.7) and* $\overset{\mathsf{Fam}}{\hookrightarrow}$ *is the contribution relation between families (Def. 7.10).*

Proof. Let us check each of the axioms:

1. Initial. Let R and S be different coinitial steps. Then we claim that $R \overset{\mathsf{Fam}}{\simeq} S$ does not hold. Indeed, let t^ℓ be an initially labelled variant of the source of R and S, and let R^ℓ and S^ℓ be their respective labelled variants. Then Lem. 6.19 ensures that, since R^ℓ and S^ℓ are different coinitial steps whose source is an initially labelled term, they must have different names. We conclude that $R \in \mathsf{Fam}_\simeq(R)$ but $R \notin \mathsf{Fam}_\simeq(S)$, which entails $\mathsf{Fam}_\simeq(R) \neq \mathsf{Fam}_\simeq(S)$.

2. Copy. Let $\rho R \leqslant \sigma S$, and let us show that $\rho R \overset{\mathsf{Fam}}{\simeq} \sigma S$. By definition of $\leqslant$, there exists a derivation τ such that $S \in R/\tau$ and $\rho\tau \equiv \sigma$.



Let t be the source of the derivations ρ and σ, let t^ℓ be an initially labelled variant of the term t, and let ρ^ℓ, σ^ℓ, τ^ℓ, R^ℓ, S^ℓ, $S^{\ell\ell}$ denote labelled variants of ρ, σ, τ, R, S, and S respectively, in such a way that:

- $\rho^\ell \tau^\ell S^\ell$ is a labelled variant of $\rho\tau S$ whose source is t^ℓ,
- $\rho^\ell R^\ell$ is a labelled variant of ρR whose source is t^ℓ,
- $\sigma^\ell S^{\ell\ell}$ is a labelled variant of σS whose source is t^ℓ.

To see that $\rho R \overset{\mathsf{Fam}}{\simeq} \sigma S$. it suffices to check that R^ℓ and $S^{\ell\ell}$ have the same name. Recall that coinitial labelled variants of permutation equivalent derivations must be cofinal (Prop. 6.35). This implies that $\mathsf{tgt}(\rho^\ell\tau^\ell) = \mathsf{tgt}(\sigma^\ell)$, so $S^\ell = S^{\ell\ell}$. Moreover, residuals of redexes have the same name (Lem. 6.33), and $S \in R/\tau$ so the names of R^ℓ and $S^\ell = S^{\ell\ell}$ coincide, as required.

3. Finite family developments. Let ρ be a potentially infinite derivation that contracts redexes in a finite number of families. Let t^ℓ be an initially labelled variant of the source of ρ, and let ρ^ℓ be a labelled variant of ρ starting from t^ℓ. Let P be the predicate on redex names such that $P(\mu)$ holds if and only if μ is one of the names of the redexes contracted along ρ^ℓ. Then P is bounded, since only a finite number of families are contracted by ρ^ℓ, so by Thm. 6.51 ρ^ℓ must be finite. Hence ρ is also finite.

4. Creation. Let ρR be an hredex such that R creates S, and let us check that $\mathsf{Fam}_{\simeq}(\rho R) \overset{\mathsf{Fam}}{\hookrightarrow} \mathsf{Fam}_{\simeq}(\rho RS)$. By definition, it suffices to check that $\rho R \overset{\mathsf{Fam}}{\hookrightarrow} \rho RS$.

   Consider an initially labelled variant t^ℓ of the source of ρ, and labelled variants ρ^ℓ, R^ℓ, and S^ℓ of ρ, R, and S respectively, such that $\rho^\ell\ R^\ell\ S^\ell$ is a labelled variant of ρRS whose source is t^ℓ. Let μ be the name of R^ℓ and let ν be the name of S^ℓ. By applying Prop. 6.41, we conclude that $\mu \overset{\mathsf{Name}}{\hookrightarrow}_1 \nu$, as required.

5. Contribution. This has been shown in Prop. 7.12.

□

7.4 Optimal Reduction

In previous sections we have endowed the LSC with a notion of Lévy labels (Def. 6.6) and we have used this notion of labeling to define a notion of *redex family* for the LSC without gc (Def. 7.7): two redexes are in the same family if the labeling scheme gives them the same name. We have also shown that this notion of family is well-behaved, in the sense that the LSC without gc forms a Deterministic Family Structure (Thm. 7.13).

In this section: first, we state Lévy's optimality theorem in the setting of the λ-calculus. This is not strictly necessary for our purposes but it hopefully clarifies the rest of the exposition. Second, we state and prove a generalization of Lévy's optimality theorem for an arbitrary Deterministic Family Structure, due to Glauert and Khasidashvili. Finally, using the fact that the LSC without gc forms a Deterministic Family Structure, we obtain an optimality theorem for the LSC without gc, meaning that certain kinds of reductions are optimal. To this purpose, we study the notion of normal forms *up to* gc.

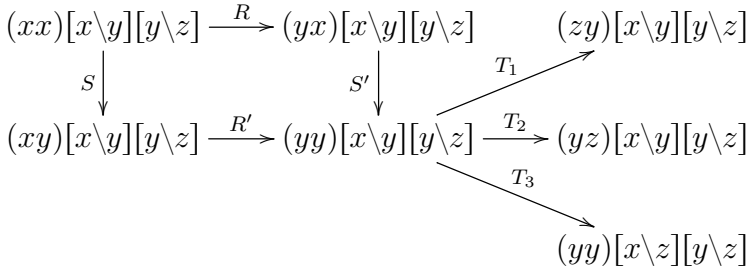
Optimality in the λ-calculus

To state Lévy's optimality result more precisely, we need to introduce a few definitions. Recall the notions of *multistep* and *multiderivation* from Def. 2.43, and also recall from Convention 2.44 that if $\mathcal{M}$ is a multistep we may write just $\mathcal{M}$ to stand for its canonical complete development, which is known to exist and to be unique modulo permutation equivalence. The definitions and results in this subsection can be traced back to Lévy's work and are nicely exposed in Asperti and Guerrini's book [14].

Definition 7.14 (Family reduction)**.** Let $(\mathcal{A}, \simeq, \hookrightarrow)$ be a DFS. A *family reduction* is a multiderivation $\mathcal{M}_1 \dots \mathcal{M}_n$ in $\mathcal{A}$ such that for each $i \in \{1, \dots, n\}$ all the steps in $\mathcal{M}_i$ belong to the same family. More precisely, for all $i \in \{1, \dots, n\}$ and for any two steps $R, S \in \mathcal{M}_i$ we have that $\mathcal{M}_1 \dots \mathcal{M}_{i-1} R \simeq \mathcal{M}_1 \dots \mathcal{M}_{i-1} S$. Moreover, a family reduction is *complete* if each $\mathcal{M}_i$ is a maximal set of steps that have $\mathsf{src}(\mathcal{M}_i)$ as their source and belong to the same family.

The motivation behind Lévy's definition of complete family reduction is that an optimal implementation should never duplicate work. Rather it should *share* the computational work of contracting all the copies of a redex. Performing one computational step in an optimal implementation should correspond to contracting all and only the redexes in some family.

Example 7.15 (Family reductions)**.** *Consider the following diagram in the LSC without* gc*:*



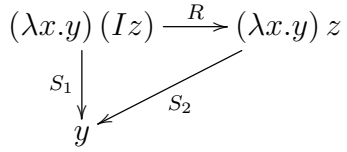
The multiderivation $\{R, S\}$ (consisting of a sequence of exactly one multistep) is not a family reduction, because R and S are not in the same family, while $\{R\}\{S'\}$ and $\{S\}\{R'\}$ are both complete family reductions. The multiderivation $\{R\}\{S'\}\{T_1, T_2\}$ is a family reduction, but it is not complete because the set $\{T_1, T_2\}$ is not a maximal set of coinitial steps in the same family. The multiderivation $\{R\}\{S'\}\{T_1, T_2, T_3\}$ is a complete family reduction.

Starting from a term t, we are interested in finding the *optimal*, *i.e.* the *shortest* family reduction.

Definition 7.16 (Optimal reduction)**.** Let $x \in \mathcal{A}$ be an object in a DFS. A family reduction starting on x and reaching the normal form of x is *optimal* if its length is minimum among all the family reductions reaching the normal form of x.

By requiring that a multiderivation is a complete family reduction, one guarantees that no computational work is ever duplicated. Still, a complete family reduction may not be optimal, because it may perform unnecessary computational work. For example, in the λ-calculus,

given the diagram:



the multiderivation $\{R\}\{S_2\}$ is a complete family reduction that reaches the normal form. However, it is not optimal, since $\{S_1\}$ is a shorter complete family reduction reaching the normal form.

To formally define what it means for a multiderivation to perform only necessary computational work, Lévy defines a step $R : t \to s$ to be *needed* if every coinitial derivation $\sigma : t \twoheadrightarrow u$ that reaches the normal form of t contracts at least one residual of R. A family reduction $\mathcal{M}_1 \dots \mathcal{M}_n$ is *needed* if every multistep $\mathcal{M}_i$ contains at least one needed step. Lévy's optimality result asserts that:

Theorem 7.17 (Optimality — Lévy, 1978)**.** *In the λ-calculus, any needed, complete family reduction reaching a normal form is optimal.*

Proof. A particular case of Thm. 7.24 in the next subsection. □

Optimality in Deterministic Family Structures

In [61], Glauert and Khasidashvili propose a generalization of Lévy's optimality result. This result generalizes Thm. 7.17 along two dimensions. First, the result does not only apply to the λ-calculus, but in general to any Deterministic Family Structure, of which the λ-calculus is a particular case. Second, the result does not only apply to reductions reaching a normal form, but in general to reductions reaching an *answer*, where the notion of answer is an additional parameter of the generalized optimality theorem. The notion of answer is specified by a set of terms which may vary in different settings. For example in the λ-calculus one may consider any of the following sets as the set of answers:

$$\begin{aligned}
\{t \in \mathcal{T} \mid \nexists s \in \mathcal{T}.\, t \to s\} &\quad \text{(normal forms)}\\
\{\lambda x.t \mid x \in \mathcal{V},\ t \in \mathcal{T}\} &\quad \text{(abstractions)}\\
\{\lambda x_1 \dots x_n.y\, t_1 \dots t_m \mid n, m \geqslant 0,\ x_1, \dots, x_n, y \in \mathcal{V},\ t_1, \dots, t_m \in \mathcal{T}\} &\quad \text{(head normal forms)}\\
\{\lambda x.t \mid x \in \mathcal{V}, t \in \mathcal{T}\} \cup \{x t_1 \dots t_n \mid n \geqslant 0,\ x \in \mathcal{V},\ t_1, \dots, t_n \in \mathcal{T}\} &\quad \text{(weak head normal forms)}
\end{aligned}$$

where $\mathcal{V}$ is the set of variables and $\mathcal{T}$ the set of all terms. The set of answers is denoted by $\mathcal{X}$. The definitions and results in this subsection can be traced back to Lévy's work and correspond to Glauert and Khasidashvili's generalization to arbitrary DFSs [61, 14].

Definition 7.18 ($\mathcal{X}$-needed)**.** Let $\mathcal{A}$ be an orthogonal axiomatic rewriting system and let $\mathcal{X}$ be a set of objects. A step $R : x \to y$ is *$\mathcal{X}$-needed* if every derivation $\sigma : x \twoheadrightarrow z \in \mathcal{X}$ contracts at least one residual of R. A multistep $\mathcal{M}$ is $\mathcal{X}$-needed if it contains at least one $\mathcal{X}$-needed step. A multiderivation $\mathcal{M}_1 \dots \mathcal{M}_n$ is $\mathcal{X}$-needed if the multistep $\mathcal{M}_i$ is $\mathcal{X}$-needed for all $i \in 1..n$.

For technical reasons, the set of answers may not be an arbitrary set. It must be a stable set:

Definition 7.19 (Stable set)**.** A set $\mathcal{X}$ of objects is *stable* if:

1. $\mathcal{X}$ is closed under parallel moves, *i.e.* for any $x \notin \mathcal{X}$, any $\rho : x \twoheadrightarrow y \in \mathcal{X}$, and any reduction $\sigma : x \twoheadrightarrow z$ which does not contain objects in $\mathcal{X}$, the target of ρ/σ is in $\mathcal{X}$.
2. $\mathcal{X}$ is closed under unneeded expansion, *i.e.* for any $R : x \to y$ such that $x \notin \mathcal{X}$ and $y \in \mathcal{X}$, the step R is $\mathcal{X}$-needed.

Example 7.20 (Abstractions are stable in the λ-calculus)**.** *In the λ-calculus, the set of abstractions $\{\lambda x.t \mid t \in \mathcal{T}\}$ is stable. It is easy to see that $\mathtt{NF}_\beta$ is closed under parallel moves, because if $\rho : t \twoheadrightarrow \lambda x.s$ and $\sigma : t \twoheadrightarrow u$ then $\sigma/\rho : \lambda x.s \twoheadrightarrow \lambda x.r$. To see that $\mathtt{NF}_\beta$ is closed under unneeded expansion, consider a step $R : t \to \lambda x.s$ such that t is not an abstraction. Then t must be of the form $(\lambda x.t_1)\, t_2$. Any derivation $\sigma : (\lambda x.t_1)\, t_2 \twoheadrightarrow \lambda y.u$ must contract the residual of R, otherwise all steps are internal to t_1 and t_2, and the target is still an application.*

Definition 7.21 ($\mathcal{X}$-optimal reduction)**.** Let $x \in \mathcal{A}$ be an object in a DFS and let $\mathcal{X}$ be a stable set on $\mathcal{A}$. A family reduction $D : x \twoheadrightarrow y \in \mathcal{X}$ is *$\mathcal{X}$-optimal* if its length is minimum among all the family reductions of the form $x \twoheadrightarrow y \in \mathcal{X}$ (where x is fixed and y varies).

Let $\mathsf{FAM}(D)$ denote the set of families of a multiderivation. More precisely:

$$\mathsf{FAM}(\mathcal{M}_1 \dots \mathcal{M}_n) \stackrel{\text{def}}{=} \{\mathsf{Fam}_\simeq(\mathcal{M}_1 \dots \mathcal{M}_{i-1} R) \mid 1 \leqslant i \leqslant n,\ R \in \mathcal{M}_i\}$$

Then we can prove the following auxiliary result.

Lemma 7.22. *Let $\mathcal{X}$ be a stable set of terms in a DFS. If $D : x \twoheadrightarrow y \in \mathcal{X}$ is a family reduction, then $\#\mathsf{FAM}(D) \leqslant |D|$.*

Proof. Let $D = \mathcal{M}_1 \dots \mathcal{M}_n$. By definition, each family $\phi \in \mathsf{FAM}(D)$ can be written as $\phi = \mathsf{Fam}_\simeq(\mathcal{M}_1 \dots \mathcal{M}_{i-1} R)$ for some $i \in \{1, \dots, n\}$ and some $R \in \mathcal{M}_i$. Consider the map $\mathtt{I}$ giving, for each family, the minimum such index:

$$\begin{array}{rcl} \mathtt{I} : \mathsf{FAM}(D) & \to & \{1, \dots, n\} \\ \phi & \mapsto & \min\{i \in \{1, \dots, n\} \mid \exists R \in \mathcal{M}_i.\ \phi = \mathsf{Fam}_\simeq(\mathcal{M}_1 \dots \mathcal{M}_{i-1} R)\} \end{array}$$

To show that $\#\mathsf{FAM}(D) \leqslant |D|$, it suffices to show that $\mathtt{I}$ is injective. Indeed, if $\mathtt{I}(\phi) = \mathtt{I}(\phi') = i$, then there are two steps $R, S \in \mathcal{M}_i$ such that $\phi = \mathsf{Fam}_\simeq(\mathcal{M}_1 \dots \mathcal{M}_{i-1} R)$ and $\phi' = \mathsf{Fam}_\simeq(\mathcal{M}_1 \dots \mathcal{M}_{i-1} S)$. But D is a family reduction, so $\mathcal{M}_1 \dots \mathcal{M}_{i-1} R \simeq \mathcal{M}_1 \dots \mathcal{M}_{i-1} S$. Therefore $\phi = \phi'$. □

Lemma 7.23. *Let $\mathcal{X}$ be a stable set of terms in a DFS. If $D : x \twoheadrightarrow y \in \mathcal{X}$ is a $\mathcal{X}$-needed complete family reduction, then $|D| = \#\mathsf{FAM}(D)$.*

Proof. In Lem. 7.22 we have seen that $|D| \geqslant \#\mathsf{FAM}(D)$ for any family reduction, so we are left to show that $|D| \leqslant \#\mathsf{FAM}(D)$. Let $D = \mathcal{M}_1 \dots \mathcal{M}_n$. Since D is $\mathcal{X}$-needed, for each $i \in \{1, \dots, n\}$ the set $\mathcal{M}_i$ contains an $\mathcal{X}$-needed step R_i. It suffices to show that the following map Φ is injective.

$$\begin{array}{rcl} \Phi : \{1, \dots, n\} & \to & \mathsf{FAM}(D) \\ i & \mapsto & \mathsf{Fam}_\simeq(\mathcal{M}_1 \dots \mathcal{M}_{i-1} R_i) \end{array}$$

Indeed, suppose that $\Phi(i) = \Phi(j)$ for some $1 \leqslant i, j \leqslant n$ with $i \neq j$. Without loss of generality, let $i < j$, and suppose moreover that the pair (i, j) is chosen so that j is the least possible index, *i.e.* there is no other pair (i', j') such that $\Phi(i') = \Phi(j')$ and $i' < j' < j$. We argue that this case is impossible. Let us write D_i for the multiderivation $\mathcal{M}_1 \dots \mathcal{M}_i$, for each $0 \leqslant i \leqslant n$. Since $\Phi(i) = \Phi(j)$ we have that R_i and R_j are in the same family, more precisely, $D_{i-1}R_i \simeq D_{j-1}R_j$. We consider two cases, depending on whether step R_j has an ancestor before the derivation $\mathcal{M}_i \dots \mathcal{M}_{j-1}$:

1. **If R_j has an ancestor.** That is, there is a step R'_j such that $R'_j \langle \mathcal{M}_i \dots \mathcal{M}_{j-1} \rangle R_j$. By the Copy axiom, R'_j and R_j must be in the same family, more precisely, $D_{i-1}R'_j \simeq D_{j-1}R_j$. Then by transitivity of the family relation, $D_{i-1}R'_j \simeq D_{i-1}R_i$. Since D is a complete family reduction, $\mathcal{M}_i$ is a maximal set of steps in the same family, so we obtain that $R'_j \in \mathcal{M}_i$. But then by Autoerasure $R'_j/\mathcal{M}_i \dots \mathcal{M}_{j-1}$ must be empty. This contradicts the fact that $R_j \in R'_j/\mathcal{M}_i \dots \mathcal{M}_{j-1}$.

2. **If R_j has no ancestor.** That is, there is no step R'_j such that $R'_j \langle \mathcal{M}_i \dots \mathcal{M}_{j-1} \rangle R_j$. In particular, the range $\{i, \dots, j-1\}$ cannot be empty. Let $q \in \{i, \dots, j-1\}$ be such that there is an ancestor $R'_j \langle \mathcal{M}_{q+1} \dots \mathcal{M}_{j-1} \rangle R_j$ but there is no ancestor $R''_j \langle \mathcal{M}_q \rangle R'_j$. Moreover by Creation there must be a step in $\mathcal{M}_q$ that contributes to R'_j, and since all the steps in $\mathcal{M}_q$ are in the same family, this means that $\mathsf{Fam}_{\simeq}(D_{q-1}R_q) \hookrightarrow \mathsf{Fam}_{\simeq}(D_qR'_j)$. The situation is:

$$\begin{array}{ccccccccc} \xrightarrow{\mathcal{M}_1 \dots \mathcal{M}_{i-1}} & \bullet & \xrightarrow{\mathcal{M}_i \dots \mathcal{M}_{q-1}} & \bullet & \xrightarrow{\mathcal{M}_q} & \bullet & \xrightarrow{\mathcal{M}_{q+1} \dots \mathcal{M}_{j-1}} & \bullet & \xrightarrow{\mathcal{M}_j \dots \mathcal{M}_n} \\ & \downarrow{\scriptstyle R_i} & & \downarrow{\scriptstyle R_q} & & \downarrow{\scriptstyle R'_j} & & \downarrow{\scriptstyle R_j} & \end{array}$$

   Note that $\mathsf{Fam}_{\simeq}(D_{q-1}R_q) \hookrightarrow \mathsf{Fam}_{\simeq}(D_qR'_j) = \mathsf{Fam}_{\simeq}(D_{i-1}R_i)$, so by the completeness part of the Contribution axiom, there must exist a step in the history of R_i in the same family as R_q contributing to R_i. That is, there is an index $p \in \{1, \dots, i-1\}$ such that $\mathsf{Fam}_{\simeq}(D_{q-1}R_q) = \mathsf{Fam}_{\simeq}(D_{p-1}R_p) \hookrightarrow \mathsf{Fam}_{\simeq}(D_{i-1}R_i)$. To conclude, observe that (p, q) is a pair of indices such that $\Phi(p) = \Phi(q)$ and $p < q < j$. This contradicts the request that j is the least possible index with such condition.

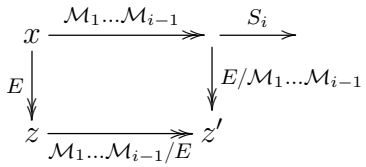
□

The following generalization of Lévy's optimality theorem (Thm. 7.17) is due to Glauert and Khasidashvili ([61, Theorem 5.2]).

Theorem 7.24 (Generalized optimality — Glauert and Khasidashvili, 1996)**.** *Let $\mathcal{X}$ be a stable set of terms in a DFS. Then any $\mathcal{X}$-needed complete family reduction $D : x \twoheadrightarrow y \in \mathcal{X}$ is $\mathcal{X}$-optimal.*

Proof. Let $D = \mathcal{M}_1 \dots \mathcal{M}_m : x \twoheadrightarrow y \in \mathcal{X}$ be an $\mathcal{X}$-needed complete family reduction and let $E = \mathcal{N}_1 \dots \mathcal{N}_n : x \twoheadrightarrow z \in \mathcal{X}$ be any family reduction to $\mathcal{X}$. First we argue that $\mathsf{FAM}(D) \subseteq \mathsf{FAM}(E)$. Let $\phi \in \mathsf{FAM}(D)$ be a family. By definition, $\phi = \mathsf{Fam}_{\simeq}(\mathcal{M}_1 \dots \mathcal{M}_{i-1}R_i)$ for some $i \in \{1, \dots, m\}$ and some $R_i \in \mathcal{M}_i$. Moreover, since D is $\mathcal{X}$-needed, for each $1 \leqslant i \leqslant m$, the

set $\mathcal{M}_i$ contains an $\mathcal{X}$-needed redex S_i. Consider the derivation $E/\mathcal{M}_1 \dots \mathcal{M}_{i-1}$. Note that its target is an object z' which coincides with the target of $\mathcal{M}_1 \dots \mathcal{M}_{i-1}/E : z \twoheadrightarrow z'$. Since $z \in \mathcal{X}$ and $\mathcal{X}$ is a stable set, hence closed under parallel moves, we have that $z' \in \mathcal{X}$ as well. So for each $i \in \{1, \dots, m\}$ the situation is:

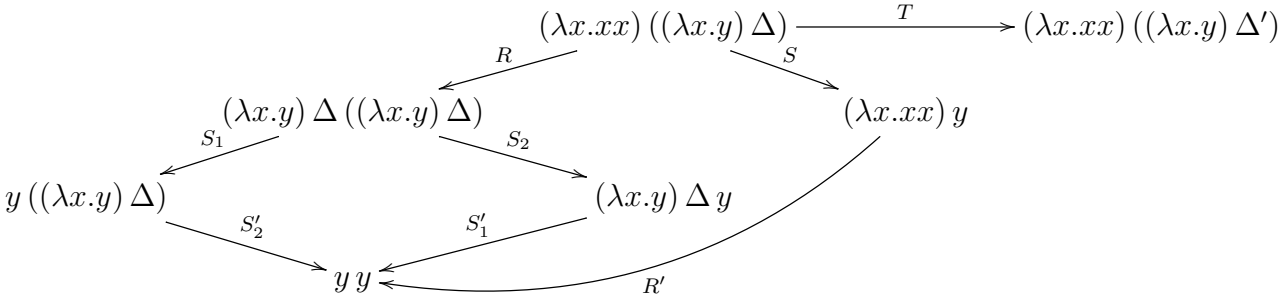


Moreover, S_i is $\mathcal{X}$-needed, so a residual of S_i is contracted somewhere along $E/\mathcal{M}_1 \dots \mathcal{M}_{i-1}$. By the Copy axiom, this means that E contracts a redex in the same family as S_i, that is, the multiderivation E, seen as a derivation, can be written, for each $i \in \{1, \dots, m\}$, as of the form $E = \rho_i T_i \sigma_i$, where $\mathcal{M}_1 \dots \mathcal{M}_{i-1} R_i \simeq \mathcal{M}_1 \dots \mathcal{M}_{i-1} S_i \simeq \rho_i T_i$. So we have that $\phi = \mathsf{Fam}_{\simeq}(\rho_i T_i) \in \mathsf{FAM}(E)$. This proves our claim that $\mathsf{FAM}(D) \subseteq \mathsf{FAM}(E)$. To conclude the proof of this theorem, observe that:

$$\begin{array}{rcll} |D| & = & \#\mathsf{FAM}(D) & \text{by Lem. 7.23 since } D \text{ is an } \mathcal{X}\text{-needed complete family reduction} \\ & \leqslant & \#\mathsf{FAM}(E) & \text{since } \mathsf{FAM}(D) \subseteq \mathsf{FAM}(E) \text{ as we have just claimed} \\ & \leqslant & |E| & \text{by Lem. 7.22 since } E \text{ is a family reduction} \end{array}$$

□

Example 7.25 (Optimal reduction in the λ-calculus)**.** *Let Δ be any term such that $\Delta \to \Delta'$ and consider the following diagram:*



Then the family reductions $\{R\}\{S_1, S_2\}$ and $\{S\}\{R'\}$ are both optimal reductions to normal form. The family reductions $\{R\}\{S_1\}\{S'_2\}$ and $\{R\}\{S_2\}\{S'_1\}$ are not complete. Any family reduction starting with $\{T\} \dots$ is not needed, because the step T is not needed to obtain a normal form.

Optimality in the LSC without gc

Combining the fact that the LSC without gc is a Deterministic Family Structure (Thm. 7.13) with the generalized optimality theorem for DFSs (Thm. 7.24), one obtains an optimality result for the LSC. However, the generalized optimality theorem depends on the choice of a stable set $\mathcal{X}$ that captures the notion of answer that one is interested in.

One may be interested in the set of answers given by the normal forms of the LSC without gc, that is, in the set:

$$\mathtt{NF}_{\mathtt{db,ls}} \stackrel{\text{def}}{=} \{t \in \mathcal{T} \mid \nexists s \in \mathcal{T}.\, t \to_{\mathtt{db,ls}} s\}$$

It is easy to show that $\mathtt{NF}_{\mathtt{db},\mathtt{ls}}$ is a stable set. But the notion of $\mathtt{NF}_{\mathtt{db},\mathtt{ls}}$-optimality that one obtains in that case is not very interesting, for two reasons. One reason is that in the LSC without gc there is no erasure, which means that every step is always $\mathtt{NF}_{\mathtt{db},\mathtt{ls}}$-needed. Another reason is that in the LSC without gc one is not really interested in obtaining the normal form of a term. For example let $\Omega = (\lambda x.xx)\,\lambda x.xx$ and consider the following derivation:

$$\begin{array}{lll}(\lambda x.\lambda y.x)\,z\,\Omega & \to_{\mathtt{db}} & (\lambda y.x)[x\backslash z]\,\Omega \\ & \to_{\mathtt{db}} & x[y\backslash\Omega][x\backslash z] \\ & \to_{\mathtt{ls}} & z[y\backslash\Omega][x\backslash z] \\ & \to_{\mathtt{db}} & z[y\backslash(xx)[x\backslash\lambda x.xx]][x\backslash z] \\ & \to & \dots\end{array}$$

In this example, reduction goes on forever without reaching a normal form, evaluating the term inside the substitution $[y\backslash\ldots]$, even though this substitution is never used. One is actually interested in the set of normal forms *up to garbage collection* of unused substitutions. This is the notion of reachable normal form defined below.

Definition 7.26 (Reachable normal forms)**.** Let $\mathsf{nf}_{\mathtt{gc}}(t)$ denote the gc-normal form of a given term t. The set RNF of *reachable normal forms* is the set of terms:

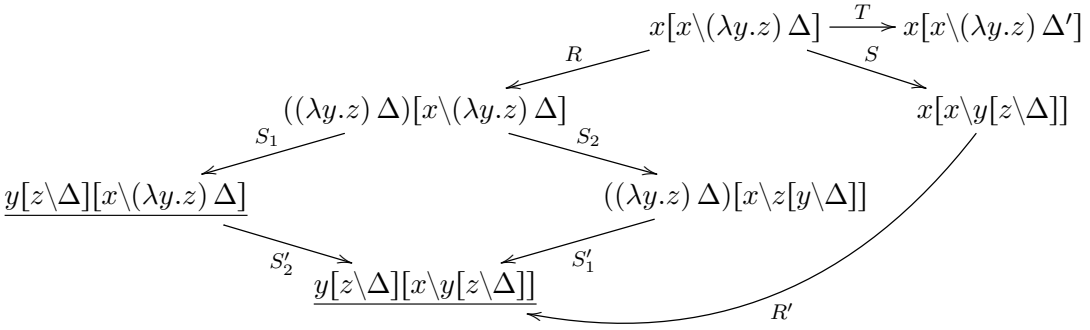
$$\mathsf{RNF} \stackrel{\text{def}}{=} \{t \in \mathcal{T} \mid \mathsf{nf}_{\mathtt{gc}}(t) \in \mathtt{NF}_{\mathtt{db},\mathtt{ls}}\}$$

The following proposition justifies that Thm. 7.24 may be applied to the notion of RNF-optimal reductions.

Proposition 7.27 (The set RNF is stable — ♣ Prop. A.112)**.**

Proof. The proof is technical and can be found in Section A.4.2 of the appendix. It requires to introduce the notion of *reachable step*, which is, intuitively, a step not erased by any sequence of gc steps. The proof also relies on the notion of *nesting* introduced by Accattoli et al. in [6]. □

Example 7.28 (Optimal RNF-reduction in the LSC without gc)**.** *Let Δ be any term such that $\Delta \to \Delta'$ and consider the following diagram, in which the terms in* RNF *have been underlined:*



Then the family reductions $\{R\}\{S_1, S_2\}$ and $\{S\}\{R'\}$ are RNF*-optimal by Thm. 7.24. Any family reduction starting with $\{T\}\ldots$ is not* RNF*-needed, because T is not needed to reach a term in* RNF*.*

Note that the family reduction $\{R\}\{S_1\}$ reaches a term in RNF *in the least possible number of multisteps, but it is not complete because $\{S_1\}$ is not maximal, so Thm. 7.24 does not ensure that it is* RNF*-optimal.*

7.5 Standardization

In a very general sense, the problem of *standardization* consists in finding, for each derivation $\rho : x \twoheadrightarrow y$ an *equivalent* derivation $\rho' : x' \twoheadrightarrow y'$ that is *standard*. There are two keywords involved here, *equivalent* and *standard*, worthy of a short discussion.

In principle, one may be interested in various different notions of *equivalence* between derivations. For example, in their original standardization result [46], one could say that Curry and Feys were interested in the equivalence relation $\sim$ that equates two derivations whenever they are coinitial and cofinal. That is, given $\rho : x \twoheadrightarrow y$ and $\rho' : x' \to y'$ one has $\rho \sim \rho'$ if and only if $(x, y) = (x', y')$.

Later, Lévy noted that the notion of equivalence that Curry and Feys were really after was the relation of permutation equivalence. Recall from Lem. 2.41 that if any two derivations are permutation equivalent then they are coinitial and cofinal, so permutation equivalence is a finer equivalence relation than $\sim$. In fact, Lévy remarked that in the λ-calculus there exist derivations that are coinitial and cofinal but which are not permutation equivalent, such as in the "syntactic accident" $I\,(I\,x) \to I\,x$.

One may also be interested in other notions of equivalence between derivations. For example, Laneve [102] studies *distributive permutation equivalence* which allows swapping adjacent steps as long as this does not cause duplication nor erasure.

The word *standardization* is most commonly used in the literature to refer to standardization with respect to the equivalence relation of permutation equivalence.

Given a fixed notion of equivalence $\sim$ between derivations, one may sometimes prove a standardization result, involving a class of derivations S, whose elements are called *standard derivations*. A standardization result states that one may find, for each derivation ρ, an equivalent standard derivation $\rho' \in S$. A stronger standardization result would moreover ensure that for each derivation ρ there is a *unique* equivalent derivation $\rho' \in S$, that is, that the set of equivalence classes modulo $\sim$ is in 1–1 correspondence with the set S. Moreover, the standardization result is usually proved constructively, by giving a procedure that yields, for every derivation ρ the standard representative ρ' of its $\sim$-equivalence class.

In this section we define a standardization procedure for Deterministic Family Structures, by requesting some additional axioms. The proof that the standardization procedure terminates relies on the Finite Family Developments property. As a corollary, we obtain a standardization theorem for the LSC without gc.

Many abstract standardization results have been studied before. The result we present here is an adaptation of Klop's parallel standardization theorem ([135, Proposition 8.5.19]) to the framework of Deterministic Family Structures.

Note that in [5, Theorem 3, Theorem 4], Accattoli, Bonelli, Lombardi, and Kesner have already proposed a standardization procedure for the LSC. Our procedure differs from theirs in in the following aspects:

1. Our procedure relies on the Finite Family Developments theorem, while [5] relies on the fact that the LSC enjoys a number of axioms proposed by Melliès in [118, Chapter 4].
2. Our standardization procedure is inspired by Klop's [135, Section 8.5.2], and it is based on *selection*, resembling selection sort, while [5] is based on *permutation* of anti–standard pairs, resembling bubble sort.
3. Our procedure does not deal with the gc rule, while [5] does.
4. Our procedure imposes a fixed order for redexes in such a way that the standard reduction is syntactically unique, while [5] considers standard forms modulo permutation of disjoint redexes, in such a way that the standard reduction is unique *up to square equivalence*.

Standardization in Deterministic Family Structures

In this subsection we prove a standardization result for Deterministic Family Structures that verify some additional constraints. The main result of this subsection is the standardization result for DFSs (Prop. 7.39). We begin by proving a simple technical result.

Proposition 7.29 (Projection does not create families)**.** *Let $\mathcal{A}$ be a DFS, let $\phi : t \twoheadrightarrow t'$ be a derivation in $\mathcal{A}$, and let ρ and σ be coinitial derivations in $\mathcal{A}$ starting from t'. Then the set of families of redexes contracted along ρ/σ is contained in the set of families of redexes contracted along ρ, relatively to the history ϕ. More precisely, if ρ/σ can be written as $\tau_1 T \tau_2$ then ρ can be written as $\upsilon_1 U \upsilon_2$ such that $\mathsf{Fam}_{\simeq}(\phi\upsilon_1 U) = \mathsf{Fam}_{\simeq}(\phi\sigma\tau_1 T)$.*

Proof. By induction on the length of ρ. The base case is trivial. If $\rho = R\rho'$ we have that $\rho/\sigma = (R/\sigma)(\rho'/(\sigma/R))$ by definition. Let ρ/σ be written as $\tau_1 T \tau_2$. We consider two subcases, depending on whether τ_1 is a proper prefix of R/σ or not:

1. *If τ_1 is a proper prefix of R/σ.* Then $R/\sigma = \tau_1 T \tau_2'$ and $\tau_2 = \tau_2'(\rho'/(\sigma/R))$. Note that $T \in (R/\sigma)/\tau_1$ so $R \langle \sigma\tau_1 \rangle T$. Then by taking $\upsilon_1 := \epsilon$, $U := R$ and $\upsilon_2 := \rho'$ we have that $\mathsf{Fam}_{\simeq}(\phi R) = \mathsf{Fam}_{\simeq}(\phi\sigma\tau_1 T)$ since T is a copy of R, and as a consequence of the Copy axiom.
2. *If τ_1 is not a proper prefix of R/σ.* Then $\rho'/(\sigma/R) = \tau_1' T \tau_2$ and $\tau_1 = (R/\sigma)\tau_1'$. By *i.h.* on the derivation ρ' (using ϕR as the new history), we conclude that ρ' can be written as $\upsilon_1' U \upsilon_2$ in such a way that:

$$\begin{array}{rcll} \mathsf{Fam}_{\simeq}(\phi R \upsilon_1' U) & = & \mathsf{Fam}_{\simeq}(\phi R(\sigma/R)\tau_1' T) & \\ & = & \mathsf{Fam}_{\simeq}(\phi\sigma(R/\sigma)\tau_1' T) & \text{by the Copy axiom,} \\ & & & \text{since } \phi R(\sigma/R)\tau_1' T \leqslant \phi\sigma(R/\sigma)\tau_1' T \\ & & & \text{since } \phi R(\sigma/R)\tau_1' \equiv \phi\sigma(R/\sigma)\tau_1' \end{array}$$

   Hence by taking $\upsilon_1 := R\upsilon_1'$ we conclude.

□

To prove the standardization result, let us state a few further auxiliary definitions, including the crucial notion of *uniform multi-selection strategy*. Recall that in an orthogonal axiomatic rewriting system, the letters $\mathcal{M}, \mathcal{N}, \ldots$ range over multisteps, $D, E, \ldots$ range over multiderivations, and Multistep stands for the set of multisteps.

Definition 7.30 (Belonging)**.** In an orthogonal axiomatic rewriting system $\mathcal{A}$, a step R *belongs* to a derivation ρ, written $R \lhd \rho$, if and only if ρ can be written as of the form $\rho = \rho_1 R' \rho_2$ where $R' \in R/\rho_1$. A multistep $\mathcal{M}$ *belongs* to a derivation ρ, written $\mathcal{M} \lhd \rho$, if and only if $R \lhd \rho$ for all $R \in \mathcal{M}$.

Definition 7.31 (Multi-selection strategy)**.** In an orthogonal axiomatic rewriting system $\mathcal{A}$, a *multi-selection strategy* is a function $\mathbb{M}$ that maps every non-empty derivation ρ to a coinitial multistep $\mathcal{M} \in \mathsf{Multistep}$ such that $\mathcal{M} \lhd \rho$ and $\mathcal{M}/\rho = \varnothing$.

Definition 7.32 (Uniform multi-selection strategy)**.** A multi-selection strategy $\mathbb{M}$ is *uniform* if $\rho \equiv \sigma$ implies $\mathbb{M}(\rho) = \mathbb{M}(\sigma)$ for any non-empty ρ, σ.

Example 7.33. *In the λ-calculus, consider the trivial multi-selection strategy $\mathbb{M}_{\text{Triv}}$ that always selects the first step of a given derivation. More precisely, let $\mathbb{M}_{\text{Triv}}(R\rho) \stackrel{\text{def}}{=} \{R\}$. Then $\mathbb{M}_{\text{Triv}}$ is a multi-selection strategy because for every non-empty derivation $R\rho$ we have that $R \lhd R\rho$ and that $R/R\rho = \varnothing$.*

However, $\mathbb{M}_{\text{Triv}}$ is not uniform. For example, if $RS' \equiv SR'$, such as in the following diagram, we have that $\mathbb{M}_{\text{Triv}}(RS') = \{R\} \neq \{S\} = \mathbb{M}_{\text{Triv}}(SR')$.

$$\begin{array}{ccc} (\lambda x.(\lambda y.z)\,x\,x)\,t & \xrightarrow{R} & (\lambda y.z)\,t\,t \\ {\scriptstyle S}\downarrow & & {\scriptstyle S'}\downarrow \\ (\lambda x.z\,x)\,t & \xrightarrow{R'} & z\,t \end{array}$$

In the remainder of this subsection, we show that any uniform multi-selection strategy $\mathbb{M}$ induces, for a given derivation ρ, a permutation equivalent derivation ρ^*. This gives us a standardization result, parametric on $\mathbb{M}$. The set of standard derivations is the set $\{\rho^* \mid \rho \text{ is a derivation}\}$. Moreover, we show that the induced derivation ρ^* is unique, up to permutation equivalence.

Definition 7.34 (Induced multiderivation)**.** In an orthogonal axiomatic rewriting system, let $\mathbb{M}$ be a multi-selection strategy and let ρ be any derivation. The *sequence induced by $\mathbb{M}$ on ρ*, written $\mathbb{M}^\star(\rho)$, is a possibly infinite sequence of multisteps, defined by the following recursive equations:

$$\mathbb{M}^\star(\rho) \stackrel{\text{def}}{=} \begin{cases} \epsilon & \text{if } \rho = \epsilon \\ \mathbb{M}(\rho) \cdot \mathbb{M}^\star(\rho/\mathbb{M}(\rho)) & \text{otherwise} \end{cases}$$

If recursion terminates, the sequence is finite and we call it the *multiderivation induced by $\mathbb{M}$ on ρ*.

In an arbitrary rewriting system, this recursive definition may not terminate. The following lemma provides sufficient conditions for $\mathbb{M}^\star(\rho)$ to be well-defined. Namely, in a Deterministic Family Structure, the recursive definition of $\mathbb{M}^\star(\rho)$ is well-founded, as a consequence of Finite Family Developments.

Lemma 7.35. *Let $\mathbb{M}$ is a multi-selection strategy in a DFS. If ρ is any (finite) derivation, then $\mathbb{M}^\star(\rho)$ is finite.*

Proof. Let ρ be a finite derivation, let $D = \mathbb{M}^\star(\rho)$ be the multiderivation induced by $\mathbb{M}$ on ρ, and let $\mathcal{F}$ be the set of redex families that are contracted along ρ, more precisely:

$$\mathcal{F} \stackrel{\text{def}}{=} \{\mathsf{Fam}_\simeq(\rho_1 R) \mid \exists \rho_2.\ \rho = \rho_1 R \rho_2\}$$

Claim. Write D as a possibly infinite sequence of multisteps $D = \mathcal{M}_1 \dots \mathcal{M}_n \dots$. Suppose that $\sigma = \sigma_1 \dots \sigma_n$ is any complete development of a prefix $\mathcal{M}_1 \dots \mathcal{M}_n$ of D, where each σ_i is a complete development of $\mathcal{M}_i$. Then the set of families of the redexes contracted along σ is contained in $\mathcal{F}$.
Proof of the claim. Let $\sigma = \sigma_1 \dots \sigma_n$ and let $\sigma_i = S_1^i \dots S_{m_i}^i$ for each $1 \leqslant i \leqslant n$. An arbitrary step of σ is one of the steps S_j^i with $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m_i$. It suffices to show that the family of each S_j^i is in $\mathcal{F}$. More precisely, we aim to show that $\mathsf{Fam}_\simeq(\sigma_1 \dots \sigma_{i-1} S_1^i \dots S_{j-1}^i S_j^i) \in \mathcal{F}$ holds for every i, j.

Let $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m_i$ be arbitrary indices. Note that S_j^i is a redex in σ_i and σ_i is a complete development of $\mathcal{M}_i$, so S_j^i has an ancestor $S^\star \langle S_1^i \dots S_{j-1}^i \rangle S_j^i$ with $S^\star \in \mathcal{M}_i$. This means that S_j^i is a copy of $S^\star$, hence they are in the same family, *i.e.* $\mathsf{Fam}_\simeq(\sigma_1 \dots \sigma_{i-1} S_1^i \dots S_j^i) = \mathsf{Fam}_\simeq(\sigma_1 \dots \sigma_{i-1} S^\star)$. Moreover, by construction, $\mathcal{M}_i = \mathbb{M}(\rho / \mathcal{M}_1 \dots \mathcal{M}_{i-1})$. Since $\mathbb{M}$ is a multi-selection strategy, we have that $S^\star \lhd \rho / \mathcal{M}_1 \dots \mathcal{M}_{i-1}$. This means that $\rho / \mathcal{M}_1 \dots \mathcal{M}_{i-1}$ can be written as $\rho_1 S^{\star\star} \rho_2$ where $S^\star \langle \rho_1 \rangle S^{\star\star}$. This means that $S^{\star\star}$ is a copy of $S^\star$, hence they are in the same family: $\mathsf{Fam}_\simeq(\sigma_1 \dots \sigma_{i-1} S^\star) = \mathsf{Fam}_\simeq(\sigma_1 \dots \sigma_{i-1} \rho_1 S^{\star\star})$. Moreover, since projection does not create families in a DFS (Prop. 7.29) and $\rho / \mathcal{M}_1 \dots \mathcal{M}_{i-1} = \rho / \sigma_1 \dots \sigma_{i-1} = \rho_1 S^{\star\star} \rho_2$ we have that $\mathsf{Fam}_\simeq(\sigma_1 \dots \sigma_{i-1} \rho_1 S^{\star\star}) \in \mathcal{F}$. Collecting all the facts we have already established above, we have that $\mathsf{Fam}_\simeq(\sigma_1 \dots \sigma_{i-1} S_1^i \dots S_j^i) = \mathsf{Fam}_\simeq(\sigma_1 \dots \sigma_{i-1} S^\star) = \mathsf{Fam}_\simeq(\sigma_1 \dots \sigma_{i-1} \rho_1 S^{\star\star}) \in \mathcal{F}$, which concludes the proof of the claim.

To conclude the proof of the lemma, note that the set $\mathcal{F}$ is finite since ρ is finite. By FFD, this implies that there cannot be infinite derivations contracting redexes whose family is in $\mathcal{F}$. Therefore D must be finite. □

By definition, a uniform multi-selection strategy $\mathbb{M}$, when given two permutation equivalent derivations, always selects the same multistep. It, in fact, yields the same multiderivation.

Lemma 7.36. *Let $\mathbb{M}$ be a uniform multi-selection strategy in a DFS, and let ρ, σ be finite derivations. If $\rho \equiv \sigma$ then $\mathbb{M}^\star(\rho) = \mathbb{M}^\star(\sigma)$.*

Proof. By Lem. 7.35, we know that $\mathbb{M}^\star(\rho)$ must be finite. We proceed by induction on the length of $\mathbb{M}^\star(\rho)$:

1. *Empty,* $\mathbb{M}^\star(\rho) = \epsilon$. Then $\rho = \epsilon$, so $\sigma = \epsilon$ and we have $\mathbb{M}^\star(\rho) = \epsilon = \mathbb{M}^\star(\sigma)$.

2. *Non-empty.* Then ρ is non-empty, so σ must be also non-empty, and we have that $\mathbb{M}^\star(\rho) = \mathbb{M}(\rho)\,\mathbb{M}^\star(\rho/\mathbb{M}(\rho))$ and $\mathbb{M}^\star(\sigma) = \mathbb{M}(\sigma)\,\mathbb{M}^\star(\sigma/\mathbb{M}(\sigma))$. First, since $\rho \equiv \sigma$ and $\mathbb{M}$ is a uniform selection strategy, we have $\mathbb{M}(\rho) = \mathbb{M}(\sigma)$. Moreover, the tail of $\mathbb{M}^\star(\rho)$ is of the form $\mathbb{M}^\star(\rho/\mathbb{M}(\rho))$, and it is strictly shorter than $\mathbb{M}^\star(\rho)$. So we can apply the *i.h.* on the tails of $\mathbb{M}^\star(\rho)$ and $\mathbb{M}^\star(\sigma)$. The *i.h.* states:

$$\rho/\mathbb{M}(\rho) \equiv \sigma/\mathbb{M}(\sigma) \implies \mathbb{M}^\star(\rho/\mathbb{M}(\rho)) = \mathbb{M}^\star(\sigma/\mathbb{M}(\sigma))$$

To conclude, we are left to show that $\rho/\mathbb{M}(\rho) \equiv \sigma/\mathbb{M}(\sigma)$ holds. This is an immediate consequence of the fact that $\rho \equiv \sigma$, since the projections of permutation equivalent derivations are again equivalent (Prop. 2.63).

□

Lemma 7.37. *Let $\mathbb{M}$ be a multi-selection strategy in a DFS, and ρ a finite derivation. Then $\rho \equiv \partial\mathbb{M}^\star(\rho)$.*

Proof. By Lem. 7.35, we have that $\mathbb{M}^\star(\rho)$ must be finite. We proceed by induction on the length of the multiderivation $\mathbb{M}^\star(\rho)$.

1. *Empty,* $\mathbb{M}^\star(\rho) = \epsilon$. Then $\rho = \epsilon$, so $\rho = \partial\epsilon = \partial\mathbb{M}^\star(\rho)$.

2. *Non-empty.* Let $\mathcal{M} = \mathbb{M}(\rho)$ be the first multistep selected by the strategy. Then $\mathbb{M}^\star(\rho) = \mathcal{M}\,\mathbb{M}^\star(\rho/\mathcal{M})$. To show that $\rho \equiv \partial\mathbb{M}^\star(\rho)$, by Lem. 2.59, it suffices for us to check that they are projection equivalent, *i.e.* that $\rho \sqsubseteq \partial\mathbb{M}^\star(\rho) \sqsubseteq \rho$.

   ($\sqsubseteq$) Let us check that $\rho/\partial\mathbb{M}^\star(\rho) = \epsilon$.

   $$\begin{array}{lll} & \rho/\partial\mathbb{M}^\star(\rho) & \\ = & \rho/\partial(\mathcal{M}\,\mathbb{M}^\star(\rho/\mathcal{M})) & \\ = & \rho/(\partial\mathcal{M})\,(\partial\mathbb{M}^\star(\rho/\mathcal{M})) & \\ = & (\rho/\partial\mathcal{M})/\partial\mathbb{M}^\star(\rho/\mathcal{M}) & \text{since } \alpha/\beta\gamma = (\alpha/\beta)/\gamma \\ = & \epsilon & \text{since by } \textit{i.h.}\ \rho/\mathcal{M} \equiv \partial\mathbb{M}^\star(\rho/\mathcal{M}) \end{array}$$

   ($\sqsupseteq$) Since $\mathbb{M}$ is a multi-selection strategy, we have that $\mathcal{M}/\rho = \varnothing$. Let us check that $\partial\mathbb{M}^\star(\rho)/\rho = \epsilon$.

   $$\begin{array}{lll} & (\partial\mathbb{M}^\star(\rho))/\rho & \\ = & (\partial(\mathcal{M}\,\mathbb{M}^\star(\rho/\mathcal{M})))/\rho & \\ = & (\partial\mathcal{M})\,(\partial\mathbb{M}^\star(\rho/\mathcal{M}))/\rho & \\ = & ((\partial\mathcal{M})/\rho)\,((\partial\mathbb{M}^\star(\rho/\mathcal{M}))/(\rho/\partial\mathcal{M})) & \text{since } \alpha\beta/\gamma = (\alpha/\beta)(\gamma/(\beta/\alpha)) \\ = & (\partial\mathbb{M}^\star(\rho/\mathcal{M}))/(\rho/\partial\mathcal{M}) & \text{since } \mathcal{M}/\rho = \varnothing, \text{ so } (\partial\mathcal{M})/\rho = \epsilon \\ = & (\partial\mathbb{M}^\star(\rho/\mathcal{M}))/(\rho/\mathcal{M}) & \text{since } \rho/\mathcal{M} \text{ stands for } \rho/\partial\mathcal{M} \\ = & \epsilon & \text{since by } \textit{i.h.}\ \rho/\mathcal{M} \equiv \partial\mathbb{M}^\star(\rho/\mathcal{M}) \end{array}$$

□

Definition 7.38 (Standard multiderivation)**.** A multiderivation D is $\mathbb{M}$-*standard* if $\mathbb{M}^\star(\partial D) = D$.

Proposition 7.39 (Standardization for DFSs)**.** *Let $\mathbb{M}$ be a uniform multi-selection strategy in a DFS. For any finite derivation ρ there exists a unique multiderivation D such that $\rho \equiv \partial D$ and D is $\mathbb{M}$-standard. Namely, $D = \mathbb{M}^\star(\rho)$.*

Proof. We prove the result in two parts:

1. *Existence.* First note that $\rho \equiv \partial\mathbb{M}^\star(\rho)$ by Lem. 7.37. To see that $\mathbb{M}^\star(\rho)$ is $\mathbb{M}$-standard, apply Lem. 7.36 on the fact that $\partial\mathbb{M}^\star(\rho) \equiv \rho$ to conclude that $\mathbb{M}^\star(\partial\mathbb{M}^\star(\rho)) = \mathbb{M}^\star(\rho)$, as required.

2. *Uniqueness.* Suppose that there is a multiderivation E such that $\rho \equiv \partial E$ and E is $\mathbb{M}$-standard. We claim that $E = \mathbb{M}^\star(\rho)$. By applying Lem. 7.36 on the fact that $\partial E \equiv \rho$, we obtain that $\mathbb{M}^\star(\partial E) = \mathbb{M}^\star(\rho)$. Finally, since E is $\mathbb{M}$-standard, $E = \mathbb{M}^\star(\partial E) = \mathbb{M}^\star(\rho)$ and we conclude.

□

Example 7.40 (Standardization in the λ-calculus)**.** *In the λ-calculus, let $\mathbb{M}_{\mathsf{Left}}(\rho) := \{R\}$ where R is the leftmost step such that $R/\rho = \varnothing$, and let $\mathbb{M}_{\mathsf{Par}}(\rho) := \{R \mid R/\rho = \varnothing\}$. It can be checked that $\mathbb{M}_{\mathsf{Left}}$ and $\mathbb{M}_{\mathsf{Par}}$ are uniform multi-selection strategies. Moreover, let $\Delta \to \Delta'$ and let ρ be the derivation:*

$$\rho : (\lambda x.yxx)\,((\lambda x.z)\Delta) \to (\lambda x.yxx)\,((\lambda x.z)\Delta') \to (\lambda x.yxx)\,z \to yzz$$

Then the (leftmost) standard *form of ρ is:*

$$\mathbb{M}^\star_{\mathsf{Left}}(\rho) : (\lambda x.yxx)\,((\lambda x.z)\Delta) \to y((\lambda x.z)\,\Delta)((\lambda x.z)\Delta) \to yz((\lambda x.z)\Delta) \to yzz$$

The parallel standard *form of ρ consists of a single multistep:*

$$\mathbb{M}^\star_{\mathsf{Par}}(\rho) : (\lambda x.yxx)\,((\lambda x.z)\Delta) \Rightarrow yzz$$

Standardization in the LSC without gc

In this subsection we apply the previous standardization result (Prop. 7.39) to the LSC without gc.

Definition 7.41 (Arbitrary selector)**.** Let $\mathsf{Out}(t)$ denote the set of steps whose source is a term t in the LSC without gc, and let $<_t$ be an arbitrary strict partial order on $\mathsf{Out}(t)$. We write $<$ for the function that, for each term $t \in \mathcal{T}$, yields a partial order $<_t \subseteq \mathsf{Out}(t) \times \mathsf{Out}(t)$.

The *arbitrary selector on* $<$ is written $\mathbb{M}_<$ and defined as the following function, taking a non-empty derivation and returning a finite set of coinitial steps:

$$\mathbb{M}_<(\rho) \stackrel{\text{def}}{=} \{R \mid R/\rho = \varnothing \text{ and } R \text{ is minimal}\}$$

By *minimal* we mean that there is no step R' such that $R'/\rho = \varnothing$ and $R' <_{\mathsf{src}(\rho)} R$.

Note that $\mathbb{M}_<(\rho)$ is a non-empty finite set. To see this, note that the set $X = \{R \mid R/\rho = \varnothing\}$ is non-empty, because $R/\rho = \varnothing$ if R is taken to be the first step of the derivation ρ. Moreover, the set X is finite, because the LSC is finitely branching. Hence X must have at least one minimal element. Moreover:

Lemma 7.42. *$\mathbb{M}_<$ is a uniform multi-selection strategy.*

Proof. Let us check that $\mathbb{M}_<$ is a multi-selection strategy and that it is uniform:

1. *$\mathbb{M}_<$ is a multi-selection strategy.* Let ρ be a non-empty reduction sequence. Recall that a function $\mathbb{M}$ is a selection strategy if $\mathbb{M}(\rho)$ is a non-empty multistep $\mathcal{M}$ coinitial to ρ such that $\mathcal{M}/\rho = \varnothing$ and $\mathcal{M} \lhd \rho$.

   In our case, we have constructed $\mathbb{M}_<(\rho)$ to be a non-empty multistep coinitial to ρ (Def. 7.41). Moreover, also by construction, any step $R \in \mathbb{M}_<(\rho)$ verifies $R/\rho = \varnothing$, so indeed $\mathbb{M}_<(\rho)/\rho = \varnothing$. Moreover, in the LSC without gc there is no erasure, so all steps are *essential*. That is, if $R/\rho = \varnothing$ then $R \lhd \rho$. Hence we have that $\mathbb{M}_<(\rho) \lhd \rho$, as required.

2. *$\mathbb{M}_<$ is uniform.* Let $\rho \equiv \sigma$, and let us check that $\mathbb{M}_<(\rho) = \mathbb{M}_<(\sigma)$. It suffices to show that the set $A_\rho = \{R \mid R/\rho = \varnothing\}$ coincides with the set $A_\sigma = \{R \mid R/\sigma = \varnothing\}$, since $\mathbb{M}_<(\rho)$ is the subset of the minimal elements of A_ρ and $\mathbb{M}_<(\sigma)$ is the subset of the minimal elements of A_σ.

   Note that:

$$\begin{array}{rcll} R \in A_\rho & \iff & R/\rho = \varnothing & \\ & \iff & R/\sigma = \varnothing & \text{since } \rho \equiv \sigma \\ & \iff & R \in A_\sigma & \end{array}$$

   So $A_\rho = A_\sigma$, as wanted.

□

Corollary 7.43 (Standardization by arbitrary selection for the LSC without gc)**.** *Let $\mathbb{M}_<$ be the arbitrary selector on $<$. For each finite sequence ρ in the LSC without* gc*, there is a unique multiderivation D such that $\rho \equiv \partial D$ and D is $\mathbb{M}_<$-standard. Moreover, if the ordering function $<$ is computable, then D is computable from ρ, namely $D = \mathbb{M}^\star_<(\rho)$.*

Proof. This is a consequence of the standardization result for DFSs (Prop. 7.39) and the fact that $\mathbb{M}_<$ is a uniform multi-selection strategy (Lem. 7.42). Moreover, it is clear by definition that $\mathbb{M}^\star_<$ is computable if the ordering function $<$ is computable. □

Example 7.44 (Standardization in the LSC without gc)**.** *In the LSC without* gc*, let $\rho : x[x\backslash t] \to x[x\backslash t'] \to t'[x\backslash t'] \to t''[x\backslash t']$, where $t \to t' \to t''$.*

1. *If $<^1$ is the trivial partial order in which every step is incomparable,* i.e. *$R <^1_t S$ never holds, then $\mathbb{M}^\star_{<^1}(\rho) : x[x\backslash t] \Rightarrow t'[x\backslash t'] \to t''[x\backslash t']$. The first step is a proper multistep.*

2. *Let $<^2$ be the total left-to-right order, defined so that $R <^2_t S$ holds whenever R is to the left of S. Then $\mathbb{M}^\star_{<^2}(\rho) : x[x\backslash t] \to t[x\backslash t] \to t'[x\backslash t] \to t'[x\backslash t'] \to t''[x\backslash t']$.*

3. *Let $<^3$ be the total right-to-left order, defined so that $R <^3_t S$ holds if R is to the right of S. Then $\mathbb{M}^{\star}_{<^3}(\rho) = \rho : x[x\backslash t] \to x[x\backslash t'] \to t'[x\backslash t'] \to t''[x\backslash t']$.*

7.6 Normalization of Strategies

A *reduction strategy* is, informally speaking, a restriction on the computational steps that may be performed in a rewriting system. For example, in the λ-calculus, *head reduction* is the restriction of the β-reduction rule that only allows to contract *head redexes*, this is, redexes that lie below a context of the form $\lambda x_1 \dots x_n.\Box\, u_1 \dots u_m$. More precisely, head reduction is defined by the following rewriting rule:

$$\lambda x_1 \dots x_n.(\lambda y.t)s\, u_1 \dots u_m \to_{\mathsf{head}} \lambda x_1 \dots x_n.t\{y := s\}\, u_1 \dots u_m$$

For instance, underlining the contracted redex, the following is a sequence of head reduction steps:

$$\lambda x.\underline{(\lambda y.y)((\lambda y.x)\Omega)} \to_{\mathsf{head}} \lambda x.\underline{(\lambda y.x)\Omega} \to_{\mathsf{head}} \lambda x.x$$

While the following is not a sequence of head reduction steps, because the first step does not contract a head redex:

$$\lambda x.(\lambda y.y)(\underline{(\lambda y.x)\Omega}) \to \lambda x.\underline{(\lambda y.y)x} \to_{\mathsf{head}} \lambda x.x$$

It can be shown that a term has at most one head redex. A term without a head redex is called a *head normal form*. It is well-known that, by repeatedly contracting the head redex, one reaches a head normal form if possible. More precisely, one has the following result:

Proposition 7.45 (Head reduction is head normalizing in the λ-calculus)**.** *Suppose that t has a head normal form, that is, there exists a head normal form s such that $t \leftrightarrow^*_\beta s$. Then there is no infinite head reduction $t \to_{\mathsf{head}} t_1 \to_{\mathsf{head}} t_2 \dots$.*

Proof. See [130, Corollary 1.5.12 (i)]. □

Observe that a term, such as $\Omega = (\lambda x.xx)(\lambda x.xx)$, may not have a head normal form, in which case it is impossible for any strategy to reach a head normal form.

As evidenced in the title of the statement, the result given in Prop. 7.45 is known as the fact that head reduction is *head normalizing*. In general, if $\mathcal{X}$ is a set of answers, a strategy is said to be $\mathcal{X}$-normalizing if repeatedly contracting a step according to the strategy leads to a term in the set $\mathcal{X}$, whenever possible.

In this section, we give sufficient conditions under which reduction strategies are $\mathcal{X}$-normalizing in Deterministic Family Structures. The proof of normalization relies on the Finite Family Developments property. As a consequence, we conclude that two specific strategies, *call-by-name* and *linear call-by-need*, are normalizing in the LSC without gc.

Many normalization results have been studied before. In particular, we should mention that Glauert and Khasidashvili prove a Relative Normalization result [61, Theorem 4.1] for

Deterministic Family Structures, which ensures that any reduction contracting $\mathcal{X}$-needed steps (*cf.* Def. 7.18) reaches a term in $\mathcal{X}$ if possible. The normalization result for DFSs that we state and prove below is a particular case, *i.e.* it is a weaker result than Glauert and Khasidashvili's Relative Normalization. The advantage is that our weaker result only requires to check a number of local syntactic conditions on rewriting diagrams in order to ensure that a strategy is $\mathcal{X}$-normalizing.

Normalization in Deterministic Family Structures

In this subsection we prove a normalization result for Deterministic Family Structures. The main result of this subsection is Prop. 7.54, in which we give sufficient conditions for a strategy to be $\mathcal{X}$-normalizing. We begin by giving formal definitions of all the required notions.

Definition 7.46 (Sub-ARS)**.** A *sub-ARS* of an ARS $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ is an ARS $\mathcal{B} = (\mathsf{Obj}', \mathsf{Stp}', \mathsf{src}', \mathsf{tgt}')$ such that $\mathsf{Obj}' \subseteq \mathsf{Obj}$, $\mathsf{Stp}' \subseteq \mathsf{Stp}$, and moreover the functions $\mathsf{src}', \mathsf{tgt}'$ are the restrictions of $\mathsf{src}, \mathsf{tgt}$ to Stp'. A sub-ARS $\mathcal{B}$ is *closed* if the set $\mathsf{NF}(\mathcal{B})$ is closed by reduction, *i.e.* if $x \to_{\mathcal{A}} y$ and $x \in \mathsf{NF}(\mathcal{B})$ then $y \in \mathsf{NF}(\mathcal{B})$.

Definition 7.47 (Strategy)**.** A *strategy* in an ARS $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ is a sub-ARS $\mathcal{B} = (\mathsf{Obj}', \mathsf{Stp}', \mathsf{src}', \mathsf{tgt}')$ having the same objects, *i.e.* $\mathsf{Obj} = \mathsf{Obj}'$, and the same normal forms, *i.e.* $\mathsf{NF}(\mathcal{A}) = \mathsf{NF}(\mathcal{B})$.

Remark 7.48. Any sub-ARS $\mathcal{B}$ can be extended to a strategy $\mathbb{S}_{\mathcal{B}}$ by adjoining the steps going out from normal forms, *i.e.* by setting $\mathsf{Stp}(\mathbb{S}_{\mathcal{B}}) := \mathsf{Stp}(\mathcal{B}) \cup \{R \in \mathsf{Stp}(\mathcal{A}) \mid \mathsf{src}(R) \in \mathsf{NF}(\mathcal{B})\}$. Note in particular that if $\mathcal{B}$ is already a strategy then $\mathbb{S}_{\mathcal{B}} = \mathcal{B}$.

Example 7.49. *In the λ-calculus, the notion of head reduction $\to_{\mathsf{head}}$ is not strictly speaking a strategy, because the set of β-normal forms does not coincide with the set of head normal forms.*

Head reduction can be extended to a strategy $\mathbb{S}_{\mathsf{head}}$ in such a way that an arbitrary β-step $R : t \to_\beta s$ is in the strategy $\mathbb{S}_{\mathsf{head}}$ whenever R contracts a head redex or, alternatively, t is a head normal form.

Definition 7.50 ($\mathcal{X}$-normalizing strategy)**.** Let $\mathcal{X}$ be a superset of the normal forms of $\mathcal{A}$. A strategy $\mathbb{S}$ is said to be *$\mathcal{X}$-normalizing* if for every object x such that there exists a reduction $x \twoheadrightarrow_{\mathcal{A}} y \in \mathcal{X}$, every maximal reduction from x in the strategy $\mathbb{S}$ contains an object in $\mathcal{X}$.

The following notion of *residual-invariance* is the key notion to give a sufficient condition for a strategy to be $\mathcal{X}$-normalizing.

Definition 7.51 (Residual-invariance)**.** Let $\mathcal{A}$ be an axiomatic rewriting system (including the notion of residual). A sub-ARS $\mathcal{B}$ of $\mathcal{A}$ is *residual-invariant* if for any steps R and S such that $R \in \mathcal{B}$ and $S \neq R$, there exists a step $R' \in \mathbb{S}$ such that $R' \in R/S$.

Example 7.52. *In the λ-calculus, the* leftmost outermost *strategy $\mathbb{S}_{\mathsf{LO}}$ is the strategy that only allows contracting the leftmost outermost step,* i.e. *the step contracting the redex whose λ is more to the left. It is easy to check that $\mathbb{S}_{\mathsf{LO}}$ is residual-invariant, because the residual of a leftmost outermost step is again leftmost outermost.*

The following is a straightforward lemma regarding residual-invariance.

Lemma 7.53 (Steps of residual-invariant sub-ARSs are preserved in DFSs)**.** *Let $F = (\mathcal{A}, \simeq, \hookrightarrow)$ be a DFS and suppose that $\mathcal{B}$ is a residual-invariant sub-ARS of $\mathcal{A}$. Let ρR be a redex with history such that R is in $\mathcal{B}$, and let σ be any finite reduction coinitial to R. Let us also suppose that σ does not contract redexes in the family of ρR. More precisely, let us suppose that whenever σ can be written as $\sigma_1 S \sigma_2$ then $\rho R \not\simeq \rho\sigma_1 S$. Then R has a residual $R' \in R/\sigma$ in $\mathcal{B}$.*

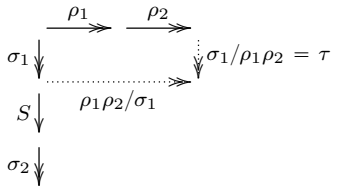
Proof. By induction on σ. If σ is empty it is immediate. If $\sigma = T\,\tau$, we know that $\rho R \not\simeq \rho T$ by hypothesis, so in particular $R \neq T$. Since $\mathcal{B}$ is residual-invariant this means that there exists a step $R' \in \mathcal{B}$ such that $R \langle T \rangle R'$. Moreover, by the Copy axiom $\rho R \simeq \rho T R'$. By *i.h.* on the derivation τ and the redex with history $\rho T R'$ we conclude that there is a step $R'' \in \mathcal{B}$ such that $R' \langle \tau \rangle R''$. So $R \langle T\tau \rangle R''$ and we are done. □

We turn to the main result of this subsection.

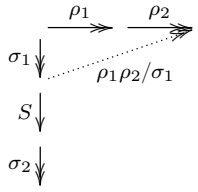
Proposition 7.54 (Normalization for DFSs)**.** *Let $\mathcal{B}$ be a closed residual-invariant sub-ARS in a Deterministic Family Structure. Then the corresponding strategy $\mathbb{S}_{\mathcal{B}}$ is $\mathsf{NF}(\mathcal{B})$-normalizing.*

Proof. Let ρ_1 be a derivation $x \twoheadrightarrow_{\mathcal{A}} y \in \mathsf{NF}(\mathcal{B})$ and consider a maximal derivation σ starting from x in the strategy $\mathbb{S}_{\mathcal{B}}$. We must show that σ contains a term in $\mathsf{NF}(\mathcal{B})$. Let $\mathcal{F}$ be the set of families of all the redexes contracted along ρ_1. The set $\mathcal{F}$ is finite, so by the FFD axiom, the derivation ρ_1 can be extended to a complete family development $\rho_1\rho_2$ of $\mathcal{F}$.

By contradiction, suppose that the reduction sequence σ has no terms in $\mathsf{NF}(\mathcal{B})$. Then σ is contained in the sub-ARS $\mathcal{B}$, and it is infinite. By the FFD axiom, σ cannot be a family development of $\mathcal{F}$, so there must be at least one redex whose family is not in $\mathcal{F}$. Let S be the first such step, *i.e.* let us write σ as $\sigma_1 S \sigma_2$ where σ_1 is a family development of $\mathcal{F}$ and $\mathsf{Fam}_{\simeq}(\sigma) \notin \mathcal{F}$. The situation is as follows, closing the square with the derivations $\rho_1\rho_2/\sigma_1$ and $\sigma_1/\rho_1\rho_2$:



First we claim that the derivation $\tau = \sigma_1/\rho_1\rho_2$ is actually empty. Indeed, by the Copy axiom the families of all the redexes contracted along $\sigma_1/\rho_1\rho_2$ are contained in the families of all the redexes contracted along σ_1. In particular, $\rho_1\rho_2\tau$ is a family development of $\mathcal{F}$. If τ were not empty, it would mean that $\tau = T\tau'$, where $\mathsf{Fam}_{\simeq}(\rho_1\rho_2 T) \in \mathcal{F}$. This contradict the fact that $\rho_1\rho_2$ is a complete family development, as it can be extended with T, so T is indeed empty. This means that the diagram is as follows:

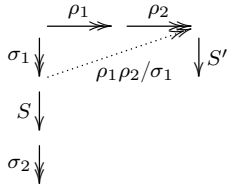


By the Copy axiom, we know that the families of all the redexes contracted along $\rho_1\rho_2/\sigma_1$ are contained in the families of all the redexes contracted along $\rho_1\rho_2$. In particular, $\sigma_1(\rho_1\rho_2/\sigma_1)$ is a family development of $\mathcal{F}$.

By Lem. 7.53, we obtain that there must exist a step $S' \in \mathcal{B}$ such that $S \langle \rho_1\rho_2/\sigma_1 \rangle S'$. To be able to apply Lem. 7.53 note that:

- S is a step in the sub-ARS $\mathcal{B}$;
- by hypothesis, the sub-ARS $\mathcal{B}$ is residual-invariant;
- the derivation $\rho_1\rho_2/\sigma_1$ does not contract redexes in the family of $\sigma_1 S$ since $\mathsf{Fam}_{\simeq}(\sigma_1 S) \notin \mathcal{F}$ while $\sigma_1(\rho_1\rho_2/\sigma_1)$ is a family development of $\mathcal{F}$.

So the situation is:



Finally, recall that $\mathrm{tgt}(\rho_1) \in \mathsf{NF}(\mathcal{B})$, and that, by hypothesis, $\mathcal{B}$ is *closed* residual-invariant, which means that the set $\mathsf{NF}(\mathcal{B})$ is closed by reduction. So $\mathrm{tgt}(\rho_2) \in \mathsf{NF}(\mathcal{B})$, contradicting the fact that there is an outgoing step S' in the sub-ARS $\mathcal{B}$. We conclude that σ must be have a term in $\mathsf{NF}(\mathcal{B})$, as required. □

Normalization in the LSC without gc

In the following subsections, we give the definition of two strategies in the LSC without gc, and we prove that they are normalizing.

First, we study *head linear reduction*, also known as *call-by-name*, in the LSC without gc. As we see in Chapter 3, this strategy is in close correspondence with evaluation of λ-terms in the Krivine Abstract Machine. Moreover, this strategy is closely related with Vincent Danos and Laurent Regnier's notion of head linear reduction in the λ-calculus [47].

Second, we define a new strategy that we baptize *needed linear reduction*. Needed linear reduction is very similar to the (weak) call-by-need strategy that we study in Chapters 3 and 4, with the slight difference that the linear substitution rule that we study here is of the form given in (7.1) below, rather than of the form given in (7.2) below.

$$\mathtt{C}\langle x\rangle[x\backslash \mathtt{vL}] \to \mathtt{C}\langle \mathtt{vL}\rangle[x\backslash \mathtt{vL}] \tag{7.1}$$

$$\mathtt{C}\langle x\rangle[x\backslash \mathtt{vL}] \to \mathtt{C}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\mathtt{L} \tag{7.2}$$

The advantage of the weak call-by-need strategy, as given in (7.2), is that it only copies the subterm v, sharing the substitution context L. Moreover, (7.2) corresponds more closely with Ariola et al.'s established notion of call-by-need [12, 113]. Unfortunately, the weak call-by-need is not a sub-ARS of the LSC, so it would not be possible to apply our results directly to the variant of weak call-by-need given by (7.2) without redoing some of the work that we

have done in previous sections. For example, one should prove that an adapted variant of the LSC with the sharing linear substitution rule also forms a Deterministic Family Structure. We leave this task as future work, restricting our attention to the variant of call-by-need given by (7.1) only.

Normalization of head linear reduction

In this subsection, we recall the definition of head linear reduction (call-by-name), and we prove that it is normalizing.

Definition 7.55 (Head linear reduction and head linear normal forms)**.** Head linear reduction is the sub-ARS $\mathbb{HL}$ of the LSC without gc that selects the (unique) db or ls step whose anchor is currently below a weak head context. Weak head contexts are defined by the grammar:

$$H ::= \Box \mid H\,t \mid H[x\backslash t]$$

The set of head linear normal forms HLNF is defined as the set of terms generated by the grammar:

$$\begin{array}{rcll} A & ::= & (\lambda x.t)\mathtt{L} & \\ & \mid & H\langle\langle x\rangle\rangle & \text{where } H \text{ does not bind } x \end{array}$$

Terms of the form $(\lambda x.t)\mathtt{L}$ are called *answers*, and terms of the form $H\langle\langle x\rangle\rangle$ are called *head structures* (or just *structures* if clear from the context). The variable x is called the *head variable* of a structure $H\langle\langle x\rangle\rangle$.

Corollary 7.56 (Head linear reduction is HLNF-normalizing — ♣ Coro. A.113)**.** *The strategy $\mathbb{S}_{HL}$ associated to the sub-ARS $\mathbb{HL}$ is* HLNF*-normalizing.*

Proof. A consequence of the previous proposition (Prop. 7.54), using the fact that the LSC without gc is a DFS (Thm. 7.13). It suffices to show that $\mathbb{HL}$ is a closed residual-invariant sub-ARS of the LSC without gc, and that $\mathsf{NF}(\mathbb{HL}) = \mathsf{HLNF}$. See the appendix for the proof of these facts. □

Example 7.57. *The following is a head linear reduction reaching a term in* HLNF*.*

$$\begin{array}{rcl} (\lambda x.xx)((\lambda y.y)(\lambda z.z)) & \to & (xx)[x\backslash(\lambda y.y)(\lambda z.z)] \\ & \to & ((\lambda y.y)(\lambda z.z)x)[x\backslash(\lambda y.y)(\lambda z.z)] \\ & \to & (y[y\backslash\lambda z.z]x)[x\backslash(\lambda y.y)(\lambda z.z)] \\ & \to & ((\lambda z.z)[y\backslash\lambda z.z]x)[x\backslash(\lambda y.y)(\lambda z.z)] \\ & \to & z[z\backslash x][y\backslash\lambda z.z][x\backslash(\lambda y.y)(\lambda z.z)] \\ & \to & x[z\backslash x][y\backslash\lambda z.z][x\backslash(\lambda y.y)(\lambda z.z)] \\ & \to & ((\lambda y.y)(\lambda z.z))[z\backslash x][y\backslash\lambda z.z][x\backslash(\lambda y.y)(\lambda z.z)] \\ & \to & y[y\backslash\lambda z.z][z\backslash x][y\backslash\lambda z.z][x\backslash(\lambda y.y)(\lambda z.z)] \\ & \to & (\lambda z.z)[y\backslash\lambda z.z][z\backslash x][y\backslash\lambda z.z][x\backslash(\lambda y.y)(\lambda z.z)] \end{array}$$

Normalization of needed linear reduction

In this subsection, we define a variant of call-by-need we call *needed linear reduction*, and we prove that it is normalizing.

Definition 7.58 (Needed linear reduction and needed linear normal forms)**.** Needed linear reduction is the sub-ARS $\mathbb{NL}$ of the LSC without gc defined as follows. Needed evaluation contexts are defined by the grammar

$$\mathtt{N} ::= \Box \mid \mathtt{N}\, t \mid \mathtt{N}[x\backslash t] \mid \mathtt{N}\langle\!\langle x \rangle\!\rangle [x\backslash \mathtt{N}]$$

The reduction rule $\rightarrow_{\mathbb{NL}}$ is the union of the usual db rule, and the lsnl rule

$$\mathtt{N}\langle\!\langle x \rangle\!\rangle [x\backslash \mathtt{vL}] \mapsto_{\mathtt{lsnl}} \mathtt{N}\langle \mathtt{vL} \rangle [x\backslash \mathtt{vL}]$$

both rewriting rules are closed by need contexts.

The set of *needed linear normal forms* NLNF is defined by the grammar $A ::= (\lambda x.t)\mathtt{L} \mid \mathtt{N}\langle\!\langle x \rangle\!\rangle$. Terms of the form $(\lambda x.t)\mathtt{L}$ are called *answers*, and $\mathtt{N}\langle\!\langle x \rangle\!\rangle$ are called *structures*. In structures, N does not bind x, the latter called its *needed variable*.

Corollary 7.59 (Needed linear reduction is NLNF-normalizing — ♣ Coro. A.115)**.** *The strategy* $\mathbb{S}_{\mathbb{NL}}$ *associated to the sub-ARS* $\mathbb{NL}$ *is* NLNF*-normalizing.*

Proof. A consequence of the previous proposition (Prop. 7.54), using the fact that the LSC without gc is a DFS (Thm. 7.13). It suffices to show that $\mathbb{NL}$ is a closed residual-invariant sub-ARS of the LSC without gc, and that $\mathsf{NF}(\mathbb{NL}) = \mathsf{NLNF}$. See the appendix for the proof of these facts. □

Example 7.60. *The following is a needed linear reduction reaching a term in* NLNF*.*

$$\begin{array}{rcl}
(\lambda x.xx)((\lambda y.y)(\lambda z.z)) & \rightarrow & (xx)[x\backslash(\lambda y.y)(\lambda z.z)] \\
& \rightarrow & (xx)[x\backslash y[y\backslash\lambda z.z]] \\
& \rightarrow & (xx)[x\backslash(\lambda z.z)[y\backslash\lambda z.z]] \\
& \rightarrow & ((\lambda z.z)x)[x\backslash\lambda z.z][y\backslash\lambda z.z] \\
& \rightarrow & z[z\backslash x][x\backslash\lambda z.z][y\backslash\lambda z.z] \\
& \rightarrow & z[z\backslash\lambda z.z][x\backslash\lambda z.z][y\backslash\lambda z.z] \\
& \rightarrow & (\lambda z.z)[z\backslash\lambda z.z][x\backslash\lambda z.z][y\backslash\lambda z.z]
\end{array}$$

Chapter 8

Conclusion

In this thesis we have used calculi with explicit substitutions at a distance, and in particular the Linear Substitution Calculus, to study evaluation strategies. Three main topics have been addressed:

1. We have showed that some of these evaluation strategies—call-by-name, call-by-value, call-by-need, and strong call-by-name—distill the behaviour of abstract machines, and that they are *reasonable* in terms of time complexity. This methodology has allowed us to revisit some abstract machines known from the literature (such as the Krivine abstract machine or Leroy's ZINC machine), as well as to conceive new abstract machines.

2. We have extended the call-by-need evaluation strategy to a strong setting. Our main result is the *completeness* of strong call-by-need, which relies on a recent technique by Kesner, based on using non-idempotent intersection type systems to characterize weak normalization.

3. We have studied the theory of redex families in the LSC. For this, we have proposed a labeled variant of the LSC, following Lévy's work on the λ-calculus. This theory provides us with results about the *optimal* evaluation strategy, and also gives us new proofs of standardization in the LSC, and normalization of the call-by-need strategy.

In the following sections we describe two concrete topics for future work.

8.1 An Abstract Machine for Strong Call-by-Need Reduction

In Chapter 3, we defined abstract machines for evaluation according to the call-by-name, call-by-value, call-by-need, and strong call-by-name strategies. Moreover, in Chapter 4, we defined a strong call-by-need strategy. It is only natural to wonder what an abstract machine for evaluation according to this strong call-by-need strategy would look like.

In this section, we propose an abstract machine for strong call-by-need evaluation.

Definition 8.1 (An abstract machine for strong call-by-need evaluation)**.** Abstract machine states are quadruples $\langle\ \pi \mid \varphi\ t \mid E\ \rangle$ comprising: a *stack* π, a *phase* φ, a *term* t, and an *environment* E. Terms are usual terms of the LSC. The phase is a boolean flag which may be either $\Uparrow$ or $\Downarrow$. The stack represents the current evaluation context, in a way reminiscent of *zippers* [77]. Phases, stacks and environments are defined by the following grammars:

φ	$::=$	$\Uparrow$	Going up (normal form found)
	$\mid$	$\Downarrow$	Evaluation phase
π	$::=$	ϵ	Empty stack (focus at toplevel)
	$\mid$	$\mathsf{a}(t) : \pi$	Argument (focus on the left of an application)
	$\mid$	$\mathsf{d}(t) : \pi$	Data structure (focus on the right of an application)
	$\mid$	$\mathsf{h}(E_1,\, x) : \pi$	Heap (focus on the right of a substitution)
	$\mid$	$\lambda(x) : \pi$	Lambda (focus under an abstraction)
E	$::=$	ϵ	Empty environment
	$\mid$	$E : [x \mapsto t]$	Mapping
	$\mid$	$E : [\![x \mapsto t]\!]$	Frozen mapping
	$\mid$	$E : \curlywedge(x)$	Scope delimiter

The transition rules of the abstract machine are given below:

$$
\begin{array}{rcl@{\quad}l}
\pi \mid \Downarrow t[x\backslash s] \mid E & \rightsquigarrow & \pi \mid \Downarrow t \mid [x \mapsto s]E & \text{D-Migration} \\
\pi \mid \Downarrow ts \mid E & \rightsquigarrow & \pi\mathsf{a}(s) \mid \Downarrow t \mid E & \text{D-Application} \\
\pi\mathsf{a}(s) \mid \Downarrow \lambda x.t \mid E & \rightsquigarrow & \pi \mid \Downarrow t \mid [x \mapsto s]E & \text{Beta} \\
\pi \mid \Downarrow \lambda x.t \mid E & \rightsquigarrow & \pi\lambda(x) \mid \Downarrow t \mid \curlywedge(x)E & \text{D-Strong} \\
 & & \textit{if } \pi \textit{ does not end with } \mathsf{a}(.) \textit{, or } \mathsf{h}(.,\ .) & \\
\pi \mid \Downarrow x \mid E_1[x \mapsto s]E_2 & \rightsquigarrow & \pi\mathsf{h}(E_1,\, x) \mid \Downarrow s \mid E_2 & \text{Lookup} \\
\pi \mid \Downarrow x \mid E_1[\![x \mapsto s]\!]E_2 & \rightsquigarrow & \pi \mid \Uparrow x \mid E_1[\![x \mapsto s]\!]E_2 & \text{Frozen lookup} \\
\pi\mathsf{h}(E_1,\, x) \mid \Downarrow v \mid E_2 & \rightsquigarrow & \pi \mid \Downarrow v' \mid E_1[x \mapsto v]E_2 & \text{LSV} \\
\pi \mid \Downarrow y \mid E & \rightsquigarrow & \pi \mid \Uparrow y \mid E & \text{Up} \\
 & & \textit{if } y \textit{ has no mapping in } E & \\
\pi\mathsf{a}(s) \mid \Uparrow t \mid E & \rightsquigarrow & \pi\mathsf{d}(t) \mid \Downarrow s \mid E & \text{U-Argument} \\
\pi\mathsf{h}(E_1,\, x) \mid \Uparrow t \mid E_2 & \rightsquigarrow & \pi \mid \Uparrow x \mid E_1[\![x \mapsto t]\!]E_2 & \text{U-Update} \\
\pi\mathsf{d}(t) \mid \Uparrow s \mid E & \rightsquigarrow & \pi \mid \Uparrow ts \mid E & \text{U-Application} \\
\pi\lambda(x) \mid \Uparrow t \mid [y \mapsto s]E & \rightsquigarrow & \pi\lambda(x) \mid \Uparrow t[y\backslash s] \mid E & \text{U-Migration} \\
\pi\lambda(x) \mid \Uparrow t \mid \curlywedge(x)E & \rightsquigarrow & \pi \mid \Uparrow \lambda x.t \mid E & \text{U-Strong}
\end{array}
$$

Example 8.2. *The following is an execution in the strong call-by-need abstract machine. In each step we underline the focus of evaluation,* i.e. *the pattern of the* db *redex or the variable contracted*

by the `ls` *redex:*

		ϵ	$\Downarrow$	$(\lambda x.xx)(\lambda y.z(Iz)y)$	ϵ
D-Application	$\rightsquigarrow$	$\mathsf{a}(\lambda y.z(Iz)y)$	$\Downarrow$	$\lambda x.xx$	ϵ
Beta	$\rightsquigarrow$	ϵ	$\Downarrow$	xx	$[x \mapsto \lambda y.z(Iz)y]$
D-Application	$\rightsquigarrow$	$\mathsf{a}(x)$	$\Downarrow$	x	$[x \mapsto \lambda y.z(Iz)y]$
Lookup	$\rightsquigarrow$	$\mathsf{h}(\epsilon,\, x)$	$\Downarrow$	$\lambda y.z(Iz)y$	ϵ
LSV	$\rightsquigarrow$	ϵ	$\Downarrow$	$\lambda y.z(Iz)y$	$[x \mapsto \lambda y.z(Iz)y]$
D-Strong	$\rightsquigarrow$	$\lambda(y)$	$\Downarrow$	$z(Iz)y$	$\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
D-Application	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)$	$\Downarrow$	$z(Iz)$	$\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
D-Application	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{a}(Iz)$	$\Downarrow$	z	$\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
Up	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{a}(Iz)$	$\Uparrow$	z	$\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
U-Argument	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)$	$\Downarrow$	Iz	$\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
D-Application	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)\,\mathsf{a}(z)$	$\Downarrow$	I	$\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
Beta	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)$	$\Downarrow$	w	$[w \mapsto z]\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
Lookup	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)\,\mathsf{h}(\epsilon,\, w)$	$\Downarrow$	z	$\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
Up	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)\,\mathsf{h}(\epsilon,\, w)$	$\Uparrow$	z	$\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
U-Update	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)$	$\Uparrow$	w	$[\![w \mapsto z]\!]\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
U-Application	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)$	$\Uparrow$	zw	$[\![w \mapsto z]\!]\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
U-Argument	$\rightsquigarrow$	$\lambda(y)\,\mathsf{d}(zw)$	$\Downarrow$	y	$[\![w \mapsto z]\!]\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
Up	$\rightsquigarrow$	$\lambda(y)\,\mathsf{d}(zw)$	$\Uparrow$	y	$[\![w \mapsto z]\!]\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
U-Application	$\rightsquigarrow$	$\lambda(y)$	$\Uparrow$	zwy	$[\![w \mapsto z]\!]\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
U-Migration	$\rightsquigarrow$	$\lambda(y)$	$\Uparrow$	$(zwy)[w\backslash z]$	$\lambda(y)[x \mapsto \lambda y.z(Iz)y]$
U-Strong	$\rightsquigarrow$	ϵ	$\Uparrow$	$\lambda y.(zwy)[w\backslash z]$	$[x \mapsto \lambda y.z(Iz)y]$

We omit a proposal for the decoding of machine states as terms. Proposing an appropriate notion of strong bisimulation $\equiv$ between terms in order to show that this machine simulates the strong call-by-need strategy is left as future work. The question of whether the strong call-by-need strategy can be implemented reasonably is open at the moment of writing this thesis.

8.2 Difficulties Defining an Extraction Procedure

As we detailed in Section 6.1.1, Lévy characterized redex families in the λ-calculus in three ways: (1) by means of the *zig-zag* equivalence relation whose equivalence classes are the redex families; (2) by means of an auxiliary calculus with *labels*, in such a way that two redexes with history are in the same family if and only if they have the same name; and (3) by means of an *extraction procedure*, in such a way that two redexes with history are in the same family if and only if the extraction procedure yields the same canonical representative.

In this section we mention a currently unsolved problem that we have found while attempting to characterize redex families in the LSC. In particular, we propose an extraction procedure but we leave the question of whether it has all the desired properties as an open problem.

To clarify the discussion, let us consider three equivalence relations between redexes with history in the LSC without `gc`:

- **Zig-zag family equivalence ($\simeq_{\mathsf{Z}}$)**, defined as the reflexive–symmetric–transitive closure of the copy relation $\leqslant$ defined in Def. 7.2.

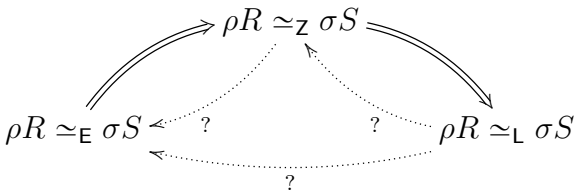
- **Labeling family equivalence ($\simeq_{\mathsf{L}}$)**, defined to hold for two redexes with history if an initially labeled variant gives them the same name. Note that this is precisely the relation $\overset{\mathsf{Fam}}{\simeq}$ defined in Def. 7.7.

- **Extraction family equivalence ($\simeq_{\mathsf{E}}$)**, that we will attempt to define below, defined to hold for two redexes with history if they have the same canonical representative.

The question is now whether these relations all characterize the same notion of redex family. We have already seen that the LSC without gc verifies the Copy axiom (Thm. 7.13), which means that $(\rho R \leqslant \sigma S) \implies (\rho R \simeq_{\mathsf{L}} \sigma S)$. From this fact, by induction on the derivation that $\rho R \simeq_{\mathsf{Z}} \sigma S$, one can easily prove that zig-zag equivalence is contained in labeling equivalence, that is, $(\rho R \simeq_{\mathsf{Z}} \sigma S) \implies (\rho R \simeq_{\mathsf{L}} \sigma S)$.

The converse implication $(\rho R \simeq_{\mathsf{L}} \sigma S) \implies (\rho R \simeq_{\mathsf{Z}} \sigma S)$ is non-trivial. In the λ-calculus, the proof of this fact that we are familiar with relies on the definition of an extraction procedure (see [14, Section 6.2.3]).

For the extraction procedure that we will propose below, it will be easy to prove that extraction equivalence is contained in zig-zag equivalence, *i.e.* the implication $(\rho R \simeq_{\mathsf{E}} \sigma S) \implies (\rho R \simeq_{\mathsf{Z}} \sigma S)$. The picture, at the time of writing this thesis, is currently:



(8.1)

Proposal of an extraction procedure

Before proposing the extraction procedure, we introduce some auxiliary notions.

Definition 8.3 (Non-duplication)**.** We write $\rho \# S$ if ρ does not duplicate S, *i.e.* $\#(S/\rho) = 1$. We say that ρ does not duplicate σ, written $\rho \# \sigma$, according to the following inductive definition:

$$\frac{}{\rho \# \epsilon} \qquad \frac{\rho \# S \quad \rho/S \# \sigma}{\rho \# S\sigma}$$

Remark 8.4. If $\rho \# \sigma$ then σ/ρ has the same length as σ.

Definition 8.5 (Internal derivation)**.** A step R is *internal to a context* $\mathtt{C}$, written $\mathtt{C} \prec R$, whenever the source of R is of the form $\mathtt{C}\langle t\rangle$ and, moreover:

- If R is a db redex, the position of the hole of $\mathtt{C}$ is a prefix of the position of the pattern of the db redex.

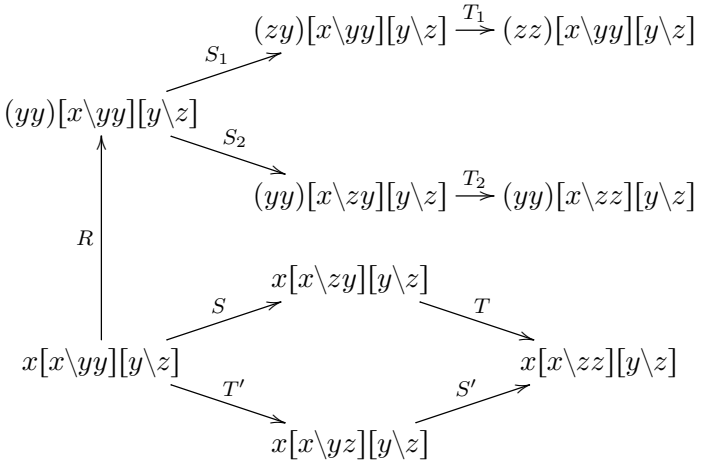
- If R is a ls redex, the position of the hole of $\mathtt{C}$ is a prefix of the position of the variable contracted by the ls redex.

Moreover, a derivation ρ is *internal to a context* $\mathtt{C}$, according to the following inductive definition:

$$\frac{}{\mathtt{C} \prec \epsilon} \qquad \frac{\mathtt{C} \prec R \quad \mathtt{C} \prec \rho}{\mathtt{C} \prec R\rho}$$

If R is a $\mathtt{ls}$ redex and σ is a composable derivation, *i.e.* $\mathrm{tgt}(R) = \mathrm{src}(\sigma)$, the derivation σ is said to be *internal to the subject (resp. argument)* of R, written $R \prec_{\mathtt{sbj}} \sigma$ (resp. $R \prec_{\mathtt{arg}} \sigma$) whenever the redex R is of the form $\mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x \rangle\!\rangle [x \backslash t] \rangle \to \mathtt{C}_1\langle \mathtt{C}_2\langle t \rangle [x \backslash t] \rangle$ and $\mathtt{C}_1\langle \mathtt{C}_2\langle \Box \rangle [x \backslash t] \rangle \prec \sigma$ (resp. $\mathtt{C}_1\langle \mathtt{C}_2\langle t \rangle [x \backslash \Box] \rangle \prec \sigma$).

Example 8.6 (Internal derivations). *For example, consider the following diagram:*



Then $\Box[x \backslash yy][y \backslash z] \prec S_1T_1$ *so* $R \prec_{\mathtt{sbj}} S_1T_1$, *and* $(yy)[x \backslash \Box][y \backslash z] \prec S_2T_2$ *so* $R \prec_{\mathtt{arg}} S_2T_2$.

Note that if $R \prec_i S_i$ for $i \in \{\mathtt{sbj}, \mathtt{arg}\}$, then S_i has an ancestor S_0, that is $S_i \in S_0/R$. Moreover, S_0/R consists of exactly two redexes, namely $S_{\mathtt{sbj}}$ and $S_{\mathtt{arg}}$ such that S_i is internal to i. Also note that S_0 does not duplicate R. This can be justified by the following diagram:

$$\begin{array}{ccc} \mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x \rangle\!\rangle [x \backslash \underline{t}] \rangle & \xrightarrow{R} & \mathtt{C}_1\langle \mathtt{C}_2\langle \underline{t} \rangle [x \backslash \underline{t}] \rangle \\ \downarrow{\scriptstyle S_0} & & \downarrow{\scriptstyle S_{\mathtt{sbj}}} \quad \downarrow{\scriptstyle S_{\mathtt{arg}}} \end{array}$$

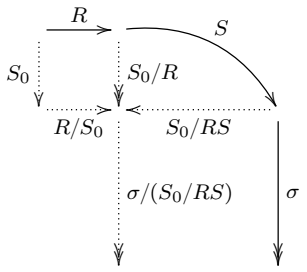
Definition 8.7 (Retraction). If $R \prec_i S_i$ for $i \in \{\mathtt{sbj}, \mathtt{arg}\}$, we write $S_i \leftharpoonup R$ for the (unique) ancestor of S_i, corresponding to S_0 in the diagram above. We call $(S_i \leftharpoonup R)$ the *retract* of S_i before R.

The notion of retract is also extended for derivations. If $R \prec_i \sigma$ for $i \in \{\mathtt{sbj}, \mathtt{arg}\}$, the *retract* of σ before R, written $\sigma \leftharpoonup R$, is defined inductively as follows:

$$\begin{array}{rcl} \epsilon \leftharpoonup R & \stackrel{\mathrm{def}}{=} & \epsilon \\ S\sigma \leftharpoonup R & \stackrel{\mathrm{def}}{=} & S_0\,(\sigma/(S_0/RS) \leftharpoonup R/S_0) \quad \text{where } S_0 = S \leftharpoonup R \end{array}$$

The retraction is well defined as R/S_0 is a single redex, since, as we have already discussed, S_0 does not duplicate R. To see that the inductive definition is in fact well-defined, it can be

checked that $R/S_0 \prec_i \sigma/(S_0/RS)$ and, moreover, that the length of $\sigma/(S_0/RS)$ coincides with the length of σ, so recursion is well-founded. The following diagram illustrates the situation:



Example 8.8 (Retraction)**.** *In the situation of Ex. 8.6, we have that* $(S_1T_1 \leftharpoonup R) = (S_2T_2 \leftharpoonup R) = ST$.

Finally, we may define an *extraction procedure*.

Definition 8.9 (Extraction procedure)**.** Extraction is a rewriting system whose objects are redexes with history. Rewriting steps are given by the two following rules:

$$\begin{array}{lcll} \rho R(\sigma/R) & \rhd & \rho\sigma & \text{if } \sigma \neq \epsilon \text{ and } R \# \sigma \\ \rho R\sigma & \rhd & \rho(\sigma \leftharpoonup R) & \text{if } \sigma \neq \epsilon \text{ and } R \prec_i \sigma \text{ for some } i \in \{\mathtt{sbj}, \mathtt{arg}\} \end{array}$$

Example 8.10. *In the situation of Ex. 8.6, we have that* $RS_1 \rhd S$, $RS_2 \rhd S$, $RS_1T_1 \rhd ST \rhd T'$, *and* $RS_2T_2 \rhd ST \rhd T'$.

It is not hard to show that $\rhd$ is strongly normalizing. Indeed, one can show that if $\rho \rhd \sigma$, then the length of σ is strictly lesser than the length of ρ. Moreover, one may define *extraction family equivalence* by declaring the relation $\rho R \simeq_{\mathsf{E}} \sigma S$ to hold if and only if $\rho R\ (\rhd \cup \rhd^{-1})^*\ \sigma S$. With this definition, it is not hard to show that $(\rho S \simeq_{\mathsf{E}} \sigma S) \implies (\rho S \simeq_{\mathsf{Z}} \sigma S)$.

On the other hand, the two following problems seem to be non-trivial, and we leave them as unsolved conjectures.

Conjecture 8.11. *The extraction procedure* $\rhd$ *is confluent.*

Conjecture 8.12. *The extraction procedure* $\rhd$ *characterizes redex families. More precisely, the implication* $(\rho S \simeq_{\mathsf{L}} \sigma S) \implies (\rho S \simeq_{\mathsf{E}} \sigma S)$ *holds, closing the circle of implications of (8.1).*

Appendix A

Technical appendix

A.1 Proofs of Chapter 3 – Distilling Abstract Machines

A.1.1 Determinism — proof of Prop. 3.11

Proposition A.1 (Full proof of Prop. 3.11—Determinism)**.** *The five reduction strategies of Def. 3.3 are deterministic. In each case, if* $\mathtt{R}_1, \mathtt{R}_2$ *are evaluation contexts,* r_1, r_2 *are* anchors (i.e. *an application that may be contracted by a multiplicative step or a variable that may be contracted by an exponential step), and* $\mathtt{R}_1\langle r_1\rangle = \mathtt{R}_2\langle r_2\rangle$ *then* $\mathtt{R}_1 = \mathtt{R}_2$ *and* $r_1 = r_2$*. So there is at most one way to reduce a term.*

We prove each case separately.

Call-by-Name

Let $t = \mathtt{H}_1\langle r_1\rangle = \mathtt{H}_2\langle r_2\rangle$. By induction on the structure of t. Cases:

- **Variable or an abstraction.** Vacuously true, because there is no redex.
- **Application.** Let $t = su$. Suppose that one of the two evaluation contexts, for instance $\mathtt{H}_1$, is equal to $\Box$. Then, we must have $s = \lambda x.s'$, but in that case it is easy to see that the result holds, because $\mathtt{H}_2$ cannot have its hole to the right of an application (in u) or under an abstraction (in s'). We may then assume that none of $\mathtt{H}_1, \mathtt{H}_2$ is equal to $\Box$. In that case, we must have $\mathtt{H}_1 = \mathtt{H}'_1 u$ and $\mathtt{H}_2 = \mathtt{H}'_2 u$, and we conclude by induction hypothesis.
- **Substitution.** Let $t = s[x\backslash u]$. This case is entirely analogous to the previous one.

Left-to-Right Call-by-Value

We prove the following statement, of which the determinism of the reduction is a consequence.

Lemma A.2. *Let* t *be a term. Then* t *has at most one subterm* s *that verifies both (i) and (ii):*

(i) Either s *is a variable* x*, or* s *is an application* $\mathtt{L}\langle \mathtt{v}\rangle\mathtt{L}'\langle \mathtt{v}'\rangle$*, for* $\mathtt{v}, \mathtt{v}'$ *being values.*

(ii) s *is under a left-to-right call-by-value evaluation context,* i.e. $t = \mathtt{R}\langle s \rangle$.

From the statement it follows that there is at most one $\to$-redex in t, *i.e.* $\to$ is deterministic.

Proof. by induction on the structure of t:

- t **is a variable.** There is only one subterm, under the empty evaluation context.
- t **is an abstraction.** There are no subterms that verify both (i) and (ii), since the only possible evaluation context is the empty one.
- t **is an application** $u\,r$**.** There are three possible situations:
  - *The left subterm* u *is not of the form* $\mathtt{L}\langle \mathtt{v} \rangle$. Then s cannot be at the root, *i.e.* $s \neq t$. Since $u\Box$ is not an evaluation context, s must be internal to $\Box r$, which is an evaluation context. We conclude by *i.h.*.
  - *The left subterm* u *is of the form* $\mathtt{L}\langle \mathtt{v} \rangle$ *with* $\mathtt{v}$ *a value, but the right subterm* r *is not.* Then s cannot be a subterm of u, and also $s \neq t$. Hence, if there is a subterm s as in the statement, it must be internal to the evaluation context $u\Box$. We conclude by *i.h.*.
  - *Both subterms have that form, i.e.* $u = \mathtt{L}\langle \mathtt{v} \rangle$ and $r = \mathtt{L}'\langle \mathtt{v}' \rangle$ with $\mathtt{v}$ and $\mathtt{v}'$ values. The only subterm that verifies both (i) and (ii) is $s = t$.
- t **is a substitution** $u[x\backslash r]$**.** Any occurrence of s must be internal to u (because $u[x\backslash\Box]$ is not an evaluation context). We conclude by *i.h.* that there is at most one such occurrence.

□

Right-to-Left Call-by-Value

Exactly as in the case for left-to-right call-by-value, we prove the following property, from which determinism of the reduction follows.

Lemma A.3. *Let* t *be a term. Then* t *has at most one subterm* s *that verifies both (i) and (ii):*

(i) s *is either a variable* x *or an application* $\mathtt{L}\langle \mathtt{v} \rangle \mathtt{L}'\langle \mathtt{v}' \rangle$*, where* $\mathtt{v}$ *and* $\mathtt{v}'$ *are values.*

(ii) s *is under a right-to-left call-by-value evaluation context,* i.e. $t = \mathtt{R}\langle s \rangle$.

As a corollary, any term t has at most one $\to$-redex.

Proof. By induction on the structure of t:

- **Variable or abstraction.** Immediate.
- **Application.** If $t = u\,r$, there are three cases:
  - *The right subterm* r *is not of the form* $\mathtt{L}'\langle \mathtt{v}' \rangle$. Then s cannot be at the root. Since $\Box\, r$ is not an evaluation context, s must be internal to r and we conclude by *i.h.*.

- *The right subterm r is of the form $\mathtt{L}'\langle\mathtt{v}'\rangle$ but the left subterm u is not.* Again s cannot be at the root. Moreover, r has no applications or variables under an evaluation context. Therefore s must be internal to u and we conclude by *i.h.*.
- *Both subterms have that form, i.e.* $u = \mathtt{L}\langle\mathtt{v}\rangle$ and $r = \mathtt{L}'\langle\mathtt{v}'\rangle$. We first note that u and r have no applications or variables under an evaluation context. The only possibility that remains is that s is at the root, *i.e.* $s = t$.

- **Substitution.** If $t = u[x\backslash r]$ is a substitution, s must be internal to u (because $u[x\backslash\Box]$ is not an evaluation context), and we conclude by *i.h.*.

□

Call-by-Need

We first need an auxiliary result:

Lemma A.4. *Let $t = \mathtt{N}\langle\!\langle x\rangle\!\rangle$ for an evaluation context $\mathtt{N}$. Then:*

1. *for every substitution context $\mathtt{L}$ and abstraction $\mathtt{v}$, one has $t \neq \mathtt{L}\langle\mathtt{v}\rangle$;*
2. *for every evaluation context $\mathtt{R}'$ and variable y, one has that $t = \mathtt{R}'\langle y\rangle$ implies $\mathtt{R}' = \mathtt{N}$ and $y = x$;*
3. *t is a call-by-need normal form.*

Proof. In all items we use a structural induction on $\mathtt{N}$. For item 1:

- $\mathtt{N} = \Box$: obvious.
- $\mathtt{N} = \mathtt{N}_1 s$: obvious.
- $\mathtt{N} = \mathtt{N}_1[y\backslash u]$: suppose that $\mathtt{L} = \mathtt{L}'[y\backslash u]$ (for otherwise the result is obvious); then we apply the induction hypothesis to $\mathtt{N}_1$ to obtain $\mathtt{N}_1\langle x\rangle \neq \mathtt{L}'\langle\mathtt{v}\rangle$.
- $\mathtt{N} = \mathtt{N}_1\langle y\rangle[y\backslash\mathtt{N}_2]$: suppose that $\mathtt{L} = \mathtt{L}'[y\backslash\mathtt{N}_2\langle x\rangle]$ (for otherwise the result is obvious); then we apply the induction hypothesis to $\mathtt{N}_1$ to obtain $\mathtt{N}_1\langle y\rangle \neq \mathtt{L}'\langle\mathtt{v}\rangle$.

For item 2:

- $\mathtt{N} = \Box$: obvious.
- $\mathtt{N} = \mathtt{N}_1 s$: we must necessarily have $\mathtt{R}' = \mathtt{R}'_1 s$ and we conclude by induction hypothesis.
- $\mathtt{N} = \mathtt{N}_1[z\backslash s]$: in principle, there are two cases. First, we may have $\mathtt{R}' = \mathtt{R}'_1[z\backslash s]$, which allows us to conclude immediately by induction hypothesis, as above. The second possibility would be $\mathtt{R}' = \mathtt{R}'_1\langle z\rangle[z\backslash\mathtt{R}'_2]$, with $\mathtt{R}'_2\langle y\rangle = s$, but this is actually impossible. In fact, it would imply $\mathtt{N}_1\langle x\rangle = \mathtt{R}'_1\langle z\rangle$, which by induction hypothesis would give us $z = x$, contradicting the hypothesis $x \in \mathtt{fv}(t)$.

- $\mathtt{N} = \mathtt{N}_1\langle z\rangle[z\backslash\mathtt{N}_2]$: by symmetry with the above case, the only possibility is $\mathtt{R}' = \mathtt{N}_1\langle z\rangle[z\backslash\mathtt{R}'_2]$, which allows us to conclude immediately by induction hypothesis.

For item 3, let r be a redex (*i.e.*, a term matching the left hand side of $\mapsto_{\mathtt{db}}$ or $\mapsto_{\mathtt{lslsv}}$) and let $\mathtt{R}'$ be an evaluation context. We will show by structural induction on $\mathtt{N}$ that $t \neq \mathtt{R}'\langle r\rangle$. We will do this by considering, in each inductive case, all the possible shapes of $\mathtt{R}'$.

- $\mathtt{N} = \Box$: obvious.
- $\mathtt{N} = \mathtt{N}_1 s$: the result is obvious unless $\mathtt{R}' = \Box$ or $\mathtt{R}' = \mathtt{R}'_1 s$. In the latter case, we conclude by induction hypothesis (on $\mathtt{N}_1$). In the former case, since r is a redex, we are forced to have $r = \mathtt{L}\langle\mathtt{v}\rangle s'$ for some abstraction $\mathtt{v}$, substitution context $\mathtt{L}$ and term s'. Now, even supposing $s' = s$, we are still allowed to conclude because $\mathtt{N}_1\langle x\rangle \neq \mathtt{L}\langle\mathtt{v}\rangle$ by item 1.
- $\mathtt{N} = \mathtt{N}_1[y\backslash s]$: the result is obvious unless:
  - $\mathtt{R}' = \Box$: this time, the fact that r is a redex forces $r = \mathtt{R}'_1\langle y\rangle[y\backslash s]$. Even if we admit that $s = \mathtt{L}\langle\mathtt{v}\rangle$, we may still conclude because $x \neq y$ (by the hypothesis $x \in \mathtt{fv}(t)$), hence $\mathtt{N}_1\langle x\rangle \neq \mathtt{R}'_1\langle y\rangle$ by item 2.
  - $\mathtt{R}' = \mathtt{R}'_1[y\backslash s]$: immediate by induction hypothesis on $\mathtt{N}_1$.
  - $\mathtt{R}' = \mathtt{R}'_1\langle y\rangle[y\backslash\mathtt{R}'_2]$: even if $\mathtt{R}'_2\langle r\rangle = s$, we may still conclude because, again, $x \neq y$ implies $\mathtt{N}_1\langle x\rangle \neq \mathtt{R}'_1\langle y\rangle$ by item 2.
- $\mathtt{N} = \mathtt{N}_1\langle y\rangle[y\backslash\mathtt{N}_2]$: again, the result is obvious unless:
  - $\mathtt{R}' = \Box$: the fact that r is a redex implies $r = \mathtt{R}'_1\langle y\rangle[y\backslash\mathtt{L}\langle\mathtt{v}\rangle]$. Even assuming $\mathtt{R}'_1 = \mathtt{N}_1$, we may still conclude because $\mathtt{N}_2\langle x\rangle \neq \mathtt{L}\langle\mathtt{v}\rangle$ by item 1.
  - $\mathtt{R}' = \mathtt{R}'_1[y\backslash\mathtt{N}_2\langle x\rangle]$: since $y \in \mathtt{fv}(\mathtt{N}_1\langle y\rangle)$, we conclude because the induction hypothesis gives us $\mathtt{N}_1\langle y\rangle \neq \mathtt{R}'_1\langle r\rangle$.
  - $\mathtt{R}' = \mathtt{N}_1\langle y\rangle[y\backslash\mathtt{R}'_2]$: we conclude at once by applying the induction hypothesis to $\mathtt{N}_2$.

□

Now, the proof of Prop. 3.11 is by structural induction on $t := \mathtt{N}_1\langle r_1\rangle = \mathtt{N}_2\langle r_2\rangle$. Cases:

- **Variable or abstraction.** Impossible, since variables and abstractions are both call-by-need normal.
- **Application, *i.e.*** $t = su$**.** This case is treated exactly as in the corresponding case of the proof of Prop. 3.11.
- **Substitution, *i.e.*** $t = s[x\backslash u]$**.** Cases:
  - *Both contexts have their holes in* s *or* u. It follows from the *i.h.*.

– *One of the contexts—say* $\mathtt{N}_1$*—is empty,* i.e. $s = \mathtt{N}_3\langle x\rangle$, $u = \mathtt{L}\langle \mathtt{v}\rangle$, *and* $r_1 = \mathtt{N}_3\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$. This case is impossible. Indeed, 1) the hole of $\mathtt{N}_2$ cannot be in $\mathtt{L}\langle \mathtt{v}\rangle$, because it is call-by-need normal, and 2) it cannot be inside $\mathtt{N}_3\langle x\rangle$ because by Lemma A.4.3 $\mathtt{R}\langle x\rangle$ is call-by-need normal.

– *One of the contexts—say* $\mathtt{N}_1$*—has its hole in* u *and the other one has its hole in* s*,* i.e. $\mathtt{N}_1 = \mathtt{N}_3\langle x\rangle[x\backslash \mathtt{N}_4]$ *and* $\mathtt{N}_2 = \mathtt{N}_5[x\backslash u]$. This case is impossible, because by Lemma A.4.3 $\mathtt{N}_3\langle x\rangle$ is call-by-need normal.

Strong Call-by-Name

Let $\mathtt{S}_1\langle r_1\rangle = \mathtt{S}_2\langle r_2\rangle$ where $\mathtt{S}_1, \mathtt{S}_2$ are strong call-by-name evaluation contexts, *i.e.* $\mathtt{S}_1, \mathtt{S}_2 \in \mathcal{S}$. Consider the steps contracting these redexes, $\mathtt{S}_1\langle r_1\rangle \to_{\text{LO}} s_1$ and $\mathtt{S}_2\langle r_2\rangle \to_{\text{LO}} s_2$. By Lem. 3.10, these are leftmost-outermost steps. Since there is only one leftmost-outermost redex, we have that $\mathtt{S}_1 = \mathtt{S}_2$ and $r_1 = r_2$, as required.

A.1.2 Structural equivalence is a strong bisimulation — proof of Prop. 3.11

Proposition A.5 (Full proof of Prop. 3.14—Structural equivalence is a strong bisimulation)**.** *Let* $\mathtt{x} \in \{\mathtt{m}, \mathtt{e}\}$. *If* $t \equiv_{\mathbb{S}} t' \to_{\mathtt{x}\mathbb{S}} s$ *then there exists* s' *such that* $t \to_{\mathtt{x}\mathbb{S}} s' \equiv_{\mathbb{S}} s$.

We prove it for each strategy separately in the following sections:

1. Call-by-Name (`name`): Section A.1.2,
2. Left-to-Right Call-by-Value ($\mathtt{value}^{\mathtt{LR}}$): Section A.1.2,
3. Call-by-Need (`need`): Section A.1.2,
4. Strong Call-by-Name ($\mathtt{name}^{\mathtt{S}}$): Section A.1.2

The proof for Right-to-Left Call-by-Value ($\mathtt{value}^{\mathtt{RL}}$) is obtained as minimal variation of the proof for Left-to-Right Call-by-Value ($\mathtt{value}^{\mathtt{LR}}$), and therefore it is omitted.

Call-by-Name (`name`)

Before proving the main result, we need two auxiliary lemmas, proved by straightforward inductions on $\mathtt{H}$:

Lemma A.6. *Let* t *be a term,* $\mathtt{H}$ *be a call-by-name evaluation context not capturing any variable in* $\mathtt{fv}(t)$*, and* $x \notin \mathtt{fv}(\mathtt{R}\langle y\rangle)$. *Then* $\mathtt{R}\langle t[x\backslash s]\rangle \equiv \mathtt{R}\langle t\rangle[x\backslash s]$.

Lemma A.7. *The equivalence relation* $\equiv$ *as defined for call-by-name preserves the shape of* $\mathtt{R}\langle x\rangle$*. More precisely, if* $\mathtt{R}\langle x\rangle \equiv t$*, with* x *not captured by* $\mathtt{H}$*, then* t *is of the form* $\mathtt{R}'\langle x\rangle$*, with* x *not captured by* $\mathtt{R}'$.

Now we turn to the proof of the proposition itself. Let $\leftrightarrows$ be the symmetric closure of the union of the axioms defining $\equiv$ for call-by-name, that is of $\equiv_{\mathtt{gc}} \cup \equiv_{\mathtt{dup}} \cup \equiv_{@} \cup \equiv_{\mathtt{com}} \cup \equiv_{[\cdot]}$. Note that $\equiv$ is the reflexive–transitive closure of $\leftrightarrows$. The proof is in two parts:

(I) Prove the property holds for $\Leftrightarrow$, *i.e.* if $t \to_a s$ and $t \Leftrightarrow u$, there exists r such that $u \to_a r$ and $s \equiv r$.

(II) Prove the property holds for $\equiv$ (*i.e.* for many steps of $\Leftrightarrow$) by resorting to (I).

The proof of (II) is immediate by induction on the number of $\Leftrightarrow$ steps. The proof of (I) goes by induction on the rewriting step $\to$ (that, since $\to$ is closed by evaluation contexts, becomes a proof by induction on the evaluation context $\mathtt{H}$). In principle, we should always consider the two directions of $\Leftrightarrow$. Most of the time, however, one direction is obtained by simply reading the diagram of the other direction bottom-up, instead than top-down; these cases are simply omitted, we distinguish the two directions only when it is relevant.

1. **Base case 1: multiplicative root step** $t = \mathtt{L}\langle\lambda x.t'\rangle s' \mapsto_{\mathtt{db}} \mathtt{L}\langle t'[x\backslash s']\rangle = s$**.**

   If the $\Leftrightarrow$ step is internal to s' or internal to one of the substitutions in $\mathtt{L}$, the pattern of the $\Leftrightarrow$ redex does not overlap with the $\mapsto_{\mathtt{db}}$ step, and the proof is immediate, the two steps commute. Otherwise, we consider every possible case for $\Leftrightarrow$:

   1.1 *Garbage Collection* $\equiv_{\mathtt{gc}}$. The garbage collected substitution must be one of the substitutions in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle\mathtt{L}''[y\backslash u']\rangle$. Let $\widehat{\mathtt{L}} := \mathtt{L}'\langle\mathtt{L}''\rangle$. Then:

$$\begin{array}{ccc} \mathtt{L}\langle\lambda x.t'\rangle s' & \xrightarrow{\mathtt{db}} & \mathtt{L}\langle t'[x\backslash s']\rangle \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ \widehat{\mathtt{L}}\langle\lambda x.t'\rangle s' & \dashrightarrow^{\mathtt{db}} & \widehat{\mathtt{L}}\langle t'[x\backslash s']\rangle \end{array}$$

   1.2 *Duplication* $\equiv_{\mathtt{dup}}$. The duplicated substitution must be one of the substitutions in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle\mathtt{L}''[y\backslash u']\rangle$. Then:

$$\begin{array}{ccc} \mathtt{L}'\langle\mathtt{L}''\langle\lambda x.t'\rangle[y\backslash u']\rangle s' & \xrightarrow{\mathtt{db}} & t_1 \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ t_2 & \overset{\mathtt{db}}{\dashrightarrow} & t_3 \end{array}$$

   where

$$\begin{aligned} t_1 &:= \mathtt{L}'\langle\mathtt{L}''\langle t'[x\backslash s']\rangle[y\backslash u']\rangle, \\ t_2 &:= \mathtt{L}'\langle\ (\mathtt{L}''\langle\lambda x.t'\rangle)_{[z]_y}[y\backslash u'][z\backslash u']\ \rangle s', \\ t_3 &:= \mathtt{L}'\langle\ (\mathtt{L}''\langle t'[x\backslash s']\rangle)_{[z]_y}[y\backslash u'][z\backslash u']\ \rangle. \end{aligned}$$

   1.3 *Commutation with application* $\equiv_{@}$. Here $\equiv_{@}$ can only be applied in one direction. The diagram is:

$$\begin{array}{ccc}
\mathtt{L}\langle\lambda y.t'\rangle[x\backslash q']s'[x\backslash q'] & \equiv_{@} & t_1 \\
\Big\downarrow{\scriptstyle \mathtt{db}} & & \vdots{\scriptstyle \mathtt{db}} \\
t_4 & & t_2 \\
 & & \equiv_{\mathtt{dup}} \\
=_{\alpha} & & t_3 \\
 & & \equiv^{*}_{\mathtt{com}} \\
t_5 & \equiv_{[\cdot]} & t_6
\end{array}$$

where

$$\begin{aligned}
t_1 &:= (\mathtt{L}\langle\lambda y.t'\rangle s')[x\backslash q'],\\
t_2 &:= (\mathtt{L}\langle t'[y\backslash s']\rangle)[x\backslash q'],\\
t_3 &:= (\mathtt{L}\langle t'[y\backslash s'\{x/y\}]\rangle)[x\backslash q'][y\backslash q'],\\
t_4 &:= \mathtt{L}\langle t'[y\backslash s'[x\backslash q']]\rangle[x\backslash q'],\\
t_5 &:= (\mathtt{L}\langle t'[y\backslash s'\{x/y\}[y\backslash q']]\rangle)[x\backslash q'],\\
t_6 &:= (\mathtt{L}\langle t'[y\backslash s'\{x/y\}][y\backslash q']\rangle)[x\backslash q'].
\end{aligned}$$

1.4 *Commutation of independent substitutions* $\equiv_{\mathtt{com}}$. The substitutions that are commuted by the $\equiv_{\mathtt{com}}$ rule must be both in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle\mathtt{L}''[y\backslash u'][z\backslash r']\rangle$ with $z \notin \mathtt{fv}(u')$. Let $\widehat{\mathtt{L}} = \mathtt{L}'\langle\mathtt{L}''[z\backslash r'][y\backslash u']\rangle$. Then:

$$\begin{array}{ccc}
\mathtt{L}\langle\lambda x.t'\rangle s' & \xrightarrow{\mathtt{db}} & \mathtt{L}\langle t'[x\backslash s']\rangle \\
\equiv_{\mathtt{com}} & & \equiv_{\mathtt{com}} \\
\widehat{\mathtt{L}}\langle\lambda x.t'\rangle s' & \dashrightarrow^{\mathtt{db}} & \widehat{\mathtt{L}}\langle t'[x\backslash s']\rangle
\end{array}$$

1.5 *Composition of substitutions* $\equiv_{[\cdot]}$. The substitutions that appear in the left-hand side of the $\equiv_{[\cdot]}$ rule must both be in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle\mathtt{L}''[y\backslash u'][z\backslash r']\rangle$ with $z \notin \mathtt{fv}(\mathtt{L}''\langle\lambda x.t'\rangle)$. Let $\widehat{\mathtt{L}} = \mathtt{L}'\langle\mathtt{L}''[y\backslash u'[z\backslash r']]\rangle$. Exactly as in the previous case:

$$\begin{array}{ccc}
\mathtt{L}\langle\lambda x.t'\rangle s' & \xrightarrow{\mathtt{db}} & \mathtt{L}\langle t'[x\backslash s']\rangle \\
\equiv_{[\cdot]} & & \equiv_{[\cdot]} \\
\widehat{\mathtt{L}}\langle\lambda x.t'\rangle s' & \dashrightarrow^{\mathtt{db}} & \widehat{\mathtt{L}}\langle t'[x\backslash s']\rangle
\end{array}$$

2. **Base case 2: exponential root step** $t = \mathtt{R}'\langle x\rangle[x\backslash t'] \mapsto_{\mathtt{ls}} \mathtt{R}'\langle t'\rangle[x\backslash t'] = s$**.** If the $\Leftrightarrow$ step is internal to t', the proof is immediate, since there is no overlap with the pattern of the $\mapsto_{\mathtt{ls}}$ redex. Similarly, if the $\Leftrightarrow$ step is internal to $\mathtt{R}\langle x\rangle$, the proof is straightforward by resorting to Lem. A.7.

   Now we proceed by case analysis on the $\Leftrightarrow$ step:

   2.1 *Garbage collection* $\equiv_{\mathtt{gc}}$. Note that $\equiv_{\mathtt{gc}}$ cannot remove $[x\backslash t']$, because by hypothesis x does occur in its scope. If the removed substitution belongs to $\mathtt{R}'$, *i.e.* $\mathtt{R}' = \mathtt{H}''\langle\mathtt{H}'''[y\backslash s']\rangle$. Let $\widehat{\mathtt{R}'} := \mathtt{H}''\langle\mathtt{H}'''\rangle$. Then:

$$\begin{array}{ccc} \mathtt{R}'\langle x\rangle[x\backslash t'] & \xrightarrow{\mathtt{ls}} & \mathtt{R}'\langle t'\rangle[x\backslash t'] \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ \widehat{\mathtt{R}}'\langle x\rangle[x\backslash t'] & \dashrightarrow^{\mathtt{ls}} & \widehat{\mathtt{R}}'\langle t'\rangle[x\backslash t'] \end{array}$$

If $\equiv_{\mathtt{gc}}$ adds a substitution as topmost constructor the diagram is analogous.

2.2 *Duplication* $\equiv_{\mathtt{dup}}$. Two sub-cases:

2.2.1 *The equivalence* $\equiv_{\mathtt{dup}}$ *acts on a substitution internal to* $\mathtt{R}'$. This case goes as for Garbage collection.

2.2.2 *The equivalence* $\equiv_{\mathtt{dup}}$ *acts on* $[x\backslash t']$. There are two further sub-cases:

- *The substituted occurrence is renamed by* $\equiv_{\mathtt{dup}}$:

$$\begin{array}{ccc} \mathtt{R}'\langle x\rangle[x\backslash t'] & \xrightarrow{\mathtt{ls}} & \mathtt{R}'\langle t'\rangle[x\backslash t'] \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ \mathtt{R}'_{[y]_x}\langle y\rangle[x\backslash t'][y\backslash t'] & \dashrightarrow^{\mathtt{ls}} & t_1 \end{array}$$

  where $t_1 := \mathtt{R}'_{[y]_x}\langle t'\rangle[x\backslash t'][y\backslash t']$ and $\mathtt{R}'_{[y]_x}$ is the context obtained from $\mathtt{R}'$ by renaming some (possibly none) occurrences of x as y.

- *The substituted occurrence is not renamed by* $\equiv_{\mathtt{dup}}$. Essentially as in the previous case:

$$\begin{array}{ccc} \mathtt{R}'\langle x\rangle[x\backslash t'] & \xrightarrow{\mathtt{ls}} & \mathtt{R}'\langle t'\rangle[x\backslash t'] \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ \mathtt{R}'_{[y]_x}\langle x\rangle[x\backslash t'][y\backslash t'] & \dashrightarrow^{\mathtt{ls}} & t_1 \end{array}$$

  where $t_1 := \mathtt{R}'_{[y]_x}\langle t'\rangle[x\backslash t'][y\backslash t']$.

2.3 *Commutation with application* $\equiv_{@}$. Two sub-cases:

2.3.1 *The equivalence* $\equiv_{@}$ *acts on a substitution internal to* $\mathtt{R}'$. This case goes as for Garbage collection.

2.3.2 *The equivalence* $\equiv_{@}$ *acts on* $[x\backslash t']$. It must be the case that $\mathtt{R}'$ is of the form $\mathtt{H}''s'$. Then:

$$\begin{array}{ccc} (\mathtt{H}''\langle x\rangle s')[x\backslash t'] & \xrightarrow{\mathtt{ls}} & t_1 \\ \equiv_{@} & & \equiv_{@} \\ t_2 & \dashrightarrow^{\mathtt{ls}} & t_3 \end{array}$$

where

$$\begin{aligned} t_1 &:= (\mathtt{H}''\langle t'\rangle s')[x\backslash t'], \\ t_2 &:= \mathtt{H}''\langle x\rangle[x\backslash t']s'[x\backslash t'], \\ t_3 &:= \mathtt{H}''\langle t'\rangle[x\backslash t']s'[x\backslash t']. \end{aligned}$$

2.4 *Commutation of independent substitutions* $\equiv_{\mathtt{com}}$. Two sub-cases:

2.4.1 *The equivalence* $\equiv_{\mathtt{com}}$ *acts on two substitutions internal to* $\mathtt{R}'$. This case goes as for Garbage collection.

2.4.2 *The equivalence* $\equiv_{\mathtt{com}}$ *acts on* $[x\backslash t']$. It must be the case that $\mathtt{R}'$ is of the form $\mathtt{H}''$. Then:

$$\begin{array}{ccc} \mathtt{H}''\langle x\rangle[y\backslash s'][x\backslash t'] & \xrightarrow{\mathtt{ls}} & \mathtt{H}''\langle t'\rangle[y\backslash s'][x\backslash t'] \\ \equiv_{\mathtt{com}} & & \equiv_{\mathtt{com}} \\ \mathtt{H}''\langle x\rangle[x\backslash t'][y\backslash s'] & \dashrightarrow^{\mathtt{ls}} & \mathtt{H}''\langle t'\rangle[x\backslash t'][y\backslash s'] \end{array}$$

2.5 *Composition of substitutions* $\equiv_{[\cdot]}$. Two sub-cases:

2.5.1 *The equivalence* $\equiv_{[\cdot]}$ *acts on two substitutions internal to* $\mathtt{R}'$. This case goes as for Garbage collection.

2.5.2 *The equivalence* $\equiv_{[\cdot]}$ *acts on* $[x\backslash t']$. Note that the equivalence $\equiv_{[\cdot]}$ cannot be applied from left to right to $[x\backslash t']$, because $\mathtt{R}'\langle x\rangle$ must be of the form $\mathtt{H}''\langle x\rangle[y\backslash s']$ with $x \notin \mathtt{fv}(\mathtt{H}''\langle x\rangle)$, which is clearly not possible. It can be applied from right to left. The diagram is:

$$\begin{array}{ccc} \mathtt{R}'\langle x\rangle[x\backslash t'[y\backslash s]] & \equiv_{@} & t_1 \\ \downarrow_{\mathtt{ls}} & & \vdots_{\mathtt{ls}} \\ t_4 & & t_2 \\ \equiv \text{ by Lem. A.6} & & \equiv_{\mathtt{dup}} \\ t_5 & & t_3 \\ \equiv_{[\cdot]} & & \equiv_{\mathtt{com}} \\ t_6 & =_{\alpha} & t_7 \end{array}$$

where

$$\begin{aligned} t_1 &:= \mathtt{R}'\langle x\rangle[x\backslash t'][y\backslash s], \\ t_2 &:= \mathtt{R}'\langle t'\rangle[x\backslash t'][y\backslash s], \\ t_3 &:= \mathtt{R}'\langle t'\{y/z\}\rangle[x\backslash t'][z\backslash s][y\backslash s], \\ t_4 &:= \mathtt{R}'\langle t'[y\backslash s]\rangle[x\backslash t'[y\backslash s]], \\ t_5 &:= \mathtt{R}'\langle t'\rangle[y\backslash s][x\backslash t'[y\backslash s]], \\ t_6 &:= \mathtt{R}'\langle t'\rangle[y\backslash s][x\backslash t'][y\backslash s], \\ t_7 &:= \mathtt{R}'\langle t'\{y/z\}\rangle[z\backslash s][x\backslash t'][y\backslash s]. \end{aligned}$$

3. **Inductive case 1: left of an application** $\mathtt{H} = \mathtt{R}'q$**.** The situation is:

$$t = t'q \to_a s'q = s$$

for terms t', s' such that either $t' \to_{\mathtt{m}} s'$ or $t' \to_{\mathtt{e}} s'$. Two sub-cases:

3.1 *The* $t \Leftrightarrow u$ *step is internal to* t'. The proof simply uses the *i.h.* applied to the (strictly smaller) evaluation context of the step $t' \to_a s'$.

3.2 *The* $t \Leftrightarrow u$ *step involves the topmost application.* The $\Leftrightarrow$ step can only be a commutation with the root application. Moreover, for $t'q$ to match with the right-hand

side of the $\equiv_{@}$ rule, t' must have the form $u'[x\backslash r']$ and q the form $q'[x\backslash r']$, so that the $\Leftrightarrow$ is:

$$u = (u'q')[x\backslash r'] \equiv_{@} u'[x\backslash r']q'[x\backslash r'] = t$$

Three sub-cases:

3.2.1 *The rewriting step is internal to* u'. Then the two steps trivially commute. Let $a \in \{\mathtt{db}, \mathtt{ls}\}$:

$$\begin{array}{ccc} u'[x\backslash r']q'[x\backslash r'] & \xrightarrow{a} & u''[x\backslash r']q'[x\backslash r'] \\ \equiv_{@} & & \equiv_{@} \\ (u'q')[x\backslash r'] & \dashrightarrow^{a} & (u''q')[x\backslash r'] \end{array}$$

3.2.2 `db`*-step not internal to* u'. Exactly as the multiplicative root case 1.3 (read in the other direction).

3.3 `ls`*-step not internal to* u'. Not possible: the topmost constructor is an application, consequently any $\to_{\mathtt{e}}$ has to take place in u'.

4. **Inductive case 2: left of a substitution** $\mathtt{H} = \mathtt{R}'[x\backslash q]$**.** The situation is:

$$t = t'[x\backslash q] \to s'[x\backslash q] = s$$

with $t' = \mathtt{R}'\langle t''\rangle$. If the $\Leftrightarrow$ step is internal to $\mathtt{R}'\langle t'\rangle$, the proof we conclude using the *i.h.*. Otherwise:

4.1 *Garbage Collection* $\equiv_{\mathtt{gc}}$. If the garbage collected substitution is $[x\backslash q]$ then:

$$\begin{array}{ccc} t'[x\backslash q] & \longrightarrow & s'[x\backslash q] \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ t' & \dashrightarrow & s' \end{array}$$

If the substitution is introduced out of the blue, *i.e.* $t'[x\backslash q] \equiv_{\mathtt{gc}} t'[x\backslash q][y\backslash q']$ or $t'[x\backslash q] \equiv_{\mathtt{gc}} t'[y\backslash q'][x\backslash q]$ the diagram is analogous.

4.2 *Duplication* $\equiv_{\mathtt{dup}}$. If the duplicated substitution is $[x\backslash q]$ then:

$$\begin{array}{ccc} t'[x\backslash q] & \longrightarrow & s'[x\backslash q] \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ t'_{[y]_x}[x\backslash q][y\backslash q] & \dashrightarrow & s'_{[y]_x}[x\backslash q] \end{array}$$

If duplication is applied in the other direction, *i.e.* $t' = t''[y\backslash q]$ and

$$t'[x\backslash q] = t''[y\backslash q][x\backslash q] \equiv_{\mathtt{dup}} t''\{y/x\}[x\backslash q] = t'[x\backslash q]$$

the interesting case is when $t'' = \mathtt{H}''\langle y\rangle$ and the step is exponential:

$$\begin{array}{ccc} \mathtt{H}''\langle y\rangle[y\backslash q][x\backslash q] & \xrightarrow{\mathtt{ls}} & \mathtt{H}''\langle q\rangle[y\backslash q][x\backslash q] \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ \mathtt{H}''\langle x\rangle\{y/x\}[x\backslash q] & \dashrightarrow^{\mathtt{ls}} & \mathtt{H}''\langle q\rangle\{y/x\}[x\backslash q] \end{array}$$

If t' is $\mathtt{H}''\langle x\rangle$ it is an already treated base case and if t' has another form the rewriting step does not interact with the duplication, and so they simply commute.

4.3 *Commutation with application* $\equiv_{@}$. Then $t' = t''s''$. Three sub-cases:

4.3.1 *The* $\to$ *step is internal to* t''. Then:

$$\begin{array}{ccc} (t''s'')[x\backslash q] & \longrightarrow & (t'''s'')[x\backslash q] \\ \equiv_{@} & & \equiv_{@} \\ t''[x\backslash q]s''[x\backslash q] & \dashrightarrow & t'''[x\backslash q]s''[x\backslash q] \end{array}$$

4.3.2 *The* $\to$ *step is a multiplicative step.* If $t'' = \mathtt{L}\langle\lambda y.t'''\rangle$ then it goes like the diagram of the multiplicative root case 1.3 (read in the other direction).

4.3.3 *The* $\to$ *step is an exponential step.* Then it must be $[x\backslash q]$ that substitutes on the head variable, but this case has already been treated as a base case (case 2.3).

4.4 *Commutation of independent substitutions* $\equiv_{\mathtt{com}}$. It must be $t' = t''[y\backslash q']$ with $x \notin \mathtt{fv}(q')$, so that $t''[y\backslash q'][x\backslash q] \equiv_{\mathtt{com}} t''[x\backslash q][y\backslash q']$. Three sub-cases:

4.4.1 *Reduction takes place in* t''. Then reduction and the equivalence simply commute, as in case 4.3.1.

4.4.2 *Exponential steps involving* $[x\backslash q]$. This is an already treated base case (case 2.4.2).

4.4.3 *Exponential step involving* $[y\backslash q']$. This case is solved reading bottom-up the diagram of case 2.4.2.

4.5 *Composition of substitutions* $\equiv_{[\cdot]}$. It must be $t' = t''[y\backslash q']$ with $x \notin \mathtt{fv}(t'')$, so that $t''[y\backslash q'][x\backslash q] \equiv_{[\cdot]} t''[x\backslash q[y\backslash q']]$. Three sub-cases:

4.5.1 *Reduction takes place in* t''. Then reduction and the equivalence simply commute, as in case 4.3.1.

4.5.2 *Exponential steps involving* $[x\backslash q]$. This case is solved reading bottom-up the diagram of case 2.5.2.

4.5.3 *Exponential step involving* $[y\backslash q']$. Impossible, because by hypothesis $x \notin \mathtt{fv}(t'')$.

Left-to-Right Call-by-Value ($\mathtt{value}^{\mathtt{LR}}$**)**

We follow the structure of the proof in for call-by-name.

Before proving the main result, we need the following auxiliary lemmas, proved by straightforward inductions on the contexts. Lem. A.8.2 is the adaptation of Lem. A.7 already stated for call-by-name:

Lemma A.8. *The equivalence relation* $\equiv$ *preserves the "shapes" of* $\mathtt{L}\langle\mathtt{v}\rangle$ *and* $\mathtt{R}\langle x\rangle$. *Formally:*

1. *If* $\mathtt{L}\langle\mathtt{v}\rangle \equiv t$, *then* t *is of the form* $\mathtt{L}'\langle\mathtt{v}'\rangle$.
2. *If* $\mathtt{R}\langle x\rangle \equiv t$, *with* x *not bound by* $\mathtt{V}$, *then* t *is of the form* $\mathtt{R}'\langle x\rangle$, *with* x *not bound by* $\mathtt{R}'$.

Lemma A.9. $\mathtt{L}\langle t[x\backslash s]\rangle \equiv \mathtt{L}\langle t[x\backslash\mathtt{L}\langle s\rangle]\rangle$

Proof. By induction on $\mathtt{L}$. The base case is trivial. For $\mathtt{L} = \mathtt{L}'\Box[y\backslash u]$, by *i.h.* we have:

$$\mathtt{L}'\langle t[x\backslash s]\rangle[y\backslash u] \equiv \mathtt{L}'\langle t[x\backslash \mathtt{L}'\langle s\rangle]\rangle[y\backslash u]$$

Let $(\mathtt{L}'\langle s\rangle)_{[z]_y}$ be the result of replacing *all* occurrences of y by z in $\mathtt{L}'\langle s\rangle$. Then:

$$\begin{array}{ll} & \mathtt{L}'\langle t[x\backslash \mathtt{L}'\langle s\rangle]\rangle[y\backslash u] \\ \equiv_{\mathtt{dup}} & \mathtt{L}'\langle t[x\backslash (\mathtt{L}'\langle s\rangle)_{[z]_y}]\rangle[y\backslash u][z\backslash u] \\ \equiv^{*}_{\mathtt{com}} & \mathtt{L}'\langle t[x\backslash (\mathtt{L}'\langle s\rangle)_{[z]_y}][z\backslash u]\rangle[y\backslash u] \\ \equiv_{[\cdot]} & \mathtt{L}'\langle t[x\backslash (\mathtt{L}'\langle s\rangle)_{[z]_y}[z\backslash u]]\rangle[y\backslash u] \\ =_{\alpha} & \mathtt{L}'\langle t[x\backslash \mathtt{L}'\langle s\rangle[y\backslash u]]\rangle[y\backslash u] \end{array}$$

□

Now we prove the strong bisimulation property, by induction on the derivation of the reduction step.

1. **Base case 1: multiplicative root step** $t = \mathtt{L}\langle \lambda x.t'\rangle \mathtt{L}'\langle \mathtt{v}\rangle \mapsto_{\mathtt{dblsv}} s = \mathtt{L}\langle t'[x\backslash \mathtt{L}'\langle \mathtt{v}\rangle]\rangle$**.** The nontrivial cases are when the $\Leftrightarrow$ step overlaps the pattern of the $\mathtt{dbv}$-redex. Note that by Lem. A.8.1, if the $\Leftrightarrow$ is internal to $\mathtt{L}'\langle \mathtt{v}\rangle$, the proof is direct, since the $\mathtt{dbv}$-redex is preserved. More precisely, if $\mathtt{L}'\langle \mathtt{v}\rangle \Leftrightarrow \mathtt{L}''\langle \mathtt{v}'\rangle$, we have:

$$\begin{array}{ccc} \mathtt{L}\langle \lambda x.t'\rangle \mathtt{L}'\langle \mathtt{v}\rangle & \xrightarrow{\mathtt{dbv}} & \mathtt{L}\langle t'[x\backslash \mathtt{L}'\langle \mathtt{v}\rangle]\rangle \\ \Leftrightarrow & & \Leftrightarrow \\ \mathtt{L}\langle \lambda x.t'\rangle \mathtt{L}''\langle \mathtt{v}'\rangle & \dashrightarrow^{\mathtt{dbv}} & \mathtt{L}\langle t'[x\backslash \mathtt{L}''\langle \mathtt{v}'\rangle]\rangle \end{array}$$

   Consider the remaining possibilities for $\Leftrightarrow$:

   1.1 *Garbage collection* $\equiv_{\mathtt{gc}}$. The garbage collected substitution must be in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}_1\langle \mathtt{L}_2[y\backslash \mathtt{L}''\langle \mathtt{v}'\rangle]\rangle$ with $y \notin \mathtt{fv}(\mathtt{L}_2\langle \lambda x.t'\rangle)$. Let $\widehat{\mathtt{L}} := \mathtt{L}_1\langle \mathtt{L}_2\rangle$. Then:

$$\begin{array}{ccc} \mathtt{L}\langle \lambda x.t'\rangle \mathtt{L}'\langle \mathtt{v}\rangle & \xrightarrow{\mathtt{dbv}} & \mathtt{L}\langle t'[x\backslash \mathtt{L}'\langle \mathtt{v}\rangle]\rangle \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ \widehat{\mathtt{L}}\langle \lambda x.t'\rangle \mathtt{L}'\langle \mathtt{v}\rangle & \dashrightarrow^{\mathtt{dbv}} & \widehat{\mathtt{L}}\langle t'[x\backslash \mathtt{L}'\langle \mathtt{v}\rangle]\rangle \end{array}$$

   1.2 *Duplication* $\equiv_{\mathtt{dup}}$. The duplicated substitution must be in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}_1\langle \mathtt{L}_2[y\backslash s']\rangle$. Let $\widehat{\mathtt{L}} := \mathtt{L}_1\langle \Box[y\backslash s'][z\backslash s']\rangle$. Then:

$$\begin{array}{ccc} \mathtt{L}\langle \lambda x.t'\rangle \mathtt{L}'\langle \mathtt{v}\rangle & \xrightarrow{\mathtt{dbv}} & \mathtt{L}\langle t'[x\backslash \mathtt{L}'\langle \mathtt{v}\rangle]\rangle \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ \widehat{\mathtt{L}}\langle (\mathtt{L}_2\langle \lambda x.t'\rangle)_{[z]_y}\rangle \mathtt{L}'\langle \mathtt{v}\rangle & \dashrightarrow^{\mathtt{dbv}} & t_1 \end{array}$$

   where $t_1 := \widehat{\mathtt{L}}\langle (\mathtt{L}_2\langle t'[x\backslash \mathtt{L}'\langle \mathtt{v}\rangle]\rangle)_{[z]_y}\rangle$.

1.3 *Commutation with application* $\equiv_{@}$. The axiom can be applied only in one direction and there must be the same explicit substitution $[y\backslash q]$ as topmost constructor of each of the two sides of the application. The diagram is:

$$\begin{array}{ccc} \mathtt{L}\langle\lambda x.t'\rangle[x\backslash q]\,\mathtt{L}'\langle\mathtt{v}\rangle[y\backslash q] & \xrightarrow{\ \mathtt{dbv}\ } & t_1 \\ \equiv_{@} & & \equiv \\ (\mathtt{L}\langle\lambda x.t'\rangle\,\mathtt{L}'\langle\mathtt{v}\rangle)[y\backslash q] & \dashrightarrow^{\mathtt{dbv}} & t_2 \end{array}$$

where

$$\begin{aligned} t_1 &:= \mathtt{L}\langle t'[x\backslash \mathtt{L}'\langle\mathtt{v}\rangle[y\backslash q]]\rangle[x\backslash q], \\ t_2 &:= \mathtt{L}\langle t'[x\backslash \mathtt{L}'\langle\mathtt{v}\rangle]\rangle[y\backslash q]. \end{aligned}$$

To prove the equivalence on the right, let $\mathtt{L}'\langle\mathtt{v}\rangle_{[z]_x}$ denote the result of replacing *all* occurrences of x by a fresh variable z in $\mathtt{L}'\langle\mathtt{v}\rangle$. The equivalence holds because:

$$\begin{array}{ll} & \mathtt{L}\langle t''[y\backslash \mathtt{L}'\langle\mathtt{v}\rangle]\rangle[x\backslash q] \\ \equiv_{\mathtt{dup}} & \mathtt{L}\langle t''[y\backslash \mathtt{L}'\langle\mathtt{v}\rangle_{[z]_x}]\rangle[x\backslash q][z\backslash q] \\ \equiv^{*}_{\mathtt{com}} & \mathtt{L}\langle t''[y\backslash \mathtt{L}'\langle\mathtt{v}\rangle_{[z]_x}][z\backslash q]\rangle[x\backslash q] \\ \equiv_{[\cdot]} & \mathtt{L}\langle t''[y\backslash \mathtt{L}'\langle\mathtt{v}\rangle_{[z]_x}[z\backslash q]]\rangle[x\backslash q] \\ =_{\alpha} & \mathtt{L}\langle t''[y\backslash \mathtt{L}'\langle\mathtt{v}\rangle[x\backslash q]]\rangle[x\backslash q] \end{array}$$

1.4 *Commutation of independent substitutions* $\equiv_{\mathtt{com}}$. The commutation of substitutions must be in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}_1\langle\mathtt{L}_2[y\backslash s'][z\backslash u']\rangle$ with $z \notin \mathtt{fv}(s')$. Let $\widehat{\mathtt{L}} := \mathtt{L}_1\langle\mathtt{L}_2[z\backslash u'][y\backslash s']\rangle$. Then:

$$\begin{array}{ccc} \mathtt{L}\langle\lambda x.t'\rangle\,\mathtt{L}'\langle\mathtt{v}\rangle & \xrightarrow{\ \mathtt{dbv}\ } & \mathtt{L}\langle t'[x\backslash \mathtt{L}'\langle\mathtt{v}\rangle]\rangle \\ \equiv_{\mathtt{com}} & & \equiv_{\mathtt{com}} \\ \widehat{\mathtt{L}}\langle\lambda x.t'\rangle\,\mathtt{L}'\langle\mathtt{v}\rangle & \dashrightarrow^{\mathtt{dbv}} & \widehat{\mathtt{L}}\langle t'[x\backslash \mathtt{L}'\langle\mathtt{v}\rangle]\rangle \end{array}$$

1.5 *Composition of substitutions* $\equiv_{[\cdot]}$. The composition of substitutions must be in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}_1\langle\mathtt{L}_2[y\backslash s'][z\backslash u']\rangle$ with $z \notin \mathtt{fv}(\mathtt{L}_2\langle\lambda x.t'\rangle)$. Let $\widehat{\mathtt{L}} := \mathtt{L}_1\langle\mathtt{L}_2[y\backslash s'[z\backslash u']]\rangle$. As in the previous case:

$$\begin{array}{ccc} \mathtt{L}\langle\lambda x.t'\rangle\,\mathtt{L}'\langle\mathtt{v}\rangle & \xrightarrow{\ \mathtt{dbv}\ } & \mathtt{L}\langle t'[x\backslash \mathtt{L}'\langle\mathtt{v}\rangle]\rangle \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ \widehat{\mathtt{L}}\langle\lambda x.t'\rangle\,\mathtt{L}'\langle\mathtt{v}\rangle & \dashrightarrow^{\mathtt{dbv}} & \widehat{\mathtt{L}}\langle t'[x\backslash \mathtt{L}'\langle\mathtt{v}\rangle]\rangle \end{array}$$

2. **Base case 2: exponential root step** $t = \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}\langle\mathtt{v}\rangle] \mapsto_{\mathtt{lslsv}} s = \mathtt{L}\langle\mathtt{R}\langle\mathtt{v}\rangle[x\backslash\mathtt{v}]\rangle$**.** Consider first the case when the $\Leftrightarrow$-redex is internal to $\mathtt{R}\langle x\rangle$. By Lem. A.8. we know $\Leftrightarrow$ preserves the shape of $\mathtt{R}\langle x\rangle$, *i.e.* $\mathtt{R}\langle x\rangle \Leftrightarrow \widehat{\mathtt{V}}\langle x\rangle$. Then:

$$\begin{array}{ccc} \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}\langle\mathtt{v}\rangle] & \xrightarrow{\ \mathtt{lsv}\ } & \mathtt{L}\langle\mathtt{R}\langle\mathtt{v}\rangle[x\backslash\mathtt{v}]\rangle \\ \Leftrightarrow & & \equiv \\ \widehat{\mathtt{V}}\langle x\rangle[x\backslash \mathtt{L}\langle\mathtt{v}\rangle] & \dashrightarrow^{\mathtt{lsv}} & \mathtt{L}\langle\widehat{\mathtt{V}}\langle\mathtt{v}\rangle[x\backslash\mathtt{v}]\rangle \end{array}$$

If the $\Leftrightarrow$-redex is internal to one of the substitutions in $\mathtt{L}$, the proof is straightforward. Note that the $\Leftrightarrow$-redex has always a substitution at the root. The remaining possibilities are such that substitution is in $\mathtt{L}$, or that it is precisely $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$. Axiom by axiom:

2.1 *Garbage collection* $\equiv_{\mathtt{gc}}$. If the garbage collected substitution is in $\mathtt{L}$, let $\widehat{\mathtt{L}}$ be $\mathtt{L}$ without such substitution. Then:

$$\begin{array}{ccc} \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\quad\mathtt{lsv}\quad} & \mathtt{L}\langle \mathtt{R}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ \mathtt{R}\langle x\rangle[x\backslash \widehat{\mathtt{L}}\langle \mathtt{v}\rangle] & \dashrightarrow^{\mathtt{lsv}} & \widehat{\mathtt{L}}\langle \mathtt{R}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle \end{array}$$

The garbage collected substitution cannot be $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$, since this would imply $x \notin \mathtt{fv}(\mathtt{R}\langle x\rangle)$, which is a contradiction.

2.2 *Duplication* $\equiv_{\mathtt{dup}}$. If the duplicated substitution is in $\mathtt{L}$, then $\mathtt{L}$ is of the form $\mathtt{L}_1\langle \mathtt{L}_2[y\backslash t']\rangle$. Let $\widehat{\mathtt{L}} = \mathtt{L}_1\langle [y\backslash t'][z\backslash t']\rangle$. Then:

$$\begin{array}{ccc} \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\quad\mathtt{lsv}\quad} & \mathtt{L}\langle \mathtt{R}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ t_1 & \dashrightarrow^{\mathtt{lsv}} & t_2 \end{array}$$

where

$$\begin{aligned} t_1 &:= \mathtt{R}\langle x\rangle[x\backslash \widehat{\mathtt{L}}\langle \mathtt{L}_{2[z]_y}\langle \mathtt{v}_{[z]_y}\rangle\rangle], \\ t_2 &:= \widehat{\mathtt{L}}\langle \mathtt{L}_{2[z]_y}\langle \mathtt{R}\langle \mathtt{v}_{[z]_y}\rangle[x\backslash \mathtt{v}_{[z]_y}]\rangle\rangle. \end{aligned}$$

If the duplicated substitution is $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$, there are two possibilities, depending on whether the occurrence of x substituted by the $\mapsto_{\mathtt{lslsv}}$ step is replaced by the fresh variable y, or left untouched. If it is not replaced:

$$\begin{array}{ccc} \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\quad\mathtt{lsv}\quad} & \mathtt{L}\langle \mathtt{R}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle \\ & & \equiv_{\mathtt{dup}} \\ \equiv_{\mathtt{dup}} & & t_2 \\ & & \equiv \text{ (Lem. A.9)} \\ t_4 & \dashrightarrow^{\mathtt{lsv}} & t_3 \end{array}$$

where

$$\begin{aligned} t_2 &:= \mathtt{L}\langle (\mathtt{R}\langle \mathtt{v}\rangle)_{[y]_x}[x\backslash \mathtt{v}][y\backslash \mathtt{v}]\rangle, \\ t_3 &:= \mathtt{L}\langle (\mathtt{R}\langle \mathtt{v}\rangle)_{[y]_x}[x\backslash \mathtt{v}][y\backslash \mathtt{L}\langle \mathtt{v}\rangle]\rangle, \\ t_4 &:= (\mathtt{R}\langle x\rangle)_{[y]_x}[x\backslash \mathtt{L}\langle \mathtt{v}\rangle][y\backslash \mathtt{L}\langle \mathtt{v}\rangle]. \end{aligned}$$

If the occurrence of x substituted by the $\mapsto_{\mathtt{lslsv}}$ step is replaced by the fresh variable y, the situation is essentially analogous.

2.3 *Commutation with application* $\equiv_{@}$. The only possibility is that the substitution $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$ is commuted with the outermost application in $\mathtt{R}\langle x\rangle$. Two cases:

2.3.1 *The substitution acts on the left of the application,* i.e. $\mathtt{V} = \mathtt{R}'t'$.

$$\begin{array}{ccc} (\mathtt{R}'\langle x\rangle\, t')[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\ \mathtt{lsv}\ } & t_1 \\ & & \equiv^{*}_{@} \\ \equiv_{@} & & t_2 \\ & & \equiv^{*}_{[\cdot]} \\ t_3 & \xrightarrow{\ \mathtt{lsv}\ } & t_4 \end{array}$$

where

$$\begin{aligned} t_1 &:= \mathtt{L}\langle (\mathtt{R}'\langle \mathtt{v}\rangle\, t')[x\backslash \mathtt{v}]\rangle, \\ t_2 &:= \mathtt{L}\langle \mathtt{R}'\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle \mathtt{L}\langle t'[x\backslash \mathtt{v}]\rangle, \\ t_3 &:= \mathtt{R}'\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]t'[x\backslash \mathtt{L}\langle \mathtt{v}\rangle], \\ t_4 &:= \mathtt{L}\langle \mathtt{R}'\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle t'[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]. \end{aligned}$$

2.3.2 *The substitution acts on the right of the application,* i.e. $\mathtt{V} = \mathtt{L}'\langle \mathtt{v}'\rangle\mathtt{R}'$. Similar to the previous case:

$$\begin{array}{ccc} & & t_2 \\ & & \\ (\mathtt{L}'\langle \mathtt{v}'\rangle\, \mathtt{R}'\langle x\rangle)[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\ \mathtt{lsv}\ } & t_1 \\ & & \equiv^{*}_{@} \\ \equiv_{@} & & \\ & & \equiv^{*}_{[\cdot]} \\ t_3 & \xrightarrow{\ \mathtt{lsv}\ } & t_4 \end{array}$$

where

$$\begin{aligned} t_1 &:= \mathtt{L}\langle (\mathtt{L}'\langle \mathtt{v}'\rangle\, \mathtt{R}'\langle \mathtt{v}\rangle)[x\backslash \mathtt{v}]\rangle, \\ t_2 &:= \mathtt{L}\langle \mathtt{L}'\langle \mathtt{v}'\rangle[x\backslash \mathtt{v}]\rangle \mathtt{L}\langle \mathtt{R}'\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle, \\ t_3 &:= \mathtt{L}'\langle \mathtt{v}'\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]\mathtt{R}'\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle], \\ t_4 &:= \mathtt{L}'\langle \mathtt{v}'\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]\mathtt{L}\langle \mathtt{R}'\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle. \end{aligned}$$

2.4 *Commutation of independent substitutions* $\equiv_{\mathtt{com}}$. If the commuted substitutions both belong to $\mathtt{L}$, let $\widehat{\mathtt{L}}$ be the result of commuting them, and the situation is exactly as for Garbage collection.

The remaining possibility is that $\mathtt{V} = \mathtt{R}'[y\backslash t']$ and $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$ commutes with $[y\backslash t']$ (which implies $x \notin \mathtt{fv}(t')$). Then:

$$\begin{array}{ccc} \mathtt{R}'\langle x\rangle[y\backslash t'][x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\ \mathtt{lsv}\ } & \mathtt{L}\langle \mathtt{R}'\langle \mathtt{v}\rangle[y\backslash t'][x\backslash \mathtt{v}]\rangle \\ \equiv_{\mathtt{com}} & & \equiv^{*}_{\mathtt{com}} \\ \mathtt{R}'\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle][y\backslash t'] & \xrightarrow{\ \mathtt{lsv}\ } & \mathtt{L}\langle \mathtt{R}'\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle[y\backslash t'] \end{array}$$

2.5 *Composition of substitutions* $\equiv_{[\cdot]}$. If the composed substitutions both belong to $\mathtt{L}$, let $\widehat{\mathtt{L}}$ be the result of composing them, and the situation is exactly as for Garbage collection.

The remaining possibility is that $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$ is the outermost substitution composed by $\equiv_{[\cdot]}$. This is not possible if the rule is applied from left to right, since it would imply that $\mathtt{R}\langle x\rangle = \mathtt{R}'\langle x\rangle[y\backslash t']$ with $x \notin \mathtt{R}'\langle x\rangle$, which is a contradiction.

Finally, if the $\equiv_{[\cdot]}$ rule is applied from right to left, $\mathtt{L}$ is of the form $\mathtt{L}'[y\backslash t']$ and:

$$\begin{array}{ccc} \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}'\langle \mathtt{v}\rangle[y\backslash t']] & \xrightarrow{\mathtt{lsv}} & \mathtt{L}'\langle \mathtt{R}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle[y\backslash t'] \\ \equiv_{[\cdot]} & & = \\ \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}'\langle \mathtt{v}\rangle][y\backslash t'] & \dashrightarrow^{\mathtt{lsv}} & \mathtt{L}'\langle \mathtt{R}\langle x\rangle[x\backslash \mathtt{v}]\rangle[y\backslash t'] \end{array}$$

3. **Inductive case 1: left of an application** $\mathtt{V} = \mathtt{R}'q$**.** The situation is:

$$t = \mathtt{R}'\langle t'\rangle\, q \to \mathtt{R}'\langle s'\rangle\, q = s$$

If the $\Leftrightarrow$ step is internal to $\mathtt{R}'\langle t'\rangle$, the result follows by *i.h.*. The proof is also direct if $\Leftrightarrow$ is internal to q. The nontrivial case is when the $\Leftrightarrow$ step overlaps $\mathtt{R}'\langle t'\rangle$ and q. There are two possibilities. The first is trivial: $\equiv_{\mathtt{gc}}$ is used to introduce a substitution out of the blue, but this case clearly commutes with reduction.

The second is that the application is commuted with a substitution via the $\equiv_{@}$ rule (applied from right to left). There are two cases:

3.1 *The substitution comes from* t'. That is, $\mathtt{R}' = \Box$ and t' has a substitution at its root. Then t' must be a $\mapsto_{\mathtt{lslsv}}$-redex $t' = \mathtt{V}''\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$. Moreover $q = q'[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$. We have:

$$\begin{array}{ccc} \mathtt{V}''\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]\, q'[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\mathtt{lsv}} & t_1 \\ \equiv_{@} & & \equiv \\ t_2 & \dashrightarrow^{\mathtt{lsv}} & t_3 \end{array}$$

where

$$\begin{aligned} t_1 &:= \mathtt{L}\langle \mathtt{V}''\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle\, q'[x\backslash \mathtt{L}\langle \mathtt{v}\rangle], \\ t_2 &:= (\mathtt{V}''\langle x\rangle\, q')[x\backslash \mathtt{L}\langle \mathtt{v}\rangle], \\ t_3 &:= \mathtt{L}\langle (\mathtt{V}''\langle \mathtt{v}\rangle\, q')[x\backslash \mathtt{v}]\rangle. \end{aligned}$$

For the equivalence on the right note that:

$$\begin{array}{ll} & \mathtt{L}\langle \mathtt{V}''\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle\, q'[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] \\ \equiv^*_{[\cdot]} & \mathtt{L}\langle \mathtt{V}''\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle\, \mathtt{L}\langle q'[x\backslash \mathtt{v}]\rangle \\ \equiv^*_{@} & \mathtt{L}\langle \mathtt{V}''\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\, q'[x\backslash \mathtt{v}]\rangle \\ \equiv_{@} & \mathtt{L}\langle (\mathtt{V}''\langle \mathtt{v}\rangle\, q')[x\backslash \mathtt{v}]\rangle \end{array}$$

3.2 *The substitution comes from* $\mathtt{R}'$. That is: $\mathtt{R}' = \mathtt{V}''[x\backslash u']$. Moreover, $q = q'[x\backslash u']$. The proof is then straightforward:

$$\begin{array}{ccc} \mathtt{V}''\langle t'\rangle[x\backslash u']\, q'[x\backslash u'] & \longrightarrow & t_1 \\ \equiv_{@} & & \equiv_{@} \\ t_2 & \dashrightarrow & t_3 \end{array}$$

where

$$\begin{aligned} t_1 &:= \mathtt{V}''\langle s'\rangle[x\backslash u']\, q'[x\backslash u'], \\ t_2 &:= (\mathtt{V}''\langle t'\rangle\, q')[x\backslash u'], \\ t_3 &:= (\mathtt{V}''\langle s'\rangle\, q')[x\backslash u']. \end{aligned}$$

4. **Inductive case 2: right of an application** $\mathtt{V} = \mathtt{L}\langle \mathtt{v}\rangle\mathtt{R}'$**.** The situation is:

$$t = \mathtt{L}\langle \mathtt{v}\rangle\mathtt{R}'\langle t'\rangle \to \mathtt{L}\langle \mathtt{v}\rangle\mathtt{R}'\langle s'\rangle = s$$

Reasoning as in the previous case (*left of an application*), if the $\Leftrightarrow$ step is internal to $\mathtt{R}'\langle t'\rangle$, the result follows by *i.h.*, and if it is internal to $\mathtt{L}\langle \mathtt{v}\rangle$, it is straightforward to close the diagram by resorting to the fact that $\equiv$ preserves the shape of $\mathtt{L}\langle \mathtt{v}\rangle$ (Lem. A.8).

The remaining possibility is that the $\Leftrightarrow$ step overlaps both $\mathtt{L}\langle \mathtt{v}\rangle$ and $\mathtt{R}'\langle t'\rangle$. As in the previous case, this can only be possible if $\equiv_{\mathtt{gc}}$ introduces a substitution out of the blue, which is a trivial case, or because of a *Commutation with application* rule ($\equiv_{@}$, from right to left). This again leaves two possibilities:

4.1 *The substitution comes from* t'. That is, $\mathtt{R}' = \Box$ and t' is a $\mapsto_{\mathtt{lslsv}}$-redex $t' = \mathtt{V}''\langle y\rangle[y\backslash \mathtt{L}'\langle \mathtt{v}'\rangle]$. Moreover, $\mathtt{L} = \mathtt{L}''[y\backslash \mathtt{L}'\langle \mathtt{v}'\rangle]$. Then:

$$\begin{array}{ccc} \mathtt{L}''\langle \mathtt{v}\rangle[y\backslash \mathtt{L}'\langle \mathtt{v}'\rangle]\, \mathtt{V}''\langle y\rangle[y\backslash \mathtt{L}'\langle \mathtt{v}'\rangle] & \xrightarrow{\mathtt{lsv}} & t_1 \\ \equiv_{@} & & \equiv \\ t_2 & \overset{\mathtt{lsv}}{\dashrightarrow} & t_3 \end{array}$$

where

$$\begin{aligned} t_1 &:= \mathtt{L}''\langle \mathtt{v}\rangle[y\backslash \mathtt{L}'\langle \mathtt{v}'\rangle]\, \mathtt{L}'\langle \mathtt{V}''\langle \mathtt{v}'\rangle[y\backslash \mathtt{v}']\rangle, \\ t_2 &:= (\mathtt{L}''\langle \mathtt{v}\rangle\, \mathtt{V}''\langle y\rangle)[y\backslash \mathtt{L}'\langle \mathtt{v}'\rangle], \\ t_3 &:= \mathtt{L}'\langle (\mathtt{L}''\langle \mathtt{v}\rangle\, \mathtt{V}''\langle \mathtt{v}'\rangle)[y\backslash \mathtt{v}']\rangle. \end{aligned}$$

Exactly as in the previous case, for the equivalence on the right consider:

$$\begin{array}{ll} & \mathtt{L}''\langle \mathtt{v}\rangle[y\backslash \mathtt{L}'\langle \mathtt{v}'\rangle]\, \mathtt{L}'\langle \mathtt{V}''\langle \mathtt{v}'\rangle[y\backslash \mathtt{v}']\rangle \\ \equiv^{*}_{[\cdot]} & \mathtt{L}'\langle \mathtt{L}''\langle \mathtt{v}\rangle[y\backslash \mathtt{v}']\rangle\, \mathtt{L}'\langle \mathtt{V}''\langle \mathtt{v}'\rangle[y\backslash \mathtt{v}']\rangle \\ \equiv^{*}_{@} & \mathtt{L}'\langle \mathtt{L}''\langle \mathtt{v}\rangle[y\backslash \mathtt{v}']\, \mathtt{V}''\langle \mathtt{v}'\rangle[y\backslash \mathtt{v}']\rangle \\ \equiv_{@} & \mathtt{L}'\langle (\mathtt{L}''\langle \mathtt{v}\rangle\, \mathtt{V}''\langle \mathtt{v}'\rangle)[y\backslash \mathtt{v}']\rangle \end{array}$$

4.2 *The substitution comes from* $\mathtt{R}'$. That is, $\mathtt{R}' = \mathtt{V}''[x\backslash u']$. Moreover, $\mathtt{L} = \mathtt{L}'[x\backslash u']$. This case is then straightforward:

$$\begin{array}{ccc} \mathtt{L}'\langle \mathtt{v}\rangle[x\backslash u']\,\mathtt{V}''\langle t'\rangle[x\backslash u'] & \longrightarrow & \mathtt{L}'\langle \mathtt{v}\rangle[x\backslash u']\,\mathtt{V}''\langle s'\rangle[x\backslash u'] \\ \equiv_{@} & & \equiv_{@} \\ (\mathtt{L}'\langle \mathtt{v}\rangle\,\mathtt{V}''\langle t'\rangle)[x\backslash u'] & \dashrightarrow & (\mathtt{L}'\langle \mathtt{v}\rangle\,\mathtt{V}''\langle s'\rangle)[x\backslash u'] \end{array}$$

5. **Inductive case 3: left of a substitution** $\mathtt{V} = \mathtt{R}'[x\backslash q]$**.** The situation is:

$$t = \mathtt{R}'\langle t'\rangle[x\backslash q] \to \mathtt{R}'\langle s'\rangle[x\backslash q] = s$$

If the $\Leftrightarrow$ step is internal to $\mathtt{R}'\langle t'\rangle$, the result follows by *i.h.*. If it is internal to q, the steps are orthogonal, which makes the diagram trivial. If the equivalence $\equiv_{\mathtt{gc}}$ introduces a substitution out of the blue the steps trivially commute.

The remaining possibility is that the substitution $[x\backslash q]$ is involved in the $\Leftrightarrow$ redex. By case analysis on the kind of the step $\equiv_b$:

5.1 *Garbage collection* $\equiv_{\mathtt{gc}}$. We know $x \notin \mathtt{fv}(\mathtt{R}'\langle t'\rangle)$ and therefore also $x \notin \mathtt{fv}(\mathtt{R}'\langle s'\rangle)$. We get:

$$\begin{array}{ccc} \mathtt{R}'\langle t'\rangle[x\backslash q] & \longrightarrow & \mathtt{R}'\langle s'\rangle[x\backslash q] \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ \mathtt{R}'\langle t'\rangle & \dashrightarrow & \mathtt{R}'\langle s'\rangle \end{array}$$

5.2 *Duplication* $\equiv_{\mathtt{dup}}$. The important fact is that if $\mathtt{R}'\langle t'\rangle \to \mathtt{R}'\langle s'\rangle$ and $\mathtt{R}'\langle t'\rangle_{[y]_x}$ denotes the result of renaming some (arbitrary) occurrences of x by y in $\mathtt{R}'\langle t'\rangle$, then $\mathtt{R}'\langle t'\rangle_{[y]_x} \to \mathtt{R}'\langle s'\rangle_{[y]_x}$, where $\mathtt{R}'\langle s'\rangle_{[y]_x}$ denotes the result of renaming some occurrences of x by y in $\mathtt{R}'\langle s'\rangle$. By this we conclude:

$$\begin{array}{ccc} \mathtt{R}'\langle t'\rangle[x\backslash q] & \longrightarrow & \mathtt{R}'\langle s'\rangle[x\backslash q] \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ (\mathtt{R}'\langle t'\rangle)_{[y]_x}[x\backslash q][y\backslash q] & \dashrightarrow & (\mathtt{R}'\langle s'\rangle)_{[y]_x}[x\backslash q][y\backslash q] \end{array}$$

5.3 *Commutation with application* $\equiv_{@}$. $\mathtt{R}'\langle t'\rangle$ must be an application. This allows for three possibilities:

5.3.1 *The application comes from* t'. That is, $\mathtt{R}' = \Box$ and t' is a $\mapsto_{\mathtt{dblsv}}$-redex $t' = \mathtt{L}\langle \lambda y.t''\rangle\,\mathtt{L}'\langle \mathtt{v}\rangle$. The diagram is exactly as for the multiplicative base case 1.3 (read bottom-up).

5.3.2 *The application comes from* $\mathtt{R}'$*, left case.* That is, $\mathtt{R}' = \mathtt{V}''\,u'$. This case is direct:

$$\begin{array}{ccc} (\mathtt{V}''\langle t'\rangle\,u')[x\backslash q] & \longrightarrow & t_1 \\ \equiv_{@} & & \equiv_{@} \\ t_2 & \dashrightarrow & t_3 \end{array}$$

where

$$\begin{aligned} t_1 &:= (\mathtt{V}''\langle s'\rangle\, u')[x\backslash q], \\ t_2 &:= \mathtt{V}''\langle t'\rangle[x\backslash q]\, u'[x\backslash q], \\ t_3 &:= \mathtt{V}''\langle s'\rangle[x\backslash q]\, u'[x\backslash q]. \end{aligned}$$

5.3.3 *The application comes from* $\mathtt{R}'$*, right case.* That is, $\mathtt{R}' = \mathtt{L}\langle \mathtt{v}\rangle\, \mathtt{V}''$. Analogous to the previous case.

5.4 *Commutation of independent substitutions* $\equiv_{\mathtt{com}}$. Since $\mathtt{R}'\langle t'\rangle$ must have a substitution at the root, there are two possibilities:

5.4.1 *The substitution comes from* t'. That is, $\mathtt{R}' = \Box$ and t' is a $\mapsto_{\mathtt{lslsv}}$-redex $t' = \mathtt{V}''\langle y\rangle[y\backslash \mathtt{L}\langle \mathtt{v}\rangle]$, with $x \notin \mathtt{fv}(\mathtt{L}\langle \mathtt{v}\rangle)$. Then:

$$\begin{array}{ccc} \mathtt{V}''\langle y\rangle[y\backslash \mathtt{L}\langle \mathtt{v}\rangle][x\backslash q] & \xrightarrow{\ \mathtt{lsv}\ } & \mathtt{L}\langle \mathtt{V}''\langle \mathtt{v}\rangle[y\backslash \mathtt{v}]\rangle[x\backslash q] \\ \equiv_{\mathtt{com}} & & \equiv^{*}_{\mathtt{com}} \\ \mathtt{V}''\langle y\rangle[x\backslash q][y\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \dashrightarrow^{\mathtt{lsv}} & \mathtt{L}\langle \mathtt{V}''\langle \mathtt{v}\rangle[x\backslash q][y\backslash \mathtt{v}]\rangle \end{array}$$

5.4.2 *The substitution comes from* $\mathtt{R}'$. That is, $\mathtt{R}' = \mathtt{V}''[y\backslash u']$ with $x \notin \mathtt{fv}(u')$. This case is direct:

$$\begin{array}{ccc} \mathtt{V}''\langle t'\rangle[y\backslash u'][x\backslash q] & \xrightarrow{\ \mathtt{lsv}\ } & \mathtt{V}''\langle s'\rangle[y\backslash u'][x\backslash q] \\ \equiv_{\mathtt{com}} & & \equiv_{\mathtt{com}} \\ \mathtt{V}''\langle t'\rangle[x\backslash q][y\backslash u'] & \dashrightarrow^{\mathtt{lsv}} & \mathtt{V}''\langle s'\rangle[x\backslash q][y\backslash u'] \end{array}$$

5.5 *Composition of substitutions* $\equiv_{[\cdot]}$. As in the previous case, there are two possibilities:

5.5.1 *The substitution comes from* t'. That is, $\mathtt{R}' = \Box$ and t' is a $\mapsto_{\mathtt{lslsv}}$-redex $t' = \mathtt{V}''\langle y\rangle[y\backslash \mathtt{L}\langle \mathtt{v}\rangle]$, with $x \notin \mathtt{fv}(\mathtt{V}''\langle y\rangle)$. Then:

$$\begin{array}{ccc} \mathtt{V}''\langle y\rangle[y\backslash \mathtt{L}\langle \mathtt{v}\rangle][x\backslash q] & \xrightarrow{\ \mathtt{lsv}\ } & \mathtt{L}\langle \mathtt{V}''\langle \mathtt{v}\rangle[y\backslash \mathtt{v}]\rangle[x\backslash q] \\ \equiv_{[\cdot]} & & = \\ \mathtt{V}''\langle y\rangle[y\backslash \mathtt{L}\langle \mathtt{v}\rangle[x\backslash q]] & \dashrightarrow^{\mathtt{lsv}} & \mathtt{L}\langle \mathtt{V}''\langle \mathtt{v}\rangle[y\backslash \mathtt{v}]\rangle[x\backslash q] \end{array}$$

5.5.2 *The substitution comes from* $\mathtt{R}'$. That is, $\mathtt{R}' = \mathtt{V}''[y\backslash u']$ with $x \notin \mathtt{fv}(\mathtt{V}''\langle t'\rangle)$. The proof for this case is direct:

$$\begin{array}{ccc} \mathtt{V}''\langle t'\rangle[y\backslash u'][x\backslash q] & \longrightarrow & \mathtt{V}''\langle s'\rangle[y\backslash u'][x\backslash q] \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ \mathtt{V}''\langle t'\rangle[y\backslash u'[x\backslash q]] & \dashrightarrow & \mathtt{V}''\langle s'\rangle[y\backslash u'[x\backslash q]] \end{array}$$

Call-by-Need (need)

We follow the structure of the previous proofs of strong bisimulation, in particular the proof is by induction on the derivation of the reduction step. Remember that for call-by-need the definition of the structural equivalence is given only by axioms $\equiv_{@l}$, $\equiv_{com}$, and $\equiv_{[\cdot]}$.

We need two preliminary lemmas, proved by straightforward inductions on $\mathtt{N}$:

Lemma A.10. *Let t be a term, $\mathtt{N}$ be a call-by-need evaluation context not capturing any variable in $\mathtt{fv}(t)$, and $x \notin \mathtt{fv}(\mathtt{R}\langle y\rangle)$. Then $\mathtt{R}\langle t[x\backslash s]\rangle \equiv_{\mathtt{Need}} \mathtt{R}\langle t\rangle[x\backslash s]$.*

Lemma A.11. *The equivalence relation $\equiv_{\mathtt{Need}}$ preserves the shape of $\mathtt{R}\langle x\rangle$. More precisely, if $\mathtt{R}\langle x\rangle \equiv_{\mathtt{Need}} t$, with x not captured by $\mathtt{N}$, then t is of the form $\mathtt{R}'\langle x\rangle$, with x not captured by $\mathtt{R}'$.*

1. **Base case 1: multiplicative root step** $t = \mathtt{L}\langle\lambda x.t'\rangle q \mapsto_{\mathtt{db}} s = \mathtt{L}\langle t'[x\backslash q]\rangle$**.** Every application of $\equiv$ inside q or inside one of the substitutions in $\mathtt{L}$ trivially commutes with the step. The interesting cases are those where structural equivalence has a critical pair with the step:

   1.1 *Commutation with left of an application* $\equiv_{@l}$. If $\mathtt{L} = \mathtt{L}'[y\backslash r]$ then

   $$\begin{array}{ccc} \mathtt{L}'\langle\lambda x.t'\rangle[y\backslash r]q & \xrightarrow{\mathtt{db}} & \mathtt{L}'\langle t'[x\backslash q]\rangle[y\backslash r] \\ \equiv_{@l} & & = \\ (\mathtt{L}'\langle\lambda x.t'\rangle q)[y\backslash r] & \dashrightarrow^{\mathtt{db}} & \mathtt{L}'\langle t'[x\backslash q]\rangle[y\backslash r] \end{array}$$

   1.2 *Commutation of independent substitutions* $\equiv_{com}$. The substitutions that are commuted by the $\equiv_{com}$ rule must be both in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle\mathtt{L}''[y\backslash u'][z\backslash r']\rangle$ with $z \notin \mathtt{fv}(u')$. Let $\widehat{\mathtt{L}} = \mathtt{L}'\langle\mathtt{L}''[z\backslash r'][y\backslash u']\rangle$. Then:

   $$\begin{array}{ccc} \mathtt{L}\langle\lambda x.t'\rangle s' & \xrightarrow{\mathtt{db}} & \mathtt{L}\langle t'[x\backslash s']\rangle \\ \equiv_{com} & & \equiv_{com} \\ \widehat{\mathtt{L}}\langle\lambda x.t'\rangle s' & \dashrightarrow^{\mathtt{db}} & \widehat{\mathtt{L}}\langle t'[x\backslash s']\rangle \end{array}$$

   1.3 *Composition of substitutions* $\equiv_{[\cdot]}$. The substitutions that appear in the left-hand side of the $\equiv_{[\cdot]}$ rule must both be in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle\mathtt{L}''[y\backslash u'][z\backslash r']\rangle$ with $z \notin \mathtt{fv}(\mathtt{L}''\langle\lambda x.t'\rangle)$. Let $\widehat{\mathtt{L}} = \mathtt{L}'\langle\mathtt{L}''[y\backslash u'[z\backslash r']]\rangle$. Exactly as in the previous case:

   $$\begin{array}{ccc} \mathtt{L}\langle\lambda x.t'\rangle s' & \xrightarrow{\mathtt{db}} & \mathtt{L}\langle t'[x\backslash s']\rangle \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ \widehat{\mathtt{L}}\langle\lambda x.t'\rangle s' & \dashrightarrow^{\mathtt{db}} & \widehat{\mathtt{L}}\langle t'[x\backslash s']\rangle \end{array}$$

2. **Base case 2: exponential root step** $t = \mathtt{R}\langle x\rangle[x\backslash\mathtt{L}\langle\mathtt{v}\rangle] \mapsto_{\mathtt{lslsv}} s = \mathtt{L}\langle\mathtt{R}\langle\mathtt{v}\rangle[x\backslash\mathtt{v}]\rangle$**.** Consider first the case when the $\Leftrightarrow$-redex is internal to $\mathtt{R}\langle x\rangle$. By Lem. A.11 we know

$\Leftrightarrow$ preserves the shape of $\mathtt{R}\langle x\rangle$, *i.e.* $\mathtt{R}\langle x\rangle \Leftrightarrow \widehat{\mathtt{N}}\langle x\rangle$. Then:

$$\begin{array}{ccc} \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\quad\mathtt{lsv}\quad} & \mathtt{L}\langle \mathtt{R}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle \\ \Leftrightarrow & & = \\ \widehat{\mathtt{N}}\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \dashrightarrow^{\mathtt{lsv}} & \mathtt{L}\langle \widehat{\mathtt{N}}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle \end{array}$$

If the $\Leftrightarrow$-redex is internal to one of the substitutions in $\mathtt{L}$, the proof is straightforward. Note that the $\Leftrightarrow$-redex has always a substitution at the root. The remaining possibilities are that such substitution is in $\mathtt{L}$, or that it is precisely $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$. Axiom by axiom:

2.1 *Commutation with the left of an application* $\equiv_{@\mathtt{l}}$. The only possibility is that the substitution $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$ is commuted with the outermost application in $\mathtt{R}\langle x\rangle$, *i.e.* $\mathtt{N} = \mathtt{R}'t'$. The diagram is:

$$\begin{array}{ccc} (\mathtt{R}'\langle x\rangle\, t')[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\quad\mathtt{lsv}\quad} & \mathtt{L}\langle (\mathtt{R}'\langle \mathtt{v}\rangle\, t')[x\backslash \mathtt{v}]\rangle \\ \equiv_{@\mathtt{l}} & & \equiv^*_{@\mathtt{l}} \\ \mathtt{R}'\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]t' & \dashrightarrow^{\mathtt{lsv}} & \mathtt{L}\langle \mathtt{R}'\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle\, t' \end{array}$$

2.2 *Commutation of independent substitutions* $\equiv_{\mathtt{com}}$. Two sub-cases:

2.2.1 *The commuted substitutions both belong to* $\mathtt{L}$. Let $\widehat{\mathtt{L}}$ be the result of commuting them, and the diagram is:

$$\begin{array}{ccc} \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\quad\mathtt{lsv}\quad} & \mathtt{L}\langle \mathtt{R}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle \\ \equiv_{\mathtt{com}} & & \equiv_{\mathtt{com}} \\ \mathtt{R}\langle x\rangle[x\backslash \widehat{\mathtt{L}}\langle \mathtt{v}\rangle] & \dashrightarrow^{\mathtt{lsv}} & \widehat{\mathtt{L}}\langle \mathtt{R}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle \end{array}$$

2.2.2 *One of the commuted substitutions is* $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$. Then $\mathtt{N} = \mathtt{R}'[y\backslash t']$ and $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$ commutes with $[y\backslash t']$ (which implies $x \notin \mathtt{fv}(t')$). Then:

$$\begin{array}{ccc} \mathtt{R}'\langle x\rangle[y\backslash t'][x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \xrightarrow{\quad\mathtt{lsv}\quad} & \mathtt{L}\langle \mathtt{R}'\langle \mathtt{v}\rangle[y\backslash t'][x\backslash \mathtt{v}]\rangle \\ \equiv_{\mathtt{com}} & & \equiv^*_{\mathtt{com}} \\ \mathtt{R}'\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle][y\backslash t'] & \dashrightarrow^{\mathtt{lsv}} & \mathtt{L}\langle \mathtt{R}'\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle[y\backslash t'] \end{array}$$

2.3 *Composition of substitutions* $\equiv_{[\cdot]}$. Two sub-cases:

2.3.1 *The composed substitutions both belong to* $\mathtt{L}$. Analogous to case 2.2.1.

2.3.2 *One of the composed subtitutions is* $[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$. This is not possible if the rule is applied from left to right, since it would imply that $\mathtt{R}\langle x\rangle = \mathtt{R}'\langle x\rangle[y\backslash t']$ with $x \notin \mathtt{R}'\langle x\rangle$, which is a contradiction.
Finally, if the $\equiv_{[\cdot]}$ rule is applied from right to left, $\mathtt{L}$ is of the form $\mathtt{L}'[y\backslash t']$ and:

$$\begin{array}{ccc} \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}'\langle \mathtt{v}\rangle[y\backslash t']] & \xrightarrow{\quad\mathtt{lsv}\quad} & \mathtt{L}'\langle \mathtt{R}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle[y\backslash t'] \\ \equiv_{[\cdot]} & & = \\ \mathtt{R}\langle x\rangle[x\backslash \mathtt{L}'\langle \mathtt{v}\rangle][y\backslash t'] & \dashrightarrow^{\mathtt{lsv}} & \mathtt{L}'\langle \mathtt{R}\langle x\rangle[x\backslash \mathtt{v}]\rangle[y\backslash t'] \end{array}$$

3. **Inductive case 1: left of an application** $\mathtt{N} = \mathtt{R}'q$**.** The situation is:

$$t = \mathtt{R}'\langle t'\rangle\, q \to \mathtt{R}'\langle s'\rangle\, q = s$$

If the $\Leftrightarrow$ step is internal to $\mathtt{R}'\langle t'\rangle$, the result follows by *i.h.*. The proof is also direct if $\Leftrightarrow$ is internal to q. The nontrivial cases are those where $\Leftrightarrow$ overlaps $\mathtt{R}'\langle t'\rangle$ and q. The only possible case is that a substitution commutes with the topmost application via $\equiv_{@\mathtt{l}}$ (applied from right to left). There are two cases:

3.1 *The substitution comes from* t'. That is, $\mathtt{R}' = \Box$ and t' has a substitution at its root. Then t' must be a $\mapsto_{\mathtt{lslsv}}$-redex $t' = \mathtt{N}''\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]$. We have:

$$\begin{array}{ccc} \mathtt{N}''\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{v}\rangle]\, q & \xrightarrow{\quad\mathtt{lsv}\quad} & \mathtt{L}\langle \mathtt{N}''\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\rangle\, q \\ \equiv_{@\mathtt{l}} & & \equiv^{*}_{@\mathtt{l}} \\ (\mathtt{N}''\langle x\rangle\, q)[x\backslash \mathtt{L}\langle \mathtt{v}\rangle] & \dashrightarrow^{\mathtt{lsv}} & \mathtt{L}\langle (\mathtt{N}''\langle \mathtt{v}\rangle\, q)[x\backslash \mathtt{v}]\rangle \end{array}$$

3.2 *The substitution comes from* $\mathtt{R}'$. That is: $\mathtt{R}' = \mathtt{N}''[x\backslash u']$. The proof is then straightforward:

$$\begin{array}{ccc} \mathtt{N}''\langle t'\rangle[x\backslash u']\, q & \longrightarrow & \mathtt{N}''\langle s'\rangle[x\backslash u']\, q \\ \equiv_{@\mathtt{l}} & & \equiv_{@\mathtt{l}} \\ (\mathtt{N}''\langle t'\rangle\, q)[x\backslash u'] & \dashrightarrow & (\mathtt{N}''\langle s'\rangle\, q)[x\backslash u'] \end{array}$$

4. **Inductive case 2: left of a substitution** $\mathtt{N} = \mathtt{R}'[x\backslash q]$**.** The situation is:

$$t = \mathtt{R}'\langle t'\rangle[x\backslash q] \to \mathtt{R}'\langle s'\rangle[x\backslash q] = s$$

If the $\Leftrightarrow$ step is internal to $\mathtt{R}'\langle t'\rangle$, the result follows by *i.h.*. If it is internal to q, the steps are orthogonal, which makes the diagram trivial. The remaining possibility is that the substitution $[x\backslash q]$ is involved in the $\Leftrightarrow$ redex. By case analysis on the kind of the step $\equiv_b$:

4.1 *Commutation with the left of an application* $\equiv_{@\mathtt{l}}$. $\mathtt{R}'\langle t'\rangle$ must be an application. Two possibilities:

4.1.1 *The application comes from* t'. That is, $\mathtt{R}' = \Box$ and t' is a $\mapsto_{\mathtt{db}}$-redex $t' = \mathtt{L}\langle \lambda y.t''\rangle\, r$. This is exactly as the base case 1.1 (read bottom-up).

4.1.2 *The application comes from* $\mathtt{R}'$, i.e. $\mathtt{R}' = \mathtt{N}''\, u'$. This is exactly as the inductive case 3.2 (read bottom-up).

4.2 *Commutation of independent substitutions* $\equiv_{\mathtt{com}}$. Since $\mathtt{R}'\langle t'\rangle$ must have a substitution at the root, there are two possibilities:

4.2.1 *The substitution comes from* t'. That is, $\mathtt{R}' = \Box$ and t' is a $\mapsto_{\mathtt{lslsv}}$-redex $t' = \mathtt{N}''\langle y\rangle[y\backslash \mathtt{L}\langle \mathtt{v}\rangle]$, with $x \notin \mathtt{fv}(\mathtt{L}\langle \mathtt{v}\rangle)$. This case is exactly as the base exponential case 2.2.2 (read bottom-up).

4.2.2 *The substitution comes from* $\mathtt{R}'$. That is, $\mathtt{R}' = \mathtt{N}''[y\backslash u']$ with $x \notin \mathtt{fv}(u')$. The diagram is:

$$\begin{array}{ccc} \mathtt{N}''\langle t'\rangle[y\backslash u'][x\backslash q] & \xrightarrow{\mathtt{lsv}} & \mathtt{N}''\langle s'\rangle[y\backslash u'][x\backslash q] \\ \equiv_{\mathtt{com}} & & \equiv_{\mathtt{com}} \\ \mathtt{N}''\langle t'\rangle[x\backslash q][y\backslash u'] & \xrightarrow{\mathtt{lsv}} & \mathtt{N}''\langle s'\rangle[x\backslash q][y\backslash u'] \end{array}$$

4.3 *Composition of substitutions* $\equiv_{[\cdot]}$. As in the previous case, there are two possibilities:

4.3.1 *The substitution comes from* t'. That is, $\mathtt{R}' = \Box$ and t' is a $\mapsto_{\mathtt{lslsv}}$-redex $t' = \mathtt{N}''\langle y\rangle[y\backslash \mathtt{L}\langle \mathtt{v}\rangle]$, with $x \notin \mathtt{fv}(\mathtt{N}''\langle y\rangle)$. This case is exactly as the base exponential case 2.3.2 (read bottom-up).

4.3.2 *The substitution comes from* $\mathtt{R}'$. That is, $\mathtt{R}' = \mathtt{N}''[y\backslash u']$ with $x \notin \mathtt{fv}(\mathtt{N}''\langle t'\rangle)$. The diagram is:

$$\begin{array}{ccc} \mathtt{N}''\langle t'\rangle[y\backslash u'][x\backslash q] & \longrightarrow & \mathtt{N}''\langle s'\rangle[y\backslash u'][x\backslash q] \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ \mathtt{N}''\langle t'\rangle[y\backslash u'[x\backslash q]] & \dashrightarrow & \mathtt{N}''\langle s'\rangle[y\backslash u'[x\backslash q]] \end{array}$$

5. **Inductive case 3: inside a hereditary head substitution** $\mathtt{N} = \mathtt{R}'\langle x\rangle[x\backslash \mathtt{N}'']$**.** The situation is:

$$t = \mathtt{R}'\langle x\rangle[x\backslash \mathtt{N}''\langle q\rangle] \to \mathtt{R}'\langle x\rangle[x\backslash \mathtt{N}''\langle q'\rangle] = s$$

If $\Leftrightarrow$ is internal to $\mathtt{R}'\langle x\rangle$ the two steps clearly commutes. If $\Leftrightarrow$ is internal to $\mathtt{N}''\langle q\rangle$ we conclude using the *i.h.*. The remaining cases are when $\Leftrightarrow$ overlaps with the topmost constructor. Axiom by axiom:

5.1 *Commutation with the left of an application* $\equiv_{@\mathtt{l}}$. It must be that $\mathtt{R}'\langle x\rangle = \mathtt{N}''''\langle x\rangle r$ with $x \notin \mathtt{fv}(r)$. Then the two steps simply commute:

$$\begin{array}{ccc} (\mathtt{N}''''\langle x\rangle r)[x\backslash \mathtt{N}''\langle q\rangle] & \longrightarrow & (\mathtt{N}''''\langle x\rangle r)[x\backslash \mathtt{N}''\langle q'\rangle] \\ \equiv_{@\mathtt{l}} & & \equiv_{@\mathtt{l}} \\ \mathtt{N}''''\langle x\rangle[x\backslash \mathtt{N}''\langle q\rangle]r & \dashrightarrow & \mathtt{N}''''\langle x\rangle[x\backslash \mathtt{N}''\langle q'\rangle]r \end{array}$$

5.2 *Commutation of independent substitutions* $\equiv_{\mathtt{com}}$. It must be that $\mathtt{R}'\langle x\rangle = \mathtt{N}''''\langle x\rangle[y\backslash r]$ with $x \notin \mathtt{fv}(r)$. Then the two steps simply commute:

$$\begin{array}{ccc} \mathtt{N}''''\langle x\rangle[y\backslash r][x\backslash \mathtt{N}''\langle q\rangle] & \longrightarrow & t_1 \\ \equiv_{@\mathtt{l}} & & \equiv_{@\mathtt{l}} \\ t_2 & \dashrightarrow & t_3 \end{array}$$

where

$$\begin{aligned} t_1 &:= \mathtt{N}''''\langle x\rangle[y\backslash r][x\backslash \mathtt{N}''\langle q'\rangle], \\ t_2 &:= \mathtt{N}''''\langle x\rangle[x\backslash \mathtt{N}''\langle q\rangle][y\backslash r], \\ t_3 &:= \mathtt{N}''''\langle x\rangle[x\backslash \mathtt{N}''\langle q'\rangle][y\backslash r]. \end{aligned}$$

5.3 *Composition of substitutions* $\equiv_{[\cdot]}$. There are various sub-cases

5.3.1 $[x\backslash \mathtt{N}''\langle q\rangle]$ *enters in a substitution.* It must be that $\mathtt{R}'\langle x\rangle = \mathtt{N}_1\langle y\rangle[y\backslash \mathtt{N}_2\langle x\rangle]$ with $x \notin \mathtt{fv}(\mathtt{N}_1\langle y\rangle)$. Then the diagram is:

$$\begin{array}{ccc} \mathtt{N}_1\langle y\rangle[y\backslash \mathtt{N}_2\langle x\rangle][x\backslash \mathtt{N}''\langle q\rangle] & \longrightarrow & t_1 \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ t_2 & \dashrightarrow & t_3 \end{array}$$

$$\begin{aligned} t_1 &:= \mathtt{N}_1\langle y\rangle[y\backslash \mathtt{N}_2\langle x\rangle][x\backslash \mathtt{N}''\langle q'\rangle], \\ t_2 &:= \mathtt{N}_1\langle y\rangle[y\backslash \mathtt{N}_2\langle x\rangle[x\backslash \mathtt{N}''\langle q\rangle]], \\ t_3 &:= \mathtt{N}_1\langle y\rangle[y\backslash \mathtt{N}_2\langle x\rangle[x\backslash \mathtt{N}''\langle q'\rangle]]. \end{aligned}$$

5.3.2 *a substitution pops out of* $[x\backslash \mathtt{N}''\langle q\rangle]$. Two sub-cases:

5.3.2.1 *The substitution comes from* $\mathtt{N}''$. Then $\mathtt{N}''\langle q\rangle = \mathtt{N}''''\langle q\rangle[y\backslash r]$. The diagram is:

$$\begin{array}{ccc} \mathtt{R}'\langle x\rangle[x\backslash \mathtt{N}''''\langle q\rangle[y\backslash r]] & \longrightarrow & t_1 \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ t_2 & \dashrightarrow & t_3 \end{array}$$

where

$$t_1 := \mathtt{R}'\langle x\rangle[x\backslash \mathtt{N}''''\langle q'\rangle[y\backslash r]], \tag{A.1}$$
$$t_2 := \mathtt{R}'\langle x\rangle[x\backslash \mathtt{N}''''\langle q\rangle][y\backslash r], \tag{A.2}$$
$$t_3 := \mathtt{R}'\langle x\rangle[x\backslash \mathtt{N}''''\langle q'\rangle][y\backslash r]. \tag{A.3}$$

5.3.2.2 *The substitution comes from* q. Then $\mathtt{N}'' = \Box$ and q is a $\mapsto_{\mathtt{lslsv}}$-redex $t' = \mathtt{N}''''\langle y\rangle[y\backslash \mathtt{L}\langle \mathtt{v}\rangle]$ and the diagram is:

$$\begin{array}{ccc} \mathtt{R}'\langle x\rangle[x\backslash \mathtt{N}''''\langle y\rangle[y\backslash \mathtt{L}\langle \mathtt{v}\rangle]] & \xrightarrow{\mathtt{lsv}} & t_1 \\ \equiv_{[\cdot]} & & \equiv^*_{[\cdot]} \\ t_2 & \overset{\mathtt{lsv}}{\dashrightarrow} & t_3 \end{array}$$

where

$$t_1 := \mathtt{R}'\langle x\rangle[x\backslash \mathtt{L}\langle \mathtt{N}''''\langle \mathtt{v}\rangle[y\backslash \mathtt{v}]\rangle], \tag{A.4}$$
$$t_2 := \mathtt{R}'\langle x\rangle[x\backslash \mathtt{N}''''\langle y\rangle][y\backslash \mathtt{L}\langle \mathtt{v}\rangle], \tag{A.5}$$
$$t_3 := \mathtt{L}\langle \mathtt{R}'\langle x\rangle[x\backslash \mathtt{N}''''\langle \mathtt{v}\rangle][y\backslash \mathtt{v}]\rangle. \tag{A.6}$$

Strong Call-by-Name ($\mathtt{name}^{\mathsf{S}}$)

We need the following auxiliary lemma:

Lemma A.12. *If $\mathcal{C}$ is a* LO *context and $\mathcal{C}$ does not bind any of the variables in $\mathtt{fv}(s)$, then $\mathcal{C}\langle t[x\backslash s]\rangle \equiv \mathcal{C}\langle t\rangle[x\backslash s]$.*

Proof. Recall that a context $\mathcal{C}$ is LO if and only if $\mathcal{C} \in \mathcal{S}$ (Lem. 3.10). The property is then proved by induction on the derivation that $\mathcal{C} \in \mathcal{S}$. □

Now we turn to the proof of bisimulation itself. As in the previous proofs of bisimulation, we proceed by induction on the derivation of the reduction step:

1. **Base case 1: multiplicative root step,** $t = \mathtt{L}\langle\lambda x.t'\rangle s' \mapsto_{\mathtt{db}} \mathtt{L}\langle t'[x\backslash s']\rangle$**.** If the $\Leftrightarrow$ step is internal to t', internal to s', or internal to the argument of one of the substitutions in $\mathtt{L}$, then the pattern of the $\Leftrightarrow$ redex does not overlap with the $\mapsto_{\mathtt{db}}$ step, and the proof is immediate, as the two steps commute. Otherwise, we consider every possible case of $\Leftrightarrow$:

   1.1 *Garbage collection,* $\equiv_{\mathtt{gc}}$. The garbage collected substitution must be one of the substitutions in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle\mathtt{L}''[y\backslash u']\rangle$. Then:

$$\begin{array}{ccc} \mathtt{L}'\langle\mathtt{L}''\langle\lambda x.t'\rangle[y\backslash z]\rangle s' & \xrightarrow{\mathtt{db}} & \mathtt{L}'\langle\mathtt{L}''\langle t'[x\backslash s']\rangle[y\backslash z]\rangle \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ \mathtt{L}'\langle\mathtt{L}''\langle\lambda x.t'\rangle\rangle s' & \dashrightarrow^{\mathtt{db}} & \mathtt{L}'\langle\mathtt{L}''\langle t'[x\backslash s']\rangle\rangle \end{array}$$

   1.2 *Commutation of independent substitutions,* $\equiv_{\mathtt{com}}$. The substitutions that are commuted must be both in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle\mathtt{L}''[y\backslash u'][z\backslash r']\rangle$. Then:

$$\begin{array}{ccc} \mathtt{L}'\langle\mathtt{L}''\langle\lambda x.t'\rangle[y\backslash u'][z\backslash r']\rangle s' & \xrightarrow{\mathtt{db}} & \mathtt{L}'\langle\mathtt{L}''\langle t'[x\backslash s']\rangle[y\backslash u'][z\backslash r']\rangle \\ \equiv_{\mathtt{com}} & & \equiv_{\mathtt{com}} \\ \mathtt{L}'\langle\mathtt{L}''\langle\lambda x.t'\rangle[z\backslash r'][y\backslash u']\rangle s' & \dashrightarrow^{\mathtt{db}} & \mathtt{L}'\langle\mathtt{L}''\langle t'[x\backslash s']\rangle[z\backslash r'][y\backslash u']\rangle \end{array}$$

   1.3 *Composition of substitutions,* $\equiv_{[\cdot]}$. The substitutions that are composed must be both in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle\mathtt{L}''[y\backslash u'][z\backslash r']\rangle$. Then:

$$\begin{array}{ccc} \mathtt{L}'\langle\mathtt{L}''\langle\lambda x.t'\rangle[y\backslash u'][z\backslash r']\rangle s' & \xrightarrow{\mathtt{db}} & \mathtt{L}'\langle\mathtt{L}''\langle t'[x\backslash s']\rangle[y\backslash u'][z\backslash r']\rangle \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ \mathtt{L}'\langle\mathtt{L}''\langle\lambda x.t'\rangle[y\backslash u'[z\backslash r']]\rangle s' & \dashrightarrow^{\mathtt{db}} & \mathtt{L}'\langle\mathtt{L}''\langle t'[x\backslash s']\rangle[y\backslash u'[z\backslash r']]\rangle \end{array}$$

   1.4 *Duplication,* $\equiv_{\mathtt{dup}}$. The duplicated substitution must be one of the substitutions in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle\mathtt{L}''[y\backslash u']\rangle$. Then:

$$\begin{array}{ccc} \mathtt{L}'\langle\mathtt{L}''\langle\lambda x.t'\rangle[y\backslash u']\rangle s' & \xrightarrow{\mathtt{db}} & \mathtt{L}'\langle\mathtt{L}''\langle t'[x\backslash s']\rangle[y\backslash u']\rangle \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ \mathtt{L}'\langle(\mathtt{L}''\langle\lambda x.t'\rangle)_{[z]_y}[y\backslash u'][z\backslash u']\rangle s' & \dashrightarrow^{\mathtt{db}} & \mathtt{L}'\langle(\mathtt{L}''\langle t'[x\backslash s']\rangle)_{[z]_y}[y\backslash u'][z\backslash u']\rangle \end{array}$$

   1.5 *Commutation with abstraction,* $\equiv_{\lambda}$. The commuted substitution must be the innermost substitution in $\mathtt{L}$, *i.e.* $\mathtt{L}$ must be of the form $\mathtt{L}'\langle[y\backslash u']\rangle$, and:

$$\begin{array}{ccc} \mathtt{L}'\langle(\lambda x.t')[y\backslash u']\rangle s' & \xrightarrow{\mathtt{ls}} & \mathtt{L}'\langle t'[x\backslash s'][y\backslash u']\rangle \\ \equiv_{\lambda} & & \equiv_{\mathtt{com}} \\ \mathtt{L}'\langle\lambda x.t'[y\backslash u']\rangle s' & \dashrightarrow^{\mathtt{ls}} & \mathtt{L}'\langle t'[y\backslash u'][x\backslash s']\rangle \end{array}$$

Note that the diagram can be also read from the bottom-up for a reverse application of the $\equiv_\lambda$ rule. In order to be able to apply $\equiv_{\mathtt{com}}$, note that $x \notin \mathtt{fv}(u')$ by application of the $\equiv_\lambda$ rule, and that $y \notin \mathtt{fv}(s')$ by the bound variable convention.

1.6 *Left commutation with application,* $\equiv_{@\mathtt{l}}$. The only possibility is that the outermost substitution of $\mathtt{L}$ commutes with the application taking part in the $\to_{\mathtt{db}}$ step. That is, $\mathtt{L}$ must be of the form $\mathtt{L}'[y\backslash u']$ and:

$$\begin{array}{ccc} \mathtt{L}'\langle\lambda x.t'\rangle[y\backslash u']s' & \xrightarrow{\quad\mathtt{db}\quad} & \mathtt{L}'\langle t'[x\backslash s']\rangle[y\backslash u']s' \\ \equiv_{@\mathtt{l}} & & = \\ (\mathtt{L}'\langle\lambda x.t'\rangle\, s')[y\backslash u'] & \dashrightarrow^{\mathtt{db}} & \mathtt{L}'\langle t'[x\backslash s']\rangle[y\backslash u'] \end{array}$$

1.7 *Right commutation with application,* $\equiv_{@\mathtt{r}}$. Note that every $\equiv_{@\mathtt{r}}$ (and ${\equiv_{@\mathtt{r}}}^{-1}$) redex in $(\lambda x.t')\mathtt{L}\, s'$ must be internal to either t', s', or the argument of one of the substitutions in $\mathtt{L}$. We have already argued that in these cases the steps commute.

2. **Base case 2: exponential root step,** $t = \mathcal{C}\langle x\rangle[x\backslash t'] \mapsto_{\mathtt{ls}} \mathcal{C}\langle t'\rangle[x\backslash t']$**.**

If the substitution that is contracted by the exponential step does not take part in the pattern of the $\Leftrightarrow$ step, it is immediate to check that the property holds. More precisely, suppose that $\mathcal{C}\langle x\rangle[x\backslash t'] \Leftrightarrow \mathtt{C}'\langle x\rangle[x\backslash t'']$, where $\mathtt{C}'$ and t'' result respectively from $\mathcal{C}$ and t by a single step of $\Leftrightarrow$. Note that we have that either $\mathcal{C} \Leftrightarrow \mathtt{C}'$ and $t' = t''$ or vice-versa. Then:

$$\begin{array}{ccc} \mathcal{C}\langle x\rangle[x\backslash t'] & \xrightarrow{\quad\mathtt{ls}\quad} & \mathcal{C}\langle t'\rangle[x\backslash t'] \\ \Leftrightarrow & & \Leftrightarrow^* \\ \mathtt{C}'\langle x\rangle[x\backslash t''] & \dashrightarrow^{\mathtt{ls}} & \mathtt{C}'\langle t''\rangle[x\backslash t''] \end{array}$$

Note that when commutation affects t' (*i.e.* if we are in the case in which $\mathcal{C} = \mathtt{C}'$ and $t' \Leftrightarrow t''$), then the right-hand side of the diagram must be closed by two $\Leftrightarrow$ steps: one for each copy of t'.

So we may assume that the substitution that is contracted by the exponential step does take part in the pattern of the $\Leftrightarrow$ step. We consider every possible case of $\Leftrightarrow$.

2.1 *Garbage collection,* $\equiv_{\mathtt{gc}}$. The garbage collected substitution cannot erase the contracted occurrence of x, since $\mathcal{C}$ is a LO context, and it cannot go inside substitutions. Two subcases, depending on the position of the hole of $\mathcal{C}$ with respect to the node of the garbage collected substitution:

2.1.1 If the hole of $\mathcal{C}$ lies inside the body of the garbage collected substitution, *i.e.* $\mathcal{C} = \mathtt{C}'\langle\mathtt{C}''[y\backslash s']\rangle$ with $y \notin \mathtt{fv}(\mathtt{C}''\langle x\rangle)$, then:

$$\begin{array}{ccc} \mathtt{C}'\langle\mathtt{C}''\langle x\rangle[y\backslash s']\rangle[x\backslash t'] & \xrightarrow{\quad\mathtt{ls}\quad} & \mathtt{C}'\langle\mathtt{C}''\langle t'\rangle[y\backslash s']\rangle[x\backslash t'] \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ \mathtt{C}'\langle\mathtt{C}''\langle x\rangle\rangle[x\backslash t'] & \dashrightarrow^{\mathtt{ls}} & \mathtt{C}'\langle\mathtt{C}''\langle t'\rangle\rangle[x\backslash t'] \end{array}$$

Note that $y \notin \mathtt{fv}(\mathtt{C}''\langle t'\rangle)$ since we may assume that $y \notin \mathtt{fv}(t')$ by the bound variable convention.

2.1.2 Otherwise, the hole of $\mathcal{C}$ must be disjoint from the node of the garbage collected substitution, *i.e.* there must be a two-hole context $\mathtt{C}'$ such that:

$$\mathcal{C} = \mathtt{C}'\langle \Box, s'[y\backslash u']\rangle$$

where $y \notin \mathtt{fv}(s')$. Then:

$$\begin{array}{ccc} \mathtt{C}'\langle x, s'[y\backslash u']\rangle[x\backslash t'] & \xrightarrow{\quad \mathtt{ls} \quad} & \mathtt{C}'\langle t', s'[y\backslash u']\rangle[x\backslash t'] \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ \mathtt{C}'\langle x, s'\rangle[x\backslash t'] & \dashrightarrow^{\mathtt{ls}} & \mathtt{C}'\langle t', s'\rangle[x\backslash t'] \end{array}$$

2.2 *Commutation of independent substitutions,* $\equiv_{\mathtt{com}}$. Note that the contracted occurrence of x cannot be inside the argument of any of the commuted substitutions, since $\mathcal{C}$ is a LO context and it cannot go inside substitutions. Since the contracted substitution is commuted, we have that $\mathcal{C}$ must be of the form $\mathtt{C}'[y\backslash s']$ and the situation is:

$$\begin{array}{ccc} \mathtt{C}'\langle x\rangle[y\backslash s'][x\backslash t'] & \xrightarrow{\quad \mathtt{ls} \quad} & \mathtt{C}'\langle t'\rangle[y\backslash s'][x\backslash t'] \\ \equiv_{\mathtt{com}} & & \equiv_{\mathtt{com}} \\ \mathtt{C}'\langle x\rangle[x\backslash t'][y\backslash s'] & \dashrightarrow^{\mathtt{ls}} & \mathtt{C}'\langle t'\rangle[x\backslash t'][y\backslash s'] \end{array}$$

2.3 *Composition of substitutions,* $\equiv_{[\cdot]}$. Note that the contracted occurrence of x cannot be inside the argument of any of the two substitutions that take part in the $\equiv_{[\cdot]}$ step, since $\mathcal{C}$ is a LO context and it cannot go inside substitutions. We know that the contracted substitution takes part in the $\equiv_{[\cdot]}$ step. We consider two subcases, depending on whether the $\equiv_{[\cdot]}$ rule is applied from left to right or from right to left, since the situation is not symmetrical.

2.3.1 If the $\equiv_{[\cdot]}$ step is applied from left to right, then $\mathcal{C}$ must be of the form $\mathtt{C}'[y\backslash s']$ with $x \notin \mathtt{fv}(\mathtt{C}'\langle x\rangle)$. This is a contradiction, so this case is not actually possible.

2.3.2 If the $\equiv_{[\cdot]}$ step is applied from right to left, then t' must be of the form $t''[y\backslash s']$ and:

$$\begin{array}{ccc} \mathcal{C}\langle x\rangle[x\backslash t''[y\backslash s']] & \xrightarrow{\quad \mathtt{ls} \quad} & \mathcal{C}\langle t''[y\backslash s']\rangle[x\backslash t''[y\backslash s']] \\ \equiv_{[\cdot]} & & \equiv \\ \mathcal{C}\langle x\rangle[x\backslash t''][y\backslash s'] & \dashrightarrow^{\mathtt{ls}} & \mathcal{C}\langle t''\rangle[x\backslash t''][y\backslash s'] \end{array}$$

To close the right-hand side of the diagram, we are left to show that:

$$\mathcal{C}\langle t''[y\backslash s']\rangle[x\backslash t''[y\backslash s']] \equiv \mathcal{C}\langle t''\rangle[x\backslash t''][y\backslash s']$$

First note that $\mathcal{C}$ is a LO context, and that, by the bound variable convention, $\mathcal{C}$ does not bind any of the variables in $\mathtt{fv}(s')$. By resorting to Lemma A.12,

this allows us to commute the substitution that:

$$\begin{array}{lll} & \mathcal{C}\langle t''[y\backslash s']\rangle[x\backslash t''[y\backslash s']] & \\ \equiv & \mathcal{C}\langle t''\rangle[y\backslash s'][x\backslash t''[y\backslash s']] & \text{by Lemma A.12} \\ \equiv_{[\cdot]} & \mathcal{C}\langle t''\rangle[y\backslash s'][x\backslash t''][y\backslash s'] & \\ = & \mathcal{C}\langle t''\rangle[y\backslash s'][x\backslash t''\{y/z\}][z\backslash s'] & \text{renaming } y \text{ to } z \\ \equiv_{\mathtt{com}} & \mathcal{C}\langle t''\rangle[x\backslash t''\{y/z\}][y\backslash s'][z\backslash s'] & \\ \equiv_{\mathtt{dup}} & \mathcal{C}\langle t''\rangle[x\backslash t''][y\backslash s'] & \end{array}$$

2.4 *Duplication,* $\equiv_{\mathtt{dup}}$. Note that the contracted occurrence of x cannot be inside the argument of any of the two substitutions that take part in the $\equiv_{\mathtt{dup}}$ step, since $\mathcal{C}$ is a LO context and it cannot go inside substitutions. We consider two cases, depending on whether $\equiv_{\mathtt{dup}}$ is applied from left to right or from right to left:

2.4.1 From left to right: the contracted occurrence of x is either renamed to y or left untouched as x. Let z denote x or y, correspondingly. In both cases we have:

$$\begin{array}{ccc} \mathcal{C}\langle x\rangle[x\backslash t'] & \xrightarrow{\quad\mathtt{ls}\quad} & \mathcal{C}\langle t'\rangle[x\backslash t'] \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ \mathcal{C}_{[y]_x}\langle z\rangle[x\backslash t'][y\backslash t'] & \overset{\mathtt{ls}}{\dashrightarrow} & \mathcal{C}_{[y]_x}\langle t'\rangle[x\backslash t'][y\backslash t'] \end{array}$$

2.4.2 From right to left: then $\mathcal{C}$ is of the form $\mathtt{C}'_{[x]_y}[y\backslash t']$, where $\mathtt{C}'$ has no occurrences of x, and:

$$\begin{array}{ccc} \mathtt{C}'_{[x]_y}\langle x\rangle[y\backslash t'][x\backslash t'] & \xrightarrow{\quad\mathtt{ls}\quad} & \mathtt{C}'_{[x]_y}\langle t'\rangle[y\backslash t'][x\backslash t'] \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ \mathtt{C}'\langle y\rangle[y\backslash t'] & \overset{\mathtt{ls}}{\dashrightarrow} & \mathtt{C}'\langle t'\rangle[y\backslash t'] \end{array}$$

2.5 *Commutation with abstraction,* $\equiv_\lambda$. Then $\mathcal{C}$ is of the form $\lambda y.\mathtt{C}'$ and:

$$\begin{array}{ccc} (\lambda y.\mathtt{C}'\langle x\rangle)[x\backslash t'] & \xrightarrow{\quad\mathtt{ls}\quad} & (\lambda y.\mathtt{C}'\langle t'\rangle)[x\backslash t'] \\ \equiv_\lambda & & \equiv_\lambda \\ \lambda y.\mathtt{C}'\langle x\rangle[x\backslash t'] & \overset{\mathtt{ls}}{\dashrightarrow} & \lambda y.\mathtt{C}'\langle t'\rangle[x\backslash t'] \end{array}$$

2.6 *Left commutation with application,* $\equiv_{@\mathtt{l}}$. Then $\mathcal{C}$ is of the form $\mathcal{C}\ s'$ and:

$$\begin{array}{ccc} (\mathcal{C}\langle x\rangle\ s')[x\backslash t'] & \xrightarrow{\quad\mathtt{ls}\quad} & (\mathcal{C}\langle t'\rangle\ s')[x\backslash t'] \\ \equiv_{@\mathtt{l}} & & \equiv_{@\mathtt{l}} \\ \mathcal{C}\langle x\rangle[x\backslash t']\ s' & \overset{\mathtt{ls}}{\dashrightarrow} & \mathcal{C}\langle t'\rangle[x\backslash t']\ s' \end{array}$$

2.7 *Right commutation with application,* $\equiv_{@\mathtt{r}}$. Then $\mathcal{C}$ is of the form $s'\,\mathcal{C}$ and:

$$\begin{array}{ccc} (s'\,\mathcal{C}\langle x\rangle)[x\backslash t'] & \xrightarrow{\quad\mathtt{ls}\quad} & (s'\,\mathcal{C}\langle t'\rangle)[x\backslash t'] \\ \equiv_{@\mathtt{r}} & & \equiv_{@\mathtt{r}} \\ s'\,\mathcal{C}\langle x\rangle[x\backslash t'] & \overset{\mathtt{ls}}{\dashrightarrow} & s'\,\mathcal{C}\langle t'\rangle[x\backslash t'] \end{array}$$

3. **Inductive case 1: inside an abstraction.** Suppose that $t = \lambda x.t' \to \lambda x.s' = s$. We consider two subcases, depending on whether the $\Leftrightarrow$ step is internal to the body of the abstraction, or involves the outermost abstraction:

   3.1 If the application of the $\Leftrightarrow$ step is internal to t', we have by *i.h.*:

   $$\begin{array}{ccc} t' & \longrightarrow & s' \\ \equiv & & \equiv \\ u' & \dashrightarrow & r' \end{array}$$

   so is immediate to conclude that:

   $$\begin{array}{ccc} \lambda x.t' & \longrightarrow & \lambda x.s' \\ \equiv & & \equiv \\ \lambda x.u' & \dashrightarrow & \lambda x.r' \end{array}$$

   3.2 If the outermost abstraction takes part in the $\Leftrightarrow$ step, then a $\equiv_\lambda$ step must have been applied, so t' must be of the form $t''[y\backslash s']$. We consider two further subcases, depending on whether the commuted substitution is involved in the reduction step:

   3.2.1 If the reduction step $t''[y\backslash s'] \to u'$ is an exponential, and the commuted substitution $[y\backslash s']$ is the one contracted by the exponential step, then the situation is exactly like in case 2.5 (*Commutation with abstraction* for exponential steps), by reading the diagram from the bottom up.

   3.2.2 Otherwise, note that there cannot be a multiplicative step at the root, and that the step cannot be internal to s', as LO contexts do not go inside substitutions. Therefore the reduction step must be internal to t'' and the situation is:

   $$\begin{array}{ccc} \lambda x.t''[y\backslash s'] & \longrightarrow & \lambda x.s''[y\backslash s'] \\ \equiv_\lambda & & \equiv_\lambda \\ (\lambda x.t'')[y\backslash s'] & \dashrightarrow & (\lambda x.s'')[y\backslash s'] \end{array}$$

4. **Inductive case 2: left of an application.** Suppose that $t = t'\,q \to s'\,q = s$. If the application of the $\Leftrightarrow$ step is internal to t', we may immediately conclude by *i.h.* (analogous to case 3.1). The interesting case is when the outermost application takes part in the $\Leftrightarrow$ step. There are two possibilities, depending on whether a $\equiv_{@\mathtt{l}}$ step or a $\equiv_{@\mathtt{r}}$ step is applied:

   4.1 $\equiv_{@\mathtt{l}}$ **step.** Then t' must be of the form $t''[x\backslash u']$. We consider two further subcases, depending on whether the commuted substitution is involved in the reduction step:

   4.1.1 If the reduction step $t''[x\backslash u'] \to r'$ is an exponential step and the commuted substitution $[x\backslash u']$ is also the one contracted by the exponential step, then the situation is exactly like in case 2.6 (*Left commutation with application* for exponential steps), by reading the diagram from the bottom up.

4.1.2 Otherwise, note that the reduction step cannot be internal to u', since LO contexts do not go inside substitutions, so it must be internal to t'' and the situation is:

$$\begin{array}{ccc} t''[x\backslash u']\,q & \longrightarrow & s''[x\backslash u']\,q \\ \equiv_{@\mathtt{l}} & & \equiv_{@\mathtt{l}} \\ (t''\,q)[x\backslash u'] & \dashrightarrow & (s''\,q)[x\backslash u'] \end{array}$$

4.2 $\equiv_{@\mathtt{r}}$ **step.** Then q must be of the form $q'[x\backslash u']$ and the situation is:

$$\begin{array}{ccc} t'\,q'[x\backslash u'] & \longrightarrow & s'\,q'[x\backslash u'] \\ \equiv_{@\mathtt{r}} & & \equiv_{@\mathtt{r}} \\ (t'\,q')[x\backslash u'] & \dashrightarrow & (s'\,q')[x\backslash u'] \end{array}$$

5. **Inductive case 3: right of an application.** Suppose that $t = q\,t' \to q\,s' = s$. If the application of the $\Leftrightarrow$ step is internal to t', we may immediately conclude by *i.h.* (analogous to case 3.1). The interesting case is when the outermost application takes part in the $\Leftrightarrow$ step. There are two possibilities, depending on whether a $\equiv_{@\mathtt{l}}$ step or a $\equiv_{@\mathtt{r}}$ step is applied:

5.1 $\equiv_{@\mathtt{l}}$ **step.** Then q must be of the form $q'[x\backslash u']$ and the situation is:

$$\begin{array}{ccc} q'[x\backslash u']\,t' & \longrightarrow & q'[x\backslash u']\,s' \\ \equiv_{@\mathtt{l}} & & \equiv_{@\mathtt{l}} \\ (q'\,t')[x\backslash u'] & \dashrightarrow & (q'\,s')[x\backslash u'] \end{array}$$

5.2 $\equiv_{@\mathtt{r}}$ **step.** Then t' must be of the form $t''[x\backslash u']$. We consider two further subcases, depending on whether the commuted substitution is involved in the reduction step:

5.2.1 If the reduction step $t''[x\backslash u'] \to r'$ is an exponential step and the commuted substitution $[x\backslash u']$ is also the one contracted by the exponential step, then the situation is exactly like in case 2.7 (*Right commutation with application* for exponential steps), by reading the diagram from the bottom up.

5.2.2 Otherwise, note that the reduction step cannot be internal to u', since LO contexts do not go inside substitutions, so it must be internal to t'' and the situation is:

$$\begin{array}{ccc} q\,t''[x\backslash u'] & \longrightarrow & q\,s''[x\backslash u'] \\ \equiv_{@\mathtt{r}} & & \equiv_{@\mathtt{r}} \\ (q\,t'')[x\backslash u'] & \dashrightarrow & (q\,s'')[x\backslash u'] \end{array}$$

6. **Inductive case 4: left of a substitution.** Suppose that $t = t'[x\backslash q] \to s'[x\backslash q] = s$. If the application of the $\Leftrightarrow$ step is internal to t', we may immediately conclude by *i.h.* (analogous to case 3.1). The interesting case is when the outermost substitution node takes part in the $\Leftrightarrow$ step. There are four possibilities, depending on whether a $\equiv_{\mathtt{gc}}$ step, a $\equiv_{\mathtt{com}}$ step, a $\equiv_{[\cdot]}$ step, or a $\equiv_{\mathtt{dup}}$ step is applied:

6.1 $\equiv_{\mathtt{gc}}$ **step.** The reduction step cannot be internal to q, since LO contexts may not go inside substitutions, so the step must be internal to t', and closing the diagram is trivial:

$$\begin{array}{ccc} t'[x\backslash q] & \longrightarrow & s'[x\backslash q] \\ \equiv_{\mathtt{gc}} & & \equiv_{\mathtt{gc}} \\ t' & \dashrightarrow & s' \end{array}$$

Note that if $x \notin \mathtt{fv}(t')$ then $x \notin \mathtt{fv}(s')$ by the usual property that reduction does not create free variables.

6.2 $\equiv_{\mathtt{com}}$ **step.** Then t' must be of the form $t''[y\backslash u']$ with $x \notin \mathtt{fv}(u')$. We consider two further subcases, depending on whether the commuted substitution is involved in the reduction step:

6.2.1 If the reduction step $t''[y\backslash u'] \to r'$ is an exponential step and the commuted substitution $[y\backslash u']$ is also the one contracted by the exponential step, then the situation is exactly like in case 2.2 (*Commutation of independent substitutions* for exponential steps), by reading the diagram from the bottom up.

6.2.2 Otherwise, note that the reduction step cannot be internal to u', since LO contexts may not go inside substitutions, so it must be internal to t'', and the situation is:

$$\begin{array}{ccc} t''[y\backslash u'][x\backslash q] & \longrightarrow & s''[y\backslash u'][x\backslash q] \\ \equiv_{\mathtt{com}} & & \equiv_{\mathtt{com}} \\ t''[x\backslash q][y\backslash u'] & \dashrightarrow & s''[x\backslash q][y\backslash u'] \end{array}$$

6.3 $\equiv_{[\cdot]}$ **step.** Two cases, depending on whether the $\equiv_{[\cdot]}$ step is applied from left to right or from right to left:

6.3.1 $\equiv_{[\cdot]}$ *is applied from left to right.* Then t' must be of the form $t''[y\backslash u']$ with $x \notin \mathtt{fv}(t'')$. We consider two further subcases, depending on whether the commuted substitution is involved in the reduction step:

6.3.1.1 If the reduction step $t''[y\backslash u'] \to r'$ is an exponential step and the commuted substitution $[y\backslash u']$ is also the one contracted by the exponential step, then the situation is exactly like in case 2.3.2 (*Composition of substitutions* for exponential steps), by reading the diagram from the bottom up.

6.3.1.2 Otherwise, note that the reduction step cannot be internal to u', since LO contexts may not go inside substitutions, so it must be internal to t'', and the situation is:

$$\begin{array}{ccc} t''[y\backslash u'][x\backslash q] & \longrightarrow & s''[y\backslash u'][x\backslash q] \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ t''[y\backslash u'[x\backslash q]] & \dashrightarrow & s''[y\backslash u'[x\backslash q]] \end{array}$$

Note that if $x \notin \mathtt{fv}(t'')$, then $x \notin \mathtt{fv}(s'')$, by the usual fact that reduction does not create free variables.

6.3.2 $\equiv_{[\cdot]}$ *is applied from right to left.* Then q must be of the form $q'[y\backslash u']$, and the reduction step must be internal to t', so the situation is:

$$\begin{array}{ccc} t'[x\backslash q'[y\backslash u']] & \longrightarrow & s'[x\backslash q'[y\backslash u']] \\ \equiv_{[\cdot]} & & \equiv_{[\cdot]} \\ t'[x\backslash q'][y\backslash u'] & \dashrightarrow & s'[x\backslash q'][y\backslash u'] \end{array}$$

6.4 $\equiv_{\mathtt{dup}}$ **step.** Two cases, depending on whether the $\equiv_{\mathtt{dup}}$ step is applied from left to right or from right to left:

6.4.1 $\equiv_{\mathtt{dup}}$ *is applied from left to right.* Then the reduction step is internal to t' and closing the diagram is immediate:

$$\begin{array}{ccc} t'[x\backslash q] & \longrightarrow & s'[x\backslash q] \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ t'_{[y]_x}[x\backslash q][y\backslash q] & \dashrightarrow & s'_{[y]_x}[x\backslash q][y\backslash q] \end{array}$$

6.4.2 $\equiv_{\mathtt{dup}}$ *is applied from right to left.* Then t' must be of the form $t''[y\backslash q]$. We consider two further subcases, depending on whether the commuted substitution is involved in the reduction step:

6.4.2.1 If the reduction step $t''[y\backslash q] \to r'$ is an exponential step and the affected substitution $[y\backslash q]$ is also the one contracted by the exponential step, then t'' must be of the form $\mathsf{C}'_{[x]_y}\langle y\rangle$ and the situation is:

$$\begin{array}{ccc} \mathsf{C}'_{[x]_y}\langle y\rangle[y\backslash q][x\backslash q] & \xrightarrow{\mathtt{ls}} & \mathsf{C}'_{[x]_y}\langle q\rangle[y\backslash q][x\backslash q] \\ \equiv_{\mathtt{dup}} & & \equiv_{\mathtt{dup}} \\ \mathsf{C}'\langle y\rangle[y\backslash q] & \overset{\mathtt{ls}}{\dashrightarrow} & \mathsf{C}'\langle q\rangle[y\backslash q] \end{array}$$

6.4.2.2 Otherwise, note that the reduction step cannot be internal to q, since LO contexts may not go inside substitutions, so it must be internal to t''. The situation is then exactly like in case 6.4.1, by reading the diagram from the bottom up.

□

A.1.3 Pointing MAD invariants — proof of Lem. 3.57

Lemma A.13 (Full proof of Lem. 3.57—Pointing MAD invariants)**.** *Let $S = \bar{t} \mid E \mid \pi \mid D$ be a Pointing MAD reachable state whose initial code $\bar{t}$ is well-named. Then:*

1. Subterm: *any code in S is a literal subterm of $\bar{t}$;*
2. Names: *the global closure of S is well-named.*
3. Dump-Environment Compatibility:

3.1 $(\llbracket\pi\rrbracket\langle\bar{t}\rangle, E\restriction)$ *is closed;*

3.2 for every pair (x, π') *in* D, $(\llbracket\pi'\rrbracket\langle x\rangle, E\restriction_x)$ *is closed;*

3.3 $E \propto D$ *holds.*

4. Contextual Decoding: $\llbracket(E, D)\rrbracket$ *is a call-by-need evaluation context.*

Proof. By induction on the length of the execution. Points 1 and 2 are by direct inspection of the rules. Assuming $E \propto D$, point 4 is immediate by induction on the length of D.

Thus we are only left to check point 3. We use point 2, *i.e.* that substitutions in E bind pairwise distinct variables. Following we show that transitions preserve the invariant:

1. *Conmutative 1.* We have:

$$\bar{t}\,\bar{s} \mid \pi \mid D \mid E \rightsquigarrow_{\mathtt{s}_1} \bar{t} \mid \bar{s} :: \pi \mid D \mid E$$

Trivial, since the dump and the environment are the same and $\llbracket(\bar{s} :: \pi)\rrbracket\langle\bar{t}\rangle = \llbracket\pi\rrbracket\langle\bar{t}\,\bar{s}\rangle$.

2. *Conmutative 2.* We have $S \rightsquigarrow_{\mathtt{s}_2} S'$ with:

$$S = x \mid \pi \mid D \mid E_1 :: [x\backslash\bar{t}] :: E_2$$

$$S' = \bar{t} \mid \epsilon \mid (x, \pi) :: D \mid E_1 :: [x\backslash\square] :: E_2$$

Note that since by *i.h.* $(\llbracket\pi\rrbracket\langle x\rangle, (E_1 :: [x\backslash\bar{t}] :: E_2)\restriction)$ is closed and x is free in $\llbracket\pi\rrbracket\langle x\rangle$, there cannot be any dumped substitutions in E_2. Then $(E_1 :: [x\backslash t] :: E_2)\restriction = E_1\restriction :: [x\backslash t] :: E_2$ and we know:

$$(\llbracket\pi\rrbracket\langle x\rangle, E_1\restriction :: [x\backslash\bar{t}] :: E_2) \text{ is closed} \tag{A.7}$$

For 3.1, note $(E_1 :: [x\backslash\square] :: E_2)\restriction = E_2$. Then we must show $(\bar{t}, E_2)$ is closed, which is implied by (A.7).

For 3.2, there are two cases:

- If the pair is (x, π), we must show

$$(\llbracket\pi\rrbracket\langle x\rangle, (E_1 :: [x\backslash\square] :: E_2)\restriction_x) \text{ is closed, } \textit{i.e.}$$

$$(\llbracket\pi\rrbracket\langle x\rangle, E_1\restriction :: [x\backslash\square] :: E_2) \text{ is closed}$$

  which is implied by (A.7).

- If the pair is (y, π') in D, with $y \neq x$, note first that

$$(E_1 :: [x\backslash\bar{t}] :: E_2)\restriction_y = E_1\restriction_y :: [x\backslash\bar{t}] :: E_2$$

  And similarly for $(E_1 :: [x\backslash\square] :: E_2)\restriction_y$. Moreover, by the invariant on S we know

$$(\llbracket\pi'\rrbracket\langle y\rangle, E_1\restriction_y :: [x\backslash\bar{t}] :: E_2) \text{ is closed}$$

  and this implies

$$(\llbracket\pi'\rrbracket\langle y\rangle, E_1\restriction_y :: [x\backslash\square] :: E_2) \text{ is closed}$$

  as required.

For 3.3, we have already observed that E_2 has no dumped substitutions. Then $[x\backslash\Box]$ is the rightmost dumped substitution in the environment of S', while (x, π) is the leftmost pair in the dump. We conclude by the fact that the invariant already holds for S.

3. *Multiplicative, empty dump.* We have $S \rightsquigarrow_{\mathtt{m}} S'$ with:

$$S = \lambda x.\bar{t} \mid \overline{s} :: \pi \mid \epsilon \mid E$$

$$S' = \bar{t} \mid \pi \mid \epsilon \mid [x\backslash\overline{s}] :: E$$

First note that, since the environment and the dump are dual in S, there are no dumped substitutions in E.

For point 3.1, we know that:

$$([\![\pi]\!]\langle(\lambda x.\bar{t})\,\overline{s}\rangle, E) \text{ is closed} \tag{A.8}$$

and we have to check:

$$([\![\pi]\!]\langle\bar{t}\rangle, [x\backslash\overline{s}] :: E) \text{ is closed}$$

Let $y \in \mathtt{fv}([\![\pi]\!]\langle\bar{t}\rangle)$. Then either $y = x$, which is bound by $[x\backslash\overline{s}]$, or $y \in \mathtt{fv}([\![\pi]\!]\langle\lambda x.\bar{t}\rangle)$, in which case y is bound by E. Moreover, since $[\![\pi]\!]$ is an application context, by (A.8) we get $(\overline{s}, E)$ is closed.

Points 3.2 and 3.3 are trivial since the dump is empty and the environment has no dumped substitutions.

4. *Multiplicative, non-empty dump.* We have $S \rightsquigarrow_{\mathtt{m}} S'$ with:

$$S = \lambda x.\bar{t} \mid \overline{s} :: \pi \mid (y, \pi') :: D \mid E_1 :: [y\backslash\Box] :: E_2$$

$$S' = \bar{t} \mid \pi \mid (y, \pi') :: D \mid E_1 :: [y\backslash\Box] :: [x\backslash\overline{s}] :: E_2$$

Note first that since the invariant holds for S, we know $[y\backslash\Box]$ is the rightmost dumped substitution in the environment of both S and S'. Therefore $(E_1 :: [y\backslash\Box] :: E_2)\restriction = E_2$

For proving point 3.1, we have:

$$([\![\pi]\!]\langle(\lambda x.\bar{t})\,\overline{s}\rangle, E_2) \text{ is closed}$$

and we must show:

$$([\![\pi]\!]\langle\bar{t}\rangle, [x\backslash\overline{s}] :: E_2) \text{ is closed}$$

The situation is exactly as in point 3.1 for the $\rightsquigarrow_{\mathtt{m}}$ transition, empty dump case.

For point 3.2, let (z, π'') be any pair in $(y, \pi') :: D$. Let also

$$E_1' := \begin{cases} E_1\restriction & \text{if } y = z \\ E_1\restriction_z & \text{otherwise} \end{cases}$$

and note that $(E_1 :: [y\backslash\Box] :: E) \upharpoonright_y = E_1' :: [y\backslash\Box] :: E$ for any environment E that contains no dumped substitutions. By the invariant on S, we have that:

$$(\llbracket\pi''\rrbracket\langle z\rangle, E_1' :: [y\backslash\Box] :: E_2) \text{ is closed}$$

Moreover, from point 3.1 we know $(\overline{s}, E_2)$ is closed. Both imply:

$$(\llbracket\pi''\rrbracket\langle z\rangle, E_1' :: [y\backslash\Box] :: [x\backslash\overline{s}] :: E_2) \text{ is closed}$$

as required.

For point 3.3, just note that the substitution $[x\backslash\overline{s}]$ added to the environment is not dumped, and so duality holds because it holds for S by *i.h.*.

5. *Exponential.* We have $S \rightsquigarrow_{\mathsf{e}} S'$ with:

$$S = \overline{\mathtt{v}} \mid \epsilon \mid (x, \pi) :: D \mid E_1 :: [x\backslash\Box] :: E_2$$

$$S' = \overline{\mathtt{v}}^{\alpha} \mid \pi \mid D \mid E_1 :: [x\backslash\overline{\mathtt{v}}] :: E_2$$

First note that since the environment and the dump are dual in S, we know E_2 has no dumped substitutions.

For proving point 3.1, by resorting to point 3.1 on the state S, for which the invariant already holds, we have that:

$$(\overline{\mathtt{v}}, E_2) \text{ is closed} \tag{A.9}$$

Moreover, by point 3.2 on S, specialized on the pair (x, π), we also know:

$$(\llbracket\pi\rrbracket\langle x\rangle, E_1 \upharpoonright :: [x\backslash\Box] :: E_2) \text{ is closed} \tag{A.10}$$

We must check that:

$$(\llbracket\pi\rrbracket\langle\overline{\mathtt{v}}^{\alpha}\rangle, E_1 \upharpoonright :: [x\backslash\overline{\mathtt{v}}] :: E_2) \text{ is closed}$$

Any free variable in $\llbracket\pi\rrbracket\langle\mathtt{v}^{\alpha}\rangle$ is either free in π, in which case by (A.9) it must be bound by $E_1 \upharpoonright :: [x\backslash\Box] :: E_2$, or free in $\overline{\mathtt{v}}$, in which case by (A.9) it must be bound by E_2. In both cases it is bound by $E_1 \upharpoonright :: [x\backslash\overline{\mathtt{v}}] :: E_2$, as required. To conclude the proof of point 3.1, note that by combining (A.9) and (A.10) we get $E_1 \upharpoonright :: [x\backslash\mathtt{v}] :: E_2$ is closed.

For proving point 3.2, let (y, π') be a pair in D. Using that $x \neq y$, by the invariant on S we know:

$$(\llbracket\pi'\rrbracket\langle y\rangle, E_1 \upharpoonright_y :: [x\backslash\Box] :: E_2) \text{ is closed}$$

and this implies:

$$(\llbracket\pi'\rrbracket\langle y\rangle, E_1 \upharpoonright_y :: [x\backslash\overline{\mathtt{v}}] :: E_2) \text{ is closed}$$

as wanted.

Point 3.3 is immediate, given that the environment and the dump are already dual in S.

□

A.1.4 Strong MAM invariants — proof of Lem. 3.64

Lemma A.14 (Full proof of Lem. 3.64—Strong MAM invariants). *Let $S = \varphi \mid F \mid \overline{s} \mid \pi \mid E$ be a state reachable from an initial term $\overline{t}_0$. Then:*

1. Compatibility: *F and E are compatible,* i.e. $F \propto E$.
2. Normal Form:
   - 2.1 Backtracking Code: *if $\varphi = \Uparrow$, then $\overline{s}$ is normal, and if π is non-empty, then $\overline{s}$ is neutral.*
   - 2.2 Frame: *if $F = F' :: (\overline{u}, \pi') :: F''$, then $\overline{u}$ is neutral.*
3. Backtracking Free Variables:
   - 3.1 Backtracking Code: *if $\varphi = \Uparrow$ then $\mathtt{fv}(\overline{s}) \subseteq \Lambda(F)$.*
   - 3.2 Pairs in the Frame: *if $F = F' :: (\overline{u}, \pi') :: F''$ then $\mathtt{fv}(\overline{u}) \subseteq \Lambda(F'')$.*
4. Name:
   - 4.1 Substitutions: *if $E = E' :: [x\backslash\overline{t}] :: E''$ then x is fresh wrt $\overline{t}$ and E''.*
   - 4.2 Markers: *if $E = E' :: \rhd x :: E''$ and $F = F' :: x :: F''$ then x is fresh wrt E'' and F'', and $E'(y) = \bot$ for any free variable y in F''.*
   - 4.3 Abstractions: *if $\mathbf{a}x\overline{t}$ is a subterm of F, $\overline{s}$, π, or E then x may occur only in $\overline{t}$ and in the closed subenvironment $x\lhd :: E_{\mathrm{w}} :: \rhd x$ of E, if it exists.*
5. Closure:
   - 5.1 Environment: *if $E = E' :: [x\backslash\overline{t}] :: E''$ then $E''(y) \neq \bot$ for all $y \in \mathtt{fv}(\overline{t})$.*
   - 5.2 Code, Stack, and Frame: *$E(x) \neq \bot$ for any free variable in $\overline{s}$ and in any code of π and F.*

We prove each of the items in each of the following subsections:

Compatibility Invariant

By induction on the length of the number of transitions to reach S. The invariant trivially holds for an initial state. For a non-empty evaluation sequence we list the cases for the last transitions. We only deal with those that act on the frame or on the environment, as the others immediately follows from the *i.h.*.

- **Case** $(F, \lambda x.\overline{t}, \overline{s} :: \pi, E, \Downarrow) \rightsquigarrow_{\mathtt{m}} (F, \overline{t}, \pi, [x\backslash\overline{s}] :: E, \Downarrow)$. By *i.h.* F and E are compatible, *i.e.* $F = (F_{\mathrm{w}} :: F_{\mathrm{t}}) \propto (E_{\mathrm{w}} :: E_{\mathrm{t}}) = E$ with $F_{\mathrm{t}} \propto E_{\mathrm{t}}$. Since $[x\backslash\overline{s}] :: E_{\mathrm{w}}$ is still a weak environment, we have $(F_{\mathrm{w}} :: F_{\mathrm{t}}) \propto ([x\backslash\overline{s}] :: E_{\mathrm{w}} :: E_{\mathrm{t}})$, *i.e.* $F \propto ([x\backslash\overline{s}] :: E)$.
- **Case** $(F, \lambda x.\overline{t}, \epsilon, E, \Downarrow) \rightsquigarrow_{\Downarrow_{\mathtt{s}_2}} (x :: F, \overline{t}, \epsilon, \rhd x :: E, \Downarrow)$. By *i.h.* $F \propto E$. By definition of compatibility we obtain $(x :: F) \propto (\rhd x :: E)$.

- **Case** $(x :: F, \bar{t}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_4} (F, \lambda x.\bar{t}, \epsilon, x\lhd :: E, \Uparrow)$**.** By *i.h.*, $(x :: F)\propto E$. By the factorization property of compatible pairs (Lem. 3.63) $E = E_{\mathrm{w}} :: \rhd x :: E'$ with $F \propto E'$. Now $x\lhd :: E = x\lhd :: E_{\mathrm{w}} :: \rhd x :: E' = E'_{\mathrm{w}} :: E'$. Then, from $F \propto E'$ by definition $F \propto (E'_{\mathrm{w}} :: E')$, *i.e.* $F \propto (x\lhd :: E)$.

- **Case** $((\bar{t}, \pi) :: F, \bar{s}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_5} (F, \bar{t}\bar{s}, \pi, E, \Uparrow)$**.** By *i.h.*, $((\bar{t}, \pi) :: F)\propto E$, so $F \propto E$ by (Lem. 3.63).

- **Case** $(F, \bar{t}, \bar{s} :: \pi, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_6} ((\bar{t}, \pi) :: F, \bar{s}, \epsilon, E, \Downarrow)$**.** By *i.h.*, we have that $F \propto E$ which implies $((\bar{t}, \pi) :: F)\propto E$ by (Lem. 3.63).

Normal Form Invariant

The invariant trivially holds for an initial state $\Downarrow \mid \epsilon \mid \bar{t} \mid \epsilon \mid \epsilon$. For a non-empty evaluation sequence we list the cases for the last transitions. We only consider the cases for backtracking phases ($\Uparrow$) or when the frame changes, the others ($\rightsquigarrow_{\Downarrow \mathtt{s}_1}, \rightsquigarrow_{\mathtt{m}}, \rightsquigarrow_{\mathtt{e}}$) are omitted because they follow immediately from the *i.h.*.

- **Case** $(F, \lambda x.\bar{t}, \epsilon, E, \Downarrow) \rightsquigarrow_{\Downarrow \mathtt{s}_2} (x :: F, \bar{t}, \epsilon, \rhd x :: E, \Downarrow)$**.**

  1. Trivial since $\varphi \neq \Uparrow$.
  2. Suppose $x :: F$ can be written as $x :: F' :: (\bar{s}, \pi') :: F''$. Then by *i.h.* $\bar{s}$ is a neutral term.

- **Case** $(F, x, \pi, E, \Downarrow) \rightsquigarrow_{\Downarrow \mathtt{s}_3} (F, x, \pi, E, \Uparrow)$ **with** $E(x) = \rhd$**.** Note that $x \in \Lambda(E)$, because $E(x) = \rhd$.

  1. x is a normal and neutral term.
  2. It follows from the *i.h.*, as F is unchanged.

- **Case** $(x :: F, \bar{t}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_4} (F, \lambda x.\bar{t}, \epsilon, x\lhd :: E, \Uparrow)$**.**

  1. By *i.h.* we know that $\bar{t}$ is a normal form. Then $\lambda x.\bar{t}$ is a normal form. the stack is empty, so we conclude.
  2. It follows from the *i.h.*.

- **Case** $((\bar{t}, \pi) :: F, \bar{s}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_5} (F, \bar{t}\bar{s}, \pi, E, \Uparrow)$**.**

  1. By *i.h.* we have that $\bar{s}$ is a normal term while by *i.h.* $\bar{t}$ is neutral. Therefore $\bar{t}\bar{s}$ is a neutral term.
  2. It follows from the *i.h.*.

- **Case** $(F, \bar{t}, \bar{s} :: \pi, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_6} ((\bar{t}, \pi) :: F, \bar{s}, \epsilon, E, \Downarrow)$**.**

  1. Trivial since $\varphi \neq \Uparrow$.
  2. $\bar{t}$ is a neutral term by *i.h.*.

Backtracking Free Variables Invariant

The invariant trivially holds for an initial state $\Downarrow \mid \epsilon \mid \overline{t}_0 \mid \epsilon \mid \epsilon$ if $\overline{t}_0$ is closed and well-named. For a non-empty evaluation sequence we list the cases for the last transitions. We omit the transitions involving only states in evaluating phase, as for them everything follows immediately from the *i.h.*.

- **Case** $(F, y, \pi, E, \Downarrow) \rightsquigarrow_{\Downarrow \mathtt{s}_3} (F, y, \pi, E, \Uparrow)$ **with** $E(y) = \rhd$**.**

  1. *Backtracking Code*: by hypothesis $E(y) = \rhd$, and so $y \in \Lambda(E) = \Lambda(F)$ by Lem. 3.63.
  2. *Pairs in the Frame*: it follows from the *i.h.*.

- **Case** $(y :: F, \overline{u}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_4} (F, \lambda y.\overline{u}, \epsilon, y \lhd :: E, \Uparrow)$**.**

  1. *Backtracking Code*: by *i.h.* $\mathtt{fv}(\overline{u}) \subseteq \Lambda(y :: F)$ and so $\mathtt{fv}(\mathbf{a} y \overline{u}) = \mathtt{fv}(\overline{u}) \backslash \{x\} = \Lambda(F)$.
  2. *Pairs in the Frame*: it follows from the *i.h.*.

- **Case** $((\overline{u}, \pi) :: F, \overline{r}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_5} (F, \overline{ur}, \pi, E, \Uparrow)$**.**

  1. *Backtracking Code*: by *i.h.* $\mathtt{fv}(\overline{r}) \subseteq \Lambda((\overline{u}, \pi) :: F) = \Lambda(F)$ and by *i.h.* $\mathtt{fv}(\overline{u}) \subseteq \Lambda(F)$, and so $\mathtt{fv}(\overline{ur}) \subseteq \Lambda(F)$.
  2. *Pairs in the Frame*: it follows from the *i.h.*.

- **Case** $(F, \overline{u}, \overline{r} :: \pi, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_6} ((\overline{u}, \pi) :: F, \overline{r}, \epsilon, E, \Downarrow)$**.**

  1. *Backtracking Code*: nothing to prove.
  2. *Pairs in the Frame*: by *i.h.* $\mathtt{fv}(\overline{u}) \subseteq \Lambda(F)$, the rest follows immediately from the *i.h.*.

Name Invariant

The invariant trivially holds for an initial state $\Downarrow \mid \epsilon \mid \overline{u}_0 \mid \epsilon \mid \epsilon$ if $\overline{u}_0$ is closed and well-named. For a non-empty evaluation sequence we list the cases for the last transitions:

- **Case** $(F, \overline{ur}, \pi, E, \Downarrow) \rightsquigarrow_{\Downarrow \mathtt{s}_1} (F, \overline{u}, \overline{r} :: \pi, E, \Downarrow)$**.** Every point follows from its *i.h.*.

- **Case** $(F, \lambda y.\overline{u}, \overline{r} :: \pi, E, \Downarrow) \rightsquigarrow_{\mathtt{m}} (F, \overline{u}, \pi, [y \backslash \overline{r}] :: E, \Downarrow)$**.**

  1. *Substitutions*: for $[y \backslash \overline{r}]$ it follows from the *i.h.*, for E it follows from the *i.h.*.
  2. *Markers*: note that by *i.h.* y simply cannot occur in F, the rest follows from the *i.h.*.
  3. *Abstractions*: it follows from the *i.h.*.

- **Case** $(F, \lambda y.\overline{u}, \epsilon, E, \Downarrow) \rightsquigarrow_{\Downarrow \mathtt{s}_2} (y :: F, \overline{u}, \epsilon, \rhd y :: E, \Downarrow)$**.**

1. *Substitutions*: it follows from the *i.h.*.
2. *Markers*: for y it follows from the *i.h.*, the rest follows from the *i.h.*.
3. *Abstractions*: it follows from the *i.h.*.

- **Case** $(F, y, \pi, E, \Downarrow) \rightsquigarrow_{\mathtt{e}} (F, \overline{u}^{\alpha}, \pi, E, \Downarrow)$**.** It follows by the *i.h.* and the fact that in $\overline{u}^{\alpha}$ the abstracted variables are renamed (with respect to $\overline{u}$) with fresh names.
- **Case** $(F, y, \pi, E, \Downarrow) \rightsquigarrow_{\Downarrow \mathtt{s}_3} (F, y, \pi, E, \Uparrow)$**.** Every point follows from its *i.h.*.
- **Case** $(y :: F, \overline{u}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_4} (F, \lambda y.\overline{u}, \epsilon, y\lhd :: E, \Uparrow)$**.** By the compatibility invariant $(y :: F) \propto E$, and by the factorization property of compatible pairs (Lem. 3.63) $E = E_{\mathrm{w}} :: \rhd y :: E'$.

  1. *Substitutions*: it follows from the *i.h.*.
  2. *Markers*: it follows from the *i.h.*.
  3. *Abstractions*: for $\mathbf{a}y\overline{u}$ it holds because by *i.h.* y does not appear in F nor in E_{t} (it may however occur in E_{w}, but this is taken into account by the statement). For the other abstractions, it is immediate to conclude by *i.h.*.

- **Case** $((\overline{u}, \pi) :: F, \overline{r}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_5} (F, \overline{ur}, \pi, E, \Uparrow)$**.** Every point follows from its *i.h.*.
- **Case** $(F, \overline{u}, \overline{r} :: \pi, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_6} ((\overline{u}, \pi) :: F, \overline{r}, \epsilon, E, \Downarrow)$**.** Every point follows from its *i.h.*. □

Closure Invariant

The invariant trivially holds for an initial state $\Downarrow \mid \epsilon \mid \bar{t}_0 \mid \epsilon \mid \epsilon$ if $\bar{t}_0$ is closed and well-named. For a non-empty evaluation sequence we list the cases for the last transitions:

- **Case** $(F, \overline{ur}, \pi, E, \Downarrow) \rightsquigarrow_{\Downarrow \mathtt{s}_1} (F, \overline{u}, \overline{r} :: \pi, E, \Downarrow)$**.** Every point follows from its *i.h.*.
- **Case** $(F, \lambda y.\overline{u}, \overline{r} :: \pi, E, \Downarrow) \rightsquigarrow_{\mathtt{m}} (F, \overline{u}, \pi, [y \backslash \overline{r}] :: E, \Downarrow)$**.**

  1. *Environment*: for $[y \backslash \overline{r}]$ it follows from the *i.h.*, for the rest it follows from the *i.h.*.
  2. *Code, Stack, and Frame*: for y is evident, as $[y \backslash \overline{r}] :: E$ is clearly defined on y, for the rest it follows from the *i.h.*.

- **Case** $(F, \lambda y.\overline{u}, \epsilon, E, \Downarrow) \rightsquigarrow_{\Downarrow \mathtt{s}_2} (y :: F, \overline{u}, \epsilon, \rhd y :: E, \Downarrow)$**.**

  1. *Environment*: it follows from the *i.h.*.
  2. *Code, Stack, and Frame*: for y is evident, as $\rhd y :: E$ is clearly defined on y, for the rest it follows from the *i.h.*.

- **Case** $(F, y, \pi, E, \Downarrow) \rightsquigarrow_{\mathtt{e}} (F, \overline{u}^{\alpha}, \pi, E, \Downarrow)$**.**

  1. *Environment*: it follows from the *i.h.*.

2. *Code, Stack, and Frame*: for $\overline{u}^{\alpha}$ it follows from the *i.h.*, as $\overline{u}$ appears in the environment out of all closed scopes (otherwise the transition would not take place). The rest follows from the *i.h.*.

- **Case** $(F, y, \pi, E, \Downarrow) \rightsquigarrow_{\Downarrow \mathtt{s}_3} (F, y, \pi, E, \Uparrow)$ **with** $E(y) = \rhd$**.**

  1. *Environment*: it follows from the *i.h.*.
  2. *Code, Stack, and Frame*: it follows from the *i.h.*.

- **Case** $(y :: F, \overline{u}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_4} (F, \lambda y.\overline{u}, \epsilon, y\lhd :: E, \Uparrow)$**.** By the compatibility invariant $(y :: F) \propto E$, and by the factorization property of compatible pairs (Lem. 3.63) $E = E_{\mathrm{w}} :: \rhd y :: E'$.

  1. *Environment*: it follows from the *i.h.*.
  2. *Code, Stack, and Frame*: note that

     2.1 E_{w} does not bind any variable occurring free in $\overline{u}$ by the backtracking invariant,

     2.2 E_{w} does not bind any variable occurring free in F by the name invariant, and

     2.3 the stack is empty by hypothesis.

     Then E_{w} does not bind any free variable in the code, in the stack, nor in the frame, and we conclude using the *i.h.*, because $x\lhd :: E_{\mathrm{w}} :: \rhd x :: E'$ by definition is defined on a variable z if and only if E' is.

- **Case** $((\overline{u}, \pi) :: F, \overline{r}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_5} (F, \overline{ur}, \pi, E, \Uparrow)$**.**

  1. *Environment*: it follows from the *i.h.*.
  2. *Code, Stack, and Frame*: it follows from the *i.h.*.

- **Case** $(F, \overline{u}, \overline{r} :: \pi, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_6} ((\overline{u}, \pi) :: F, \overline{r}, \epsilon, E, \Downarrow)$**.**

  1. *Environment*: it follows from the *i.h.*.
  2. *Code, Stack, and Frame*: it follows from the *i.h.*. □

A.1.5 LO decoding invariant — proof of Lem. 3.67

For the invariant we need the following lemma.

Lemma A.15 (Compatible Pairs Decode to Non-Applicative Contexts)**.** *Let F_{w} be a weak frame, E_{w} a weak environment, and $F \propto E$ a compatible pair. Then $[\![F_{\mathrm{w}}]\!]$, $[\![E_{\mathrm{w}}]\!]$, and $[\![(F, E)]\!]$ are contexts that are not applicative,* i.e. *not of the form $\mathcal{C}\langle \mathtt{L}t \rangle$.*

Proof. The fact that $[\![F_{\mathrm{w}}]\!]$ and $[\![E_{\mathrm{w}}]\!]$ are not applicative is an immediate induction over their structure. For $[\![(F, E)]\!]$ we reason by induction on the compatibility of F and E. The base case $[\![(\epsilon, \epsilon)]\!] = \Box$ is evident. Inductive cases:

1. *Weak Extension, i.e.* $(F_{\mathrm{w}} :: F_{\mathrm{t}}) \propto (E_{\mathrm{w}} :: E_{\mathrm{t}})$ with $F_{\mathrm{t}} \propto E_{\mathrm{t}}$. By *i.h.* $[\![(F_{\mathrm{t}}, E_{\mathrm{t}})]\!]$ is not applicative and both $[\![F_{\mathrm{w}}]\!]$ and $[\![E_{\mathrm{w}}]\!]$ are not applicative. By definition, $[\![((F_{\mathrm{w}} :: F_{\mathrm{t}}), (E_{\mathrm{w}} :: E_{\mathrm{t}}))]\!] = [\![(F_{\mathrm{t}}, E_{\mathrm{t}})]\!]\langle [\![E_{\mathrm{w}}]\!] \langle [\![F_{\mathrm{w}}]\!] \rangle\rangle$, which is then not applicative.
2. *Abstraction, i.e.* $(x :: F) \propto (\rhd x :: E)$ with $F \propto E$. Immediate, as $[\![(F, E)]\!]\langle \lambda x.\Box \rangle$ is not applicative.

□

We can now prove that the decoding of the data-structures of a reachable state is a LO context.

Lemma A.16 (Full proof of Lem. 3.67—LO decoding invariant)**.** *Let* $S = \langle \varphi \mid F \mid \overline{s} \mid \pi \mid E \rangle$ *be a reachable state. Then* $[\![(F, E)]\!]$ *and* $\mathcal{C}_S$ *are* LO *contexts.*

Proof. We prove that $[\![(F, E)]\!]$ is a LO context, the fact that $\mathcal{C}_S$ is a LO contexts then easily follows, as $\mathcal{C}_S := [\![(F, E)]\!]\langle [\![\pi]\!] \rangle$.

The invariant trivially holds for an initial state $\Downarrow \mid \epsilon \mid \overline{t}_0 \mid \epsilon \mid \epsilon$. For a non-empty evaluation sequence we list the cases for the last transitions. We omit the cases for which the environment and the frame do not change (*i.e.* $\rightsquigarrow_{\Downarrow \mathtt{s}_1}, \rightsquigarrow_{\mathtt{e}}, \rightsquigarrow_{\Downarrow \mathtt{s}_3}$), as for them the statement follows from the *i.h.*.

- **Case** $(F, \lambda x.\overline{t}, \overline{s} :: \pi, E, \Downarrow) \rightsquigarrow_{\mathtt{m}} (F, \overline{t}, \pi, [x \backslash \overline{s}] :: E, \Downarrow)$**.** By *i.h.* $[\![(F, E)]\!]$ is LO. Let $F = F_{\mathrm{w}} :: F_{\mathrm{t}}$, so that $[\![(F, E)]\!] = [\![(F_{\mathrm{t}}, E)]\!]\langle [\![F_{\mathrm{w}}]\!] \rangle$. Note that, by the name invariant (Lem. 3.64), the eventual occurrences of x are all in $\overline{t}$ and so $x \notin \mathtt{fv}([\![F_{\mathrm{w}}]\!])$, and in particular $x \notin \mathtt{lfv}([\![F_{\mathrm{w}}]\!])$. Then, $[\![(F_{\mathrm{t}}, E)]\!]\langle [\![F_{\mathrm{w}}]\!][x \backslash \overline{s}] \rangle$ is LO: the conditions of Def. 3.5 are satisfied either because $[\![(F, E)]\!] = [\![(F_{\mathrm{t}}, E)]\!]\langle [\![F_{\mathrm{w}}]\!] \rangle$ is LO or because $x \notin \mathtt{lfv}([\![F_{\mathrm{w}}]\!])$.
- **Case** $(F, \lambda x.\overline{t}, \epsilon, E, \Downarrow) \rightsquigarrow_{\Downarrow \mathtt{s}_2} (x :: F, \overline{t}, \epsilon, \rhd x :: E, \Downarrow)$**.** By *i.h.* we have $[\![(F, E)]\!]$ is LO and by Lem. A.15 $[\![(F, E)]\!]$ is not applicative, so $[\![((x :: F), (\rhd x :: E))]\!] = [\![(F, E)]\!]\langle \lambda x.\Box \rangle$ is LO (it satisfies the conditions of Def. 3.5 because $[\![(F, E)]\!]$ does).
- **Case** $(x :: F, \overline{t}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_4} (F, \lambda x.\overline{t}, \epsilon, x \lhd :: E, \Uparrow)$**.** By the compatibility invariant (Lem. 3.64) $(x :: F) \propto E$, and by the factorization property of compatible pairs (Lem. 3.63) $E = E_{\mathrm{w}} :: \rhd x :: E'$. By definition

  $$[\![((x :: F), (E_{\mathrm{w}} :: \rhd x :: E_{\mathrm{t}}))]\!] = [\![(F, E_{\mathrm{t}})]\!]\langle \lambda x.[\![E_{\mathrm{w}}]\!] \rangle$$

  that by *i.h.* is LO. Now, $[\![(F, E_{\mathrm{t}})]\!]$ is LO, as it satisfies the conditions of Def. 3.5 because $[\![(F, E)]\!]$ does. We conclude by noticing that the compatible pair of the target state satisfies $[\![(F, (x \lhd :: E))]\!] = [\![(F, (x \lhd :: E_{\mathrm{w}} :: \rhd x :: E_{\mathrm{t}}))]\!] =_{Lem.\ 3.66} [\![(F, E_{\mathrm{t}})]\!]$.
- **Case** $((\overline{t}, \pi) :: F, \overline{s}, \epsilon, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_5} (F, \overline{t}\overline{s}, \pi, E, \Uparrow)$**.** By *i.h.* we have that $[\![(((\overline{t}, \pi) :: F), E)]\!]$ is LO and by *frame* part of the backtracking normal form invariant (Lem. 3.64) $\overline{t}$ is neutral. By definition, $[\![(((\overline{t}, \pi) :: F), E)]\!] = [\![(F, E)]\!]\langle [\![\pi]\!] \langle \overline{t}\Box \rangle\rangle$, Then, $[\![(F, E)]\!]$—being a prefix of $[\![(((\overline{t}, \pi) :: F), E)]\!]$—verifies the conditions of Def. 3.5 and is LO.
- **Case** $(F, \overline{t}, \overline{s} :: \pi, E, \Uparrow) \rightsquigarrow_{\Uparrow \mathtt{s}_6} ((\overline{t}, \pi) :: F, \overline{s}, \epsilon, E, \Downarrow)$**.** Note that

1. $[\![(F,E)]\!]$ is LO by *i.h.*,
2. $[\![(F,E)]\!]$ is not applicative by Lem. A.15,
3. $\mathtt{fv}(\overline{t}) \subseteq \Lambda(F)$ by the backtracking free variables invariant (Lem. 3.64).
4. $\overline{t}$ is a neutral term by the backtracking normal form invariant (Lem. 3.64), because the stack at the left-hand side is not empty.

Note that the third item guarantees that $x \notin \mathtt{fv}(\overline{t})$, and so in particular $x \notin \mathtt{lfv}(\overline{t})$, for any ES $[x\backslash\overline{u}]$ in E (and so in $[\![(F,E)]\!]$). Then $[\![(F,E)]\!]\langle[\![\pi]\!]\langle\overline{t}\Box\rangle\rangle$ is LO (because it verifies the conditions of Def. 3.5, by the listed points), that is to say $[\![(((\overline{t},\pi)::F),E)]\!]$ is LO.

□

A.2 Proofs of Chapter 4 – Foundations of Strong Call-by-Need

A.2.1 Technical tools

This subsection is devoted to bringing together various definitions and properties that are used as technical tools throughout the proofs of Chapter 4 (*Foundations of Strong Call-by-Need*). Most proofs in this subsection are ommited since they are straightforward.

Convention A.17. *In the proofs of Chapter 4, we adopt the following notational conventions:*

F^ϑ, F_1^ϑ, F_2^ϑ, *etc.*	*range over*	*evaluation contexts in* E_ϑ
$\mathtt{I}^\vartheta$, $\mathtt{I}_1^\vartheta$, $\mathtt{I}_2^\vartheta$, *etc.*	*range over*	*inert evaluation contexts in* $\mathsf{E}_\vartheta^\circ$
N^ϑ, N_1^ϑ, N_2^ϑ, *etc.*	*range over*	*strong normal forms in* $\mathtt{nf}\vartheta$
M^ϑ, M_1^ϑ, M_2^ϑ, *etc.*	*range over*	*strong structures in* S_ϑ

so rather than saying "t is of the form $\mathtt{C}\langle s\rangle$, where $\mathtt{C} \in \mathsf{E}_\vartheta$ and $s \in \mathsf{S}_{\vartheta'}$", we might say "$t$ is of the form $F^\vartheta\langle M^{\vartheta'}\rangle$".

Definition A.18 (Frozen variables)**.** The *frozen variables* $\mathsf{fz}^\vartheta(\mathtt{C})$ of a context $\mathtt{C}$ are the rigid bound variables that bind the hole of $\mathtt{C}$.

$$
\begin{array}{rcll}
\mathsf{fz}^\vartheta(\Box) & = & \vartheta & \\
\mathsf{fz}^\vartheta(\mathtt{C}t) & = & \mathsf{fz}^\vartheta(\mathtt{C}) & \\
\mathsf{fz}^\vartheta(t\mathtt{C}) & = & \mathsf{fz}^\vartheta(\mathtt{C}) & \\
\mathsf{fz}^\vartheta(\lambda x.\mathtt{C}) & = & \mathsf{fz}^{\vartheta\cup\{x\}}(\mathtt{C}) & \\
\mathsf{fz}^\vartheta(t[x\backslash\mathtt{C}]) & = & \mathsf{fz}^\vartheta(\mathtt{C}) & \\
\mathsf{fz}^\vartheta(\mathtt{C}[x\backslash t]) & = & \begin{cases} \mathsf{fz}^{\vartheta\cup\{x\}}(\mathtt{C}) & \text{if } t \text{ is a strong } \vartheta\text{-structure} \\ \mathsf{fz}^\vartheta(\mathtt{C}) & \text{otherwise} \end{cases} &
\end{array}
$$

Lemma A.19. $\mathsf{fz}^\vartheta(\mathtt{C}_1\langle\mathtt{C}_2\rangle) = \mathsf{fz}^{\mathsf{fz}^\vartheta(\mathtt{C}_1)}(\mathtt{C}_2)$.

Proof. By induction on $\mathtt{C}_1$. □

Lemma A.20 (Decomposition of evaluation contexts)**.** *If* $\mathtt{C}_1\langle\mathtt{C}_2\rangle$ *is an evaluation context, then* $\mathtt{C}_1$ *and* $\mathtt{C}_2$ *are evaluation contexts. More precisely, let* $\mathbb{X}^\vartheta$ *denote either the set* E_ϑ *or the set* $\mathsf{E}^\circ_\vartheta$*. If* $\mathtt{C}_1\langle\mathtt{C}_2\rangle \in \mathbb{X}^\vartheta$ *then* $\mathtt{C}_1 \in \mathbb{X}^\vartheta$ *and* $\mathtt{C}_2 \in \mathsf{E}_{\vartheta'}$*, where* $\vartheta' = \mathsf{fz}^\vartheta(\mathtt{C}_1)$*.*

Proof. By induction on the formation rules for $\mathtt{C}_1\langle\mathtt{C}_2\rangle$ as a context in $\mathbb{X}^\vartheta$. □

Lemma A.21. *Inert evaluation contexts do not go below answers, and evaluation contexts do not go below* db*-redexes. More precisely:*

1. *If* $(\lambda x.t)\mathtt{L} = \mathtt{I}^\vartheta\langle s\rangle$ *where* $\mathtt{I}^\vartheta \in \mathsf{E}^\circ_\vartheta$ *is an inert evaluation context, then* $\mathtt{I}^\vartheta$ *is a substitution context,* i.e. $\mathtt{L}$ *can be split as* $\mathtt{L} = \mathtt{L}_1\mathtt{L}_2$ *such that* $\mathtt{I}^\vartheta = \mathtt{L}_2$*.*
2. *It cannot be the case that* $\mathtt{vL} = \mathtt{I}^\vartheta\langle\Delta\rangle$ *if* Δ *is a* db*-redex or a variable.*
3. *Let* $t = (\lambda x.s)\mathtt{L}\,u$ *be the redex pattern of a* db*-step. Suppose that* $t = F^\vartheta\langle t'\rangle$ *for some context* $F^\vartheta \in \mathsf{E}_\vartheta$*, some set of variables* ϑ*, and some term* t'*. Then* $\mathtt{L}$ *can be split as* $\mathtt{L} = \mathtt{L}_1\mathtt{L}_2$ *such that* $F^\vartheta = \mathtt{L}_2\,u$*.*

Proof. The first item is by induction on the length of the substitution context $\mathtt{L}$. The second item is an immediate consequence of the first. For the third item consider the two possible formation rules for F^ϑ as a context in E_ϑ. Rule EAppL is a consequence of the first item. Rule EAppRStr is impossible. □

Lemma A.22 (Answers are stable by reduction)**.** *Let* $(\lambda x.t)\mathtt{L} \to_{\mathtt{sh}\backslash\mathtt{gc}} s$*. Then* s *is an answer.*

Proof. By case analysis on the kind of step (db or lsv) and its position inside $(\lambda x.t)\mathtt{L}$. The interesting case is when it is a lsv step that contracts one of the substitutions in $\mathtt{L}$. If the variable contracted by the lsv step is inside t, the step is of the form $(\lambda x.\mathtt{C}\langle\!\langle y\rangle\!\rangle)\mathtt{L}_1[y\backslash\mathtt{vL}']\mathtt{L}_2 \to (\lambda x.\mathtt{C}\langle\mathtt{v}\rangle)\mathtt{L}_1[y\backslash\mathtt{v}]\mathtt{L}'\mathtt{L}_2$ and s is an answer. If the variable contracted by the lsv step is inside one of the substitutions in $\mathtt{L}$, the step is of the form $(\lambda x.t)\mathtt{L}_1[y\backslash\mathtt{C}\langle\!\langle z\rangle\!\rangle]\mathtt{L}_2[y\backslash\mathtt{vL}']\mathtt{L}_3 \to (\lambda x.t)\mathtt{L}_1[y\backslash\mathtt{C}\langle\mathtt{v}\rangle]\mathtt{L}_2[y\backslash\mathtt{v}]\mathtt{L}'\mathtt{L}_3$ and s is an answer. □

Lemma A.23 (Non-inert evaluation contexts are answers)**.** *Let* $\mathtt{C} \in \mathsf{E}_\vartheta\backslash\mathsf{E}^\circ_\vartheta$*,* i.e. *an evaluation context that is not inert. Then* $\mathtt{C}$ *has the form of an answer,* i.e. *it is either of the form* $(\lambda x.\mathtt{C}')\mathtt{L}$*, or of the form* $(\lambda x.t)\mathtt{L}_1[y\backslash\mathtt{C}']\mathtt{L}_2$*.*

Proof. By induction on the derivation that $\mathtt{C} \in \mathsf{E}_\vartheta$. □

Lemma A.24 (Weakening ϑ)**.** *The set* ϑ *can be weakened (*i.e. *extended) both for normal forms and for evaluation contexts. More precisely, let* $\vartheta \subseteq \vartheta'$*. Then:*

1. $\mathtt{nf}\vartheta \subseteq \mathtt{nf}\vartheta'$ *and* $\mathsf{S}_\vartheta \subseteq \mathsf{S}_{\vartheta'}$*.*
2. $\mathsf{E}_\vartheta \subseteq \mathsf{E}_{\vartheta'}$ *and* $\mathsf{E}^\circ_\vartheta \subseteq \mathsf{E}^\circ_{\vartheta'}$*.*

Proof. The first item is straightforward by induction on the derivation of a normal form. For the second item, let $\mathbb{X}^{\vartheta}$ stand for either E_{ϑ} or $\mathsf{E}^{\circ}_{\vartheta}$ and let us show that $\mathtt{C} \in \mathbb{X}^{\vartheta}$ implies $\mathtt{C} \in \mathbb{X}^{\vartheta'}$ by induction on the formation rules for $\mathtt{C}$ as a context in E_{ϑ} or $\mathsf{E}^{\circ}_{\vartheta}$.

Most cases are straightforward by applying the *i.h.*. The only subtle case is when $\mathtt{C}$ is built by appending a non-structural substitution (rule ESubLNonStr), *i.e.* when $\mathtt{C} = \mathtt{C}_1[x\backslash t] \in \mathbb{X}^{\vartheta}$ with $\mathtt{C}_1 \in \mathbb{X}^{\vartheta}$, $t \notin \mathsf{S}_{\vartheta}$ and $x \notin \vartheta$. Then we consider two subcases, depending on whether t is a strong ϑ'-structure:

1. If $t \in \mathsf{S}_{\vartheta'}$ Note that $\vartheta \subseteq \vartheta' \subseteq \vartheta' \cup \{x\}$ so by *i.h.* we have that $\mathtt{C}_1 \in \mathbb{X}^{\vartheta' \cup \{x\}}$. By applying the formation rule for generalized ϑ'-evaluation contexts using a *structural* substitution we conclude that $\mathtt{C}_1[x\backslash t] \in \mathbb{X}^{\vartheta'}$, as wanted.

2. If $t \notin \mathsf{S}_{\vartheta'}$ Note that $x \notin \vartheta'$ by the variable convention (*i.e.* ϑ' is a set of free variables, but x is bound by a substitution). By *i.h.* we have that $\mathtt{C}_1 \in \mathbb{X}^{\vartheta'}$. By applying the formation rule for generalized ϑ'-evaluation contexts using a non-structural substitution (ESubLNonStr) we conclude that $\mathtt{C}_1[x\backslash t] \in \mathbb{X}^{\vartheta'}$, as wanted.

□

Lemma A.25 (Strengthening for normal forms)**.** *Let t be a $(\vartheta \cup \{x\})$-normal form (resp. $(\vartheta \cup \{x\})$-structure).*

1. *If $x \notin \mathsf{ngv}(t)$, then t is a ϑ-normal form (resp. ϑ-structure).*

2. *If $x \in \mathsf{ngv}(t)$, then t can be written as $t = \mathtt{C}\langle\langle x \rangle\rangle$ where $\mathtt{C}$ is a (resp. inert) ϑ-evaluation context.*

Proof. The first item is by induction on the derivation that $t \in \mathtt{nf}\vartheta \cup \{x\}$. The second item is by induction on the derivation that $t \in \mathtt{nf}\vartheta \cup \{x\}$, using the first item. □

Lemma A.26 (Evaluation contexts are closed by adding substitutions)**.** *If $\mathtt{C}$ is a (resp. inert) ϑ-evaluation context, then $\mathtt{C}[x\backslash t]$ is a (resp. inert) ϑ-evaluation context, provided that $x \notin \vartheta$.*

Proof. By case analysis on whether t is a strong ϑ-structure, and the weakening lemma (Lem. A.24). □

Lemma A.27 (Inversion for normal forms)**.** *If $t\mathtt{L}$ is a ϑ-normal form (resp. ϑ-structure) then t is a $\mathsf{fz}^{\vartheta}(\mathtt{L})$-normal form (resp. $\mathsf{fz}^{\vartheta}(\mathtt{L})$-structure).*

Proof. By induction on $\mathtt{L}$. □

A.2.2 Characterization of ϑ-normal forms — proof of Lem. 4.15

In this subsection we prove Lem. 4.15, which states that ϑ-normal forms as defined in Def. 4.11 are indeed the normal forms of the strategy $\overset{\vartheta}{\rightsquigarrow}$.

Lemma A.28. *If $t \in \mathtt{nf}\vartheta$ and t is not an answer, then $t \in \mathsf{S}_{\vartheta}$.*

Proof. By induction on $t \in \mathtt{nf}\vartheta$. □

Lemma A.29. *Variables below evaluation contexts are reachable. More precisely, if* $t = \mathtt{C}\langle\!\langle x \rangle\!\rangle$ *where* $\mathtt{C} \in \mathsf{E}_\vartheta$ *(resp.* $t = \mathsf{E}^\circ_\vartheta$*), then* $x \in \mathsf{ngv}(t)$.

Proof. By induction on the derivation that $\mathtt{C} \in \mathsf{E}_\vartheta$. □

Lemma A.30 (Full proof of Lem. 4.15—Characterization of ϑ-normal forms)**.** *The following sets are equal:*

- *The set of* ϑ*-normal forms* $\mathtt{nf}\vartheta$ (cf. *Def. 4.11).*
- *The set of normal forms of the strong call-by-need strategy* $\overset{\vartheta}{\rightsquigarrow}$.

Proof. ($\supseteq$) Let $t \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$. Then by induction on t, using Lem. A.28 and Lem. A.25 we can check that $t \in \mathtt{nf}\vartheta$. ($\subseteq$) We prove the following more general property: $\mathtt{nf}\vartheta \cup \mathsf{S}_\vartheta \subseteq \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$, by taking a term $t \in \mathtt{nf}\vartheta \cup \mathsf{S}_\vartheta$ and proceeding by induction on the derivation that $t \in \mathtt{nf}\vartheta \cup \mathsf{S}_\vartheta$. The interesting cases are N-APP, NFSUB, and NFSUBG:

1. N-APP: $t = t_1 t_2$ with $t_1 \in \mathsf{S}_\vartheta$ and $t_2 \in \mathtt{nf}\vartheta$. By *i.h.*, $t_1 \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ and $t_2 \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$. Since $t_1 \in \mathsf{S}_\vartheta$, then t_1 is not an answer. Therefore $t \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$.
2. NFSUB: $t = t_1[x\backslash t_2]$, where $t_1 \in \mathbb{X}^{\vartheta\cup\{x\}}$ and $x \in \mathsf{ngv}(t_1)$ and $t_2 \in \mathsf{S}_\vartheta$. By *i.h.*, $t_1 \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ and $t_2 \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$. Finally, reduction at the root is not possible since t_2 is a structure, hence not an answer.
3. NFSUBG: $t = t_1[x\backslash t_2]$, where $t_1 \in \mathbb{X}^\vartheta$ and $x \notin \mathsf{ngv}(t)$. By *i.h.*, $t_1 \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$. By Lem. A.29, reduction at the root is not possible. By the same lemma, the focus of reduction cannot be any subterm in t_2. Thus $t \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$.

□

A.2.3 Unique decomposition — proof of Lem. 4.17

Our aim is to show that whenever $\mathtt{C}_1\langle r_1\rangle = \mathtt{C}_2\langle r_2\rangle$, where $\mathtt{C}_1$ and $\mathtt{C}_2$ are evaluation contexts over ϑ, and r_1 and r_2 are reducible subterms, then $\mathtt{C}_1 = \mathtt{C}_2$ and $r_1 = r_2$. A technical stumbling block is that it is not possible to reason inductively: if a term is of the form $\mathtt{C}\langle r\rangle[x\backslash t]$ where r is a reducible subterm, then in the subterm $\mathtt{C}\langle r\rangle$ it is not necessarily the case that r is a reducible subterm. For instance the underlined occurrence of x is a reducible subterm in $(\underline{x}x)[x\backslash\lambda y.y]$ but not in $\underline{x}x$. The way out of this difficulty is generalizing the notion of reducible subterm to that of **reduction place**. A reduction place is essentially a reducible subterm or the free ocurrence of a variable. Reasoning inductively will be possible using reduction places, rather than reducible subterms, since if r is a reduction place in $\mathtt{C}\langle r\rangle[x\backslash t]$ then r is a reduction place in $\mathtt{C}\langle r\rangle$. More precisely:

Definition A.31 (Reduction place)**.** In a term $F^\vartheta\langle t\rangle$, the subterm t is said to be a F^ϑ*-reduction place* if any of the following hold:

1. t is the redex pattern of a beta-step, *i.e.* $t = (\lambda x.s)\mathtt{L}\,u$;

2. t is the variable contracted by an ls-step, *i.e.* $t = x$ and $F^\vartheta = \mathtt{C}\langle \mathtt{C}'[x\backslash \mathtt{vL}]\rangle$;

3. t is a free variable (not bound by F^ϑ) such that $x \notin \mathsf{fz}^\vartheta(F^\vartheta)$.

Lemma A.32 (Reduction places are stable by trimming a context down)**.** *Let $F_1^\vartheta\langle F_2^{\vartheta'}\rangle \in \mathsf{E}_\vartheta$, and let t be a $F_1^\vartheta\langle F_2^{\vartheta'}\rangle$-reduction place. Then t is a $F_2^{\vartheta'}$-reduction place.*

Proof. Let us consider the three cases in Def. A.31 for the fact that t is a $F_1^\vartheta\langle F_2^{\vartheta'}\rangle$-reduction place:

1. If t is the redex pattern of a beta-step Then t is trivially a $F_2^{\vartheta'}$-reduction place.

2. If t is the variable contracted by an ls-step That is, $t = x$ and x is bound to an answer $\mathtt{vL}$. There are two cases, depending on whether x is bound by the external context F_1^ϑ or by the internal context $F_2^{\vartheta'}$:

   2.1 *If x is bound by F_1^ϑ.*
   Then x is not bound by $F_2^{\vartheta'}$. To show that $t = x$ is indeed a $F_2^{\vartheta'}$-reduction place, it suffices to show that $x \notin \mathsf{fz}^{\vartheta'}(F_2^{\vartheta'})$. By Lem. A.19 we know that $\mathsf{fz}^{\vartheta'}(F_2^{\vartheta'}) = \mathsf{fz}^\vartheta(F_1^\vartheta\langle F_2^{\vartheta'}\rangle)$. Since x is bound by F_1^ϑ, let us write $F_1^\vartheta = F_{11}^\vartheta\langle F_{12}^{\vartheta''}[x\backslash \mathtt{vL}]\rangle$. We know that $x \notin \vartheta$ by Barendregt's convention. By applying Lem. A.19 again we obtain that $\mathsf{fz}^\vartheta(F_1^\vartheta\langle F_2^{\vartheta'}\rangle) = \mathsf{fz}^{\vartheta'''}(F_{12}^{\vartheta''}\langle F_2^{\vartheta'}\rangle[x\backslash \mathtt{vL}])$, where $\vartheta''' = \mathsf{fz}^\vartheta(F_{11}^\vartheta)$. Note that x is not bound by F_{11}^ϑ, so $x \notin \vartheta'''$.
   Now note that vL is an answer but not a structure, so $\vartheta''' = \vartheta''$ and $\mathsf{fz}^{\vartheta'''}(F_{12}^{\vartheta''}\langle F_2^{\vartheta'}\rangle[x\backslash \mathtt{vL}]) = \mathsf{fz}^{\vartheta''}(F_{12}^{\vartheta''}\langle F_2^{\vartheta'}\rangle)$. Note also that since $x \notin \vartheta'''$ and x is not bound by $F_{12}^{\vartheta''}\langle F_2^{\vartheta'}\rangle$ we know that $x \notin \mathsf{fz}^{\vartheta''}(F_{12}^{\vartheta''}\langle F_2^{\vartheta'}\rangle)$. Finally, we may apply Lem. A.19 once more to conclude that $x \notin \mathsf{fz}^{\vartheta''}(F_{12}^{\vartheta''}\langle F_2^{\vartheta'}\rangle) = \mathsf{fz}^{\vartheta'}(F_2^{\vartheta'})$, by which we conclude that x is a $F_2^{\vartheta'}$-reduction place, as required.

   2.2 *If x is bound by $F_2^{\vartheta'}$.*
   Then $t = x$ is trivially a $F_2^{\vartheta'}$-reduction place, as it is the variable contracted by an ls-step.

3. If t is a free variable x such that $x \notin \mathsf{fz}^\vartheta(F_1^\vartheta\langle F_2^{\vartheta'}\rangle)$ As x is not bound by $F_1^\vartheta\langle F_2^{\vartheta'}\rangle$, we have that x is also not bound by $F_2^{\vartheta'}$. Moreover, by Lem. A.19 we have that $\mathsf{fz}^\vartheta(F_1^\vartheta\langle F_2^{\vartheta'}\rangle) = \mathsf{fz}^{\mathsf{fz}^\vartheta(F_1^\vartheta)}(F_2^{\vartheta'})$. Since the composition $F_1^\vartheta\langle F_2^{\vartheta'}\rangle$ is a context in E_ϑ, by the decomposition of evaluation contexts (Lem. A.20) we know that $\mathsf{fz}^\vartheta(F_1^\vartheta) = \vartheta'$, so we conclude that $x \notin \mathsf{fz}^\vartheta(F_1^\vartheta\langle F_2^{\vartheta'}\rangle) = \mathsf{fz}^{\vartheta'}(F_2^{\vartheta'})$, so t is a $F_2^{\vartheta'}$-reduction place, as required.

□

Lemma A.33 (Strong normal forms have no reduction places under an evaluation context)**.** *Let $N^\vartheta \in \mathtt{nf}\vartheta$ be a strong normal form. Then N^ϑ cannot be written as $F^\vartheta\langle t\rangle$ such that $F^\vartheta \in \mathsf{E}_\vartheta$ is a generalized evaluation context and t is an F^ϑ-reduction place.*

Proof. Suppose that $N^\vartheta = F^\vartheta\langle t\rangle$, where t is a F^ϑ-reduction place. Let us check that this is impossible by induction on F^ϑ.

1. EBox, *i.e.* $F^\vartheta = \Box$ Then N^ϑ must be a $\Box$-reduction place, for $\Box$ as a context in E_ϑ. Let us consider the three cases of the definition of $\Box$-reduction place:

   1.1 *If N^ϑ is the redex pattern of a beta-step.*
   Then $N^\vartheta = M^\vartheta\, N_1^\vartheta$ with M^ϑ an answer. But strong structures are not answers, so this case is impossible.

   1.2 *If N^ϑ is the variable x contracted by an ls-step.*
   Impossible, since x is free.

   1.3 *If N^ϑ is a free variable x such that $x \notin \mathsf{fz}^\vartheta(\Box)$.*
   Impossible, since $x \notin \vartheta$, but a variable x is a strong normal form in $\mathtt{nf}\,\vartheta$ if and only if $x \in \vartheta$.

2. EAppL, *i.e.* $F^\vartheta = \mathtt{I}^\vartheta\, s$ Then since $F^\vartheta\langle t\rangle$ is a ϑ-normal form, the subterm $\mathtt{I}^\vartheta\langle t\rangle$ must also be a ϑ-normal form. Moreover t is a a $\mathtt{I}^\vartheta$-reduction place by Lem. A.32. By *i.h.* we conclude that this is impossible.

3. ESubLNonStr, ESubLStr, ESubsR, EAppRStr, ELam Similar to EAppL.

□

Lemma A.34 (Full proof of Lem. 4.17—Unique decomposition)**.** *If $F_1^\vartheta\langle t_1\rangle = F_2^\vartheta\langle t_2\rangle$ such that t_i is a F_i^ϑ-reduction place for $i \in \{1,2\}$, then $F_1^\vartheta = F_2^\vartheta$ and $t_1 = t_2$.*

Proof. By induction on the derivation of F_1^ϑ as a context in E_ϑ:

1. EBox, $F_1^\vartheta = \Box$ By cases on the definition that t_1 is a F_1^ϑ-reduction place:

   1.1 If t_1 is the redex pattern of a beta-step Suppose that F_2^ϑ were not empty. Let $t_1 = (\lambda x.s)\mathtt{L}\, u$. Then $F_2^\vartheta\langle t_2\rangle = (\lambda x.s)\mathtt{L}\, u$. By Lem. A.21 we have that $\mathtt{L}$ can be split as $\mathtt{L} = \mathtt{L}_1\mathtt{L}_2$ such that $F_2^\vartheta = \mathtt{L}_2\, u$. This means that $t_2 = (\lambda x.s)\mathtt{L}_1$, so t_2 cannot be a F_2^ϑ-reduction place, as it is neither an application nor a variable. Hence this case is impossible.

   1.2 If t_1 is a variable x contracted by an ls-step Impossible, as there is no substitution binding x.

   1.3 If t_1 is a free variable x such that $x \notin \mathsf{fz}^\vartheta(F_1^\vartheta) = \vartheta$ Immediate, as $F_2^\vartheta = \Box$ so $t_2 = x \notin \vartheta = \mathsf{fz}^\vartheta(F_2^\vartheta)$.

2. EAppL, *i.e.* $F_1^\vartheta = \mathtt{I}_1^\vartheta\, s$ Then $\mathtt{I}_1^\vartheta\langle t_1\rangle\, s = F_2^\vartheta\langle t_2\rangle$. By case analysis on the formation rules for F_2^ϑ. Note that F_2^ϑ cannot be empty, since the symmetric situation has already been considered.

   2.1 EAppL, *i.e.* $F_2^\vartheta = \mathtt{I}_2^\vartheta\, s$ Then $\mathtt{I}_1^\vartheta\langle t_1\rangle = \mathtt{I}_2^\vartheta\langle t_2\rangle$. The contexts $\mathtt{I}_1^\vartheta$ and $\mathtt{I}_2^\vartheta$ are both in $\mathsf{E}_\vartheta^\circ$ and hence also in E_ϑ, and each t_i is a $\mathtt{I}_i^\vartheta$-reduction place (by Lem. A.32), so by *i.h.* we have $(\mathtt{I}_1^\vartheta, t_1) = (\mathtt{I}_2^\vartheta, t_2)$.

2.2 EAppRStr, *i.e.* $F_2^\vartheta = M^\vartheta\, F_{21}^\vartheta$ This implies that $M^\vartheta = \mathtt{I}_1^\vartheta\langle t_1\rangle$ where t_1 is a $\mathtt{I}_1^\vartheta$-reduction place. A strong normal form such as M^ϑ cannot have a reduction place such as t_1 under an evaluation context such as $\mathtt{I}_1^\vartheta$. This last fact is a direct application of Lem. A.33.

3. ESubLNonStr, *i.e.* $F_1^\vartheta = F_{11}^\vartheta[x\backslash s]$ with $s \notin \mathsf{S}_\vartheta$ and $x \notin \vartheta$ By case analysis on the formation rules for F_2^ϑ. Note that F_2^ϑ cannot be empty, since the symmetric situation has already been considered.

   3.1 ESubLNonStr, *i.e.* $F_2^\vartheta = F_{21}^\vartheta[x\backslash s]$ Note that each t_i is a F_{i1}^ϑ-reduction place by Lem. A.32. By the *i.h.* on F_1^ϑ we have that $(F_1^\vartheta, t_1) = (F_2^\vartheta, t_2)$, so we conclude.

   3.2 ESubLStr, *i.e.* $F_2^\vartheta = F_{21}^{\vartheta\cup\{x\}}[x\backslash M^\vartheta]$ This case is impossible, as the formation rule for F_1^ϑ implies that $s \notin \mathsf{S}_\vartheta$, while the formation rule for F_2^ϑ implies that $s = M^\vartheta \in \mathsf{S}_\vartheta$.

   3.3 ESubsR, *i.e.* $F_2^\vartheta = F_{21}^\vartheta\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{I}^\vartheta]$ We claim that this case is impossible. Note that we have that $F_{11}^\vartheta\langle t_1\rangle = F_{21}^\vartheta\langle\!\langle x\rangle\!\rangle$, where t_1 is a F_{11}^ϑ-reduction place by virtue of Lem. A.32. Moreover $x \notin \vartheta$, and x is not bound by F_{21}^ϑ (by Barendregt's convention), so $x \notin \mathsf{fz}^\vartheta(F_{21}^\vartheta)$; these conditions imply that x is a F_{21}^ϑ-reduction place. This allows us to apply the *i.h.*, obtaining $(F_{11}^\vartheta, x) = (F_{21}^\vartheta, t_1)$. Since $t_1 = x$ is a F_1^ϑ-reduction place by hypothesis, and x is bound by F_1^ϑ, we conclude that it must be involved in an ls-step. This implies that the substitution $[x\backslash s]$ contains an answer, that is, $s = \mathtt{vL}$. But from the formation rule of F_2^ϑ, we also know that $s = \mathtt{I}^\vartheta\langle t_2\rangle$. So the situation is such that $\mathtt{I}^\vartheta\langle t_2\rangle = \mathtt{vL}$. By the fact that inert evaluation contexts such as $\mathtt{I}^\vartheta$ do not go below answers (Lem. A.21) we conclude that t_2 must be of the form $\mathtt{vL}_1$. This is a contradiction, as t_2 is a F_2^ϑ-reduction place, which means that it must be either an application or a variable.

4. ESubLStr, *i.e.* $F_1^\vartheta = F_{11}^{\vartheta\cup\{x\}}[x\backslash M^\vartheta]$ By case analysis on the formation rules for F_2^ϑ. Note that F_2^ϑ cannot be empty, nor built using ESubLNonStr or ESubLStr, since the symmetric situations have already been considered.

   4.1 ESubLStr, *i.e.* $F_2^\vartheta = F_{21}^{\vartheta\cup\{x\}}[x\backslash M^\vartheta]$ Then each t_i is a $F_{i1}^{\vartheta\cup\{x\}}$-reduction place as a consequence of Lem. A.32, so we may apply the *i.h.* to conclude $(F_{11}^{\vartheta\cup\{x\}}, t_1) = (F_{21}^{\vartheta\cup\{x\}}, t_2)$, as required.

   4.2 ESubsR, *i.e.* $F_2^\vartheta = F_{21}^\vartheta\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{I}^\vartheta]$ Then we have that $M^\vartheta = \mathtt{I}^\vartheta\langle t_2\rangle$. Note that t_2 is a $\mathtt{I}^\vartheta$-reduction place by Lem. A.32. This is impossible since M^ϑ is a strong normal form, and it might not have a reduction place under an evaluation context (Lem. A.33).

5. ESubsR, *i.e.* $F_1^\vartheta = F_{11}^\vartheta\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{I}_1^\vartheta]$ By case analysis on the formation rules for F_2^ϑ. Note that F_2^ϑ cannot be empty, nor built using ESubLNonStr, ESubLNonStr, or ESubLStr, since the symmetric situations have already been considered. The only remaining possiblity is that F_2^ϑ is built using ESubsR, *i.e.* $F_2^\vartheta = F_{21}^\vartheta\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{I}_2^\vartheta]$. Then each t_i is a

$\mathtt{I}_i^\vartheta$-reduction place, as a consequence of Lem. A.32. By applying the *i.h.* we obtain that $(\mathtt{I}_1^\vartheta, t_1) = (\mathtt{I}_2^\vartheta, t_2)$, as required.

6. EAppRStr, *i.e.* $F_1^\vartheta = M_1^\vartheta\, F_{11}^\vartheta$ By case analysis on the formation rules for F_2^ϑ. Note that F_2^ϑ cannot be empty, nor built using EAppL, since the symmetric situations have already been considered. The only remaining possiblity is that F_2^ϑ is built using EAppRStr, *i.e.* $F_2^\vartheta = M_2^\vartheta\, F_{21}^\vartheta$. Then each t_i is a F_{i1}^ϑ-reduction place, as a consequence of Lem. A.32. By applying the *i.h.* we conclude that $(F_{11}^\vartheta, t_1) = (F_{21}^\vartheta, t_2)$, as required.

7. ELam, *i.e.* $F_1^\vartheta = \lambda x.F_{11}^{\vartheta\cup\{x\}}$ Then F_2^ϑ cannot be empty (the symmetric situation was already considered), so F_2^ϑ must be of the form $\lambda x.F_{21}^{\vartheta\cup\{x\}}$. By Lem. A.32 we know that each t_i must be a F_{i1}^ϑ-reduction place, so we may apply the *i.h.* to conclude that $(F_{11}^\vartheta, t_1) = (F_{21}^\vartheta, t_2)$, as required.

□

A.2.4 Conservativity — proof of Thm. 4.23

In this section we give a proof of Thm. 4.23, which states that our strong call-by-need strategy is a conservative extension of weak call-by-need. The proofs developed in this section rely on an alternative characterization of weak normal forms. The set of weak normal forms is captured by $\mathsf{WNF}_\vartheta ::= \mathtt{vL} \mid \mathtt{E}\langle\!\langle x \rangle\!\rangle$ with $x \in \vartheta$. The alternative characterization presented below is convenient for carrying out the proofs.

Definition A.35 (Head reachable variables)**.** The set of *head reachable variables* of a term is defined as follows. Note that $\mathsf{hrv}(t) \subseteq \mathtt{fv}(t)$.

$$\begin{aligned}
\mathsf{hrv}(x) &\stackrel{\text{def}}{=} \{x\} \\
\mathsf{hrv}(ts) &\stackrel{\text{def}}{=} \mathsf{hrv}(t) \\
\mathsf{hrv}(\lambda x.t) &\stackrel{\text{def}}{=} \varnothing \\
\mathsf{hrv}(t[x\backslash s]) &\stackrel{\text{def}}{=} (\mathsf{hrv}(t)\backslash x) \cup \begin{cases} \mathsf{hrv}(s) & \text{if } x \in \mathsf{hrv}(t) \\ \varnothing & \text{otherwise} \end{cases}
\end{aligned}$$

The set of *ϑ-weak normal forms* ($\mathsf{N}_\vartheta^\mathsf{w}$), is defined below, mutually inductively with the set of *ϑ-weak structures* ($\mathsf{S}_\vartheta^\mathsf{w}$). Here, $\mathbb{X}^\vartheta$ stands for either the set $\mathsf{N}_\vartheta^\mathsf{w}$ or the set $\mathsf{S}_\vartheta^\mathsf{w}$:

$$\frac{x \in \vartheta}{x \in \mathsf{S}_\vartheta^\mathsf{w}}\ \textsc{n}^\mathsf{w}\textsc{-var} \qquad \frac{t \in \mathsf{S}_\vartheta^\mathsf{w}}{ts \in \mathsf{S}_\vartheta^\mathsf{w}}\ \textsc{n}^\mathsf{w}\textsc{-app} \qquad \frac{t \in \mathsf{S}_\vartheta^\mathsf{w}}{t \in \mathsf{N}_\vartheta^\mathsf{w}}\ \textsc{n}^\mathsf{w}\textsc{-incl} \qquad \frac{}{\lambda x.t \in \mathsf{N}_\vartheta^\mathsf{w}}\ \textsc{n}^\mathsf{w}\textsc{-lam}$$

$$\frac{t \in \mathbb{X}^{\vartheta\cup\{x\}} \quad x \in \mathsf{hrv}(t) \quad s \in \mathsf{S}_\vartheta^\mathsf{w}}{t[x\backslash s] \in \mathbb{X}^\vartheta}\ \textsc{n}^\mathsf{w}\textsc{-sub}^\bullet \qquad \frac{t \in \mathbb{X}^\vartheta \quad x \notin \mathsf{hrv}(t)}{t[x\backslash s] \in \mathbb{X}^\vartheta}\ \textsc{n}^\mathsf{w}\textsc{-sub}^\circ$$

Lemma A.36. $\mathsf{N}_\vartheta^\mathsf{w} = \mathsf{WNF}_\vartheta$

Proof. Straightforward by induction on the derivations. □

Lemma A.37. *Let $t \in \mathsf{WNF}_\vartheta$ or $t \in \mathsf{S}_\vartheta^\mathsf{w}$. Then $x \in \mathsf{hrv}(t)$ implies $x \in \vartheta$.*

Proof. By simultaneous induction on the derivation that $t \in \mathsf{WNF}_\vartheta$ or $t \in \mathsf{S}^{\mathsf{w}}_\vartheta$. The cases N^{w}-VAR, N^{w}-INCL, and N^{w}-LAM are immediate.

1. N^{w}-APP: $t = t_1\, t_2$ with $t_1 \in \mathsf{S}^{\mathsf{w}}_\vartheta$. Then $x \in \mathsf{hrv}(t_1)$ and we conclude by *i.h.*.
2. N^{w}-SUB$^\bullet$: $t = t_1[y\backslash t_2]$ and $t_1 \in \mathbb{X}^{\vartheta\cup\{x\}}$ and $x \in \mathsf{hrv}(t_1)$ and $t_2 \in \mathsf{S}^{\mathsf{w}}_\vartheta$. Recall that:

$$\mathsf{hrv}(t_1[y\backslash t_2]) \quad \stackrel{\text{def}}{=} \quad (\mathsf{hrv}(t_1)\backslash y) \cup \begin{cases} \mathsf{hrv}(t_2) & \text{if } y \in \mathsf{hrv}(t_1) \\ \varnothing & \text{otherwise} \end{cases}$$

   If $x \in \mathsf{hrv}(t_1[y\backslash t_2])$, there are two cases. Either $x \in \mathsf{hrv}(t_2)$ and we conclude from the *i.h.*. Otherwise, $x \in \mathsf{hrv}(t_1)\backslash y$. Then $x \neq y$ and we also conclude from the *i.h.* too.

3. N^{w}-SUB$^\circ$: $t = t_1[y\backslash t_2]$ with $t_1 \in \mathbb{X}^\vartheta$ and $y \notin \mathsf{hrv}(t_1)$ Therefore, $\mathsf{hrv}(t_1[y\backslash t]) = \mathsf{hrv}(t_1)\backslash y$. Suppose $x \in \mathsf{hrv}(t_1)\backslash y$. Then $x \neq y$ and $x \in \mathsf{hrv}(t_2)$ and we conclude using the *i.h.* again.

□

Lemma A.38. *If $s \in \mathsf{S}^{\mathsf{w}}_\vartheta$ is a weak structure then $\mathsf{hrv}(s)$ is a singleton.*

Proof. By induction on the derivation that $s \in \mathsf{S}^{\mathsf{w}}_\vartheta$. The cases N^{w}-VAR and N^{w}-APP are immediate.

1. N^{w}-SUB$^\circ$: Then $t = t_1[x\backslash t_2]$ with $x \notin \mathsf{hrv}(t_1)$ and $t_2 \in \mathsf{S}^{\mathsf{w}}_\vartheta$. Thus $\mathsf{hrv}(t) = \mathsf{hrv}(t_1)\backslash x = \mathsf{hrv}(t_1)$ and the result follows from the *i.h.* on t_1.
2. N^{w}-SUB$^\bullet$: Then $t = t_1[x\backslash t_2]$ with $t_1 \in \mathsf{S}^{\mathsf{w}}_{\vartheta\cup\{x\}}$ and $x \in \mathsf{hrv}(t_1)$ and $t_2 \in \mathsf{S}^{\mathsf{w}}_\vartheta$. By the *i.h.* $\mathsf{hrv}(t_1) = \{z\}$, for some variable z. We consider two cases:
   2.1 If $x \in \mathsf{hrv}(t_1)$. Then $z = x$ and $\mathsf{hrv}(t_1[x\backslash t_2]) = \mathsf{hrv}(t_1)\backslash x \cup \mathsf{hrv}(t_2) = \mathsf{hrv}(t_2)$. The result follows from the *i.h.* on t_2.
   2.2 If $x \notin \mathsf{hrv}(t_1)$. Then $\mathsf{hrv}(t_1[x\backslash t_2]) = \mathsf{hrv}(t_1)\backslash x = \mathsf{hrv}(t_1) = \{z\}$.

□

Lemma A.39. *Let $t \in \mathsf{S}^{\mathsf{w}}_\vartheta$. If $x \in \mathsf{hrv}(t)$, then there is a weak evaluation context $\mathtt{E} \in \mathsf{WCtx}$ such that $t = \mathtt{E}\langle\!\langle x \rangle\!\rangle$.*

Proof. By induction on the derivation that $t \in \mathsf{S}^{\mathsf{w}}_\vartheta$.

1. N^{w}-VAR: $t = x$ with $x \in \vartheta$. Take $\mathtt{E} = \Box$.
2. N^{w}-APP: $t = t_1\, t_2$ with $t_1 \in \mathsf{S}^{\mathsf{w}}_\vartheta$. Resort to the *i.h.* to obtain $\mathtt{E}_1$ and set $\mathtt{E} \stackrel{\text{def}}{=} \mathtt{E}_1\, t_2$.
3. N^{w}-SUB$^\circ$: $t = t_1[y\backslash t_2]$ with $t_1 \in \mathsf{S}^{\mathsf{w}}_\vartheta$ and $y \notin \mathsf{hrv}(t_1)$. Thus $\mathsf{hrv}(t) = \mathsf{hrv}(t_1)\backslash y = \mathsf{hrv}(t_1)$. Therefore $x \in \mathsf{hrv}(t_1)$, hence $x \neq y$, and the *i.h.* yields $\mathtt{E}_1$ such that $t_1 = \mathtt{E}_1[x]$. We conclude by setting $\mathtt{E} \stackrel{\text{def}}{=} \mathtt{E}_1[y\backslash t]$.

4. N^{w}-SUB$^{\bullet}$: $t = t_1[y\backslash t_2]$ and $t_1 \in \mathsf{S}^{\text{w}}_{\vartheta\cup\{x\}}$ and $y \in \mathsf{hrv}(t_1)$ and $t_2 \in \mathsf{S}^{\text{w}}_{\vartheta}$. By the *i.h.* there exists $\mathtt{E}_1$ such that $t_1 = \mathtt{E}_1\langle\!\langle y \rangle\!\rangle$. Also, $\mathsf{hrv}(t_1[y\backslash t_2]) = \mathsf{hrv}(t_1)\backslash y \cup \mathsf{hrv}(t_2) = \mathsf{hrv}(t_2)$. The last equality follows from Lem. A.38. Thus $x \in \mathsf{hrv}(t_2)$ and the *i.h.*, again, yields $\mathtt{E}_2$ such that $t_1 = \mathtt{E}_2\langle\!\langle x \rangle\!\rangle$. We conclude by setting $\mathtt{E} \stackrel{\text{def}}{=} \mathtt{E}_1\langle\!\langle y \rangle\!\rangle[y\backslash\mathtt{E}_2]$.

□

Remark A.40. Strong structures are also weak structures, *i.e.* $\mathsf{S}_\vartheta \subseteq \mathsf{S}^{\text{w}}_\vartheta$.

Remark A.41 (Weakening). If $s \in \mathsf{S}^{\text{w}}_\vartheta$ is a weak structure then $s \in \mathsf{S}^{\text{w}}_{\vartheta\cup\{x\}}$.

Note: technically we require that x does not occur bound in s, which we can always guarantee by α-conversion.

Lemma A.42. *If* $\mathtt{I}^\vartheta$ *is an inert evaluation context, then* $\mathtt{I}^\vartheta\langle\!\langle x \rangle\!\rangle \in \mathsf{S}^{\text{w}}_{\vartheta\cup\{x\}}$.

Proof. By induction on the derivation that $\mathtt{I}^\vartheta \in \mathsf{E}^\circ_\vartheta$.

1. EBox, $\mathtt{I}^\vartheta = \square$. Then $x \in \vartheta \cup \{x\}$ and the result follows from N^{w}-VAR.

2. EAppL, $\mathtt{I}^\vartheta = \mathtt{I}^\vartheta_1\, t$. By the *i.h.* on $\mathtt{I}^\vartheta_1$ and N^{w}-APP.

3. EAppRStr, $\mathtt{I}^\vartheta = t_1\, F^\vartheta$ with $t_1 \in \mathsf{S}_\vartheta$. By Rem. A.40, $t_1 \in \mathsf{S}^{\text{w}}_\vartheta$. Since x does not occur bound in t_1, by Rem. A.41, $t_1 \in \mathsf{S}^{\text{w}}_{\vartheta\cup\{x\}}$. We conclude using N^{w}-APP.

4. ESubLNonStr, $\mathtt{I}^\vartheta = \mathtt{I}^\vartheta_1[y\backslash t]$ with $t \notin \mathsf{S}_\vartheta$ and $y \notin \vartheta$. By the *i.h.* $\mathtt{I}^\vartheta_1\langle\!\langle x \rangle\!\rangle \in \mathsf{S}^{\text{w}}_{\vartheta\cup\{x\}}$. Moreover, $y \notin \vartheta$ and, by the statement of the result, also $y \neq x$. Hence $y \notin \vartheta \cup \{x\}$. We conclude from Lem A.37 and N^{w}-SUB$^{\circ}$.

5. ESubLStr, $\mathtt{I}^\vartheta = \mathtt{I}_1^{\vartheta\cup\{y\}}[y\backslash t_2]$ with $t_2 \in \mathsf{S}_\vartheta$. By the *i.h.* $\mathtt{I}_1^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle$ is a $\vartheta \cup \{x, y\}$-structure. Moreover, from Rem. A.40, $t_2 \in \mathsf{S}^{\text{w}}_\vartheta$. We may assume that y does not occur in t_2. We conclude using N^{w}-SUB$^{\circ}$or N^{w}-SUB$^{\bullet}$, depending on whether $y \in \mathsf{hrv}(\mathtt{I}_1^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle)$ or not.

6. ESubsR, $\mathtt{I}^\vartheta = \mathtt{I}^\vartheta_1\langle\!\langle y \rangle\!\rangle[y\backslash\mathtt{I}^\vartheta_2]$. By the *i.h.* $\mathtt{I}^\vartheta_1\langle\!\langle y \rangle\!\rangle \in \mathsf{S}^{\text{w}}_{\vartheta\cup\{y\}}$. By by Rem. A.41 $\mathtt{I}^\vartheta_1\langle\!\langle y \rangle\!\rangle \in \mathsf{S}^{\text{w}}_{\vartheta\cup\{x,y\}}$. We conclude using N^{w}-SUB$^{\circ}$or N^{w}-SUB$^{\bullet}$, depending on whether $y \in \mathsf{hrv}(\mathtt{I}^\vartheta_1\langle\!\langle x \rangle\!\rangle)$ or not.

□

Lemma A.43 (Strengthening)**.** *If* $s \in \mathsf{S}^{\text{w}}_{\vartheta\cup\{x\}}$ *is a weak structure and* $x \notin \mathsf{hrv}(s)$*, then* $s \in \mathsf{S}^{\text{w}}_\vartheta$.

Proof. By induction on the derivation that $s \in \mathsf{S}^{\text{w}}_{\vartheta\cup\{x\}}$. □

Lemma A.44. *Let* $\mathtt{I}^\vartheta \in \mathsf{E}^\circ_\vartheta$ *be an inert evaluation context. If* $\mathtt{I}^\vartheta[s] \notin \mathsf{S}^{\text{w}}_\vartheta$*, then* $\mathtt{I}^\vartheta \in \mathsf{WCtx}$.

Proof. By induction on the derivation that $\mathtt{I}^\vartheta \in \mathsf{E}^\circ_\vartheta$. The case EBoxis immediate:

1. EAppL, $\mathtt{I}^\vartheta_1\, t$. Since $\mathtt{I}^\vartheta[s] \notin \mathsf{S}^{\text{w}}_\vartheta$, then $\mathtt{I}^\vartheta_1[s] \notin \mathsf{S}^{\text{w}}_\vartheta$. Thus by *i.h.* $\mathtt{I}^\vartheta_1 \in \mathsf{WCtx}$. Hence also $\mathtt{I}^\vartheta \in \mathsf{WCtx}$.

2. ESubLNonStr, $\mathtt{I}_1^\vartheta[x\backslash t]$ with $t \notin \mathsf{S}_\vartheta, x \notin \vartheta$. Note that $\mathtt{I}_1^\vartheta[s] \notin \mathsf{S}^{\mathsf{w}}_\vartheta$ for otherwise by $x \notin \vartheta$ and Lem. A.37, we would have $\mathtt{I}^\vartheta[s] \in \mathsf{S}^{\mathsf{w}}_\vartheta$. Therefore $\mathtt{I}_1^\vartheta \in \mathsf{WCtx}$ by the *i.h.* and hence also $\mathtt{I}^\vartheta \in \mathsf{WCtx}$.

3. ESubLStr, $\mathtt{I}_1^{\vartheta\cup\{x\}}[x\backslash t]$ with $t \in \mathsf{S}_\vartheta$. Note that $\mathtt{I}_1^{\vartheta\cup\{x\}}[s] \notin \mathsf{S}^{\mathsf{w}}_{\vartheta\cup\{x\}}$, for otherwise $\mathtt{I}^\vartheta[s] \in \mathsf{S}^{\mathsf{w}}_\vartheta$ given that $t \in \mathsf{S}_\vartheta$ and Rem. A.40. Thus $\mathtt{I}_1^{\vartheta\cup\{x\}} \in \mathsf{WCtx}$ by the *i.h.* and hence also $\mathtt{I}^\vartheta \in \mathsf{WCtx}$.

4. ESubsR, $\mathtt{I}_1^\vartheta\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{I}_2^\vartheta]$. By Lem. A.42, $\mathtt{I}_1^\vartheta\langle\!\langle x\rangle\!\rangle$ is in $\mathsf{S}^{\mathsf{w}}_{\vartheta\cup\{x\}}$. Now it must be the case that $x \in \mathsf{hrv}(\mathtt{I}_1^\vartheta\langle\!\langle x\rangle\!\rangle)$ for otherwise $\mathtt{I}_1^\vartheta\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{I}_2^\vartheta[s]] \in \mathsf{S}^{\mathsf{w}}_\vartheta$, contradicting the hypothesis. Moreover,

   - $\mathtt{I}_2^\vartheta[s]$ must not be a weak structure for otherwise, again, $\mathtt{I}_1^\vartheta\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{I}_2^\vartheta[s]]$ would be a weak structure. Thus we can apply the *i.h.* on $\mathtt{I}_2^\vartheta$ and deduce that $\mathtt{I}_2^\vartheta \in \mathsf{WCtx}$.
   - From $x \in \mathsf{hrv}(\mathtt{I}_1^\vartheta\langle\!\langle x\rangle\!\rangle)$ and Lem. A.39, we deduce the existence of $\mathtt{E}_1$ such that $\mathtt{E}_1\langle\!\langle x\rangle\!\rangle = \mathtt{I}_1^\vartheta\langle\!\langle x\rangle\!\rangle$.

   We thus set $\mathtt{E} \stackrel{\text{def}}{=} \mathtt{E}_1\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{I}_2^\vartheta]$.

5. EAppRStr, $t\, F^\vartheta$ with $t \in \mathsf{S}_\vartheta$. In this case, Rem. A.40 implies $t\, F^\vartheta$ is a weak structure, contradicting the hypothesis. So this case is not possible.

□

Lemma A.45. *If $F^\vartheta \in \mathsf{E}_\vartheta$ is an evaluation context such that $F^\vartheta\langle\!\langle x\rangle\!\rangle \notin \mathsf{WNF}_\vartheta$ then $F^\vartheta \in \mathsf{WCtx}$.*

Proof. Note that $F^\vartheta\langle\!\langle x\rangle\!\rangle \notin \mathsf{WNF}_\vartheta$ so it is not an answer. Then, by the contrapositive of Lem. A.23, the evaluation context F^ϑ must be inert, *i.e.* $F^\vartheta \in \mathsf{E}^\circ_\vartheta$. Note moreover that $F^\vartheta\langle\!\langle x\rangle\!\rangle \notin \mathsf{S}^{\mathsf{w}}_\vartheta$, so by Lem. A.44 we conclude. □

Lemma A.46. *Let $F^\vartheta \in \mathsf{E}_\vartheta$ and $y \in \mathsf{hrv}(F^\vartheta[x])$ with $x \neq y$. Then $y \in \mathsf{hrv}(F^\vartheta[t])$, for any term t.*

Proof. We show this for $F^\vartheta \in \mathbb{X}^\vartheta$, where $\mathbb{X}^\vartheta$ stands for either the set E_ϑ or the set $\mathsf{E}^\circ_\vartheta$, by induction on the derivation that $F^\vartheta \in \mathbb{X}^\vartheta$.

1. EBox, □. Holds trivially since $y \notin \mathsf{hrv}(F^\vartheta[x])$.

2. EAppL, $\mathtt{I}^\vartheta\, t$. Follows from *i.h.* and the definition of head reachable variables.

3. ESubLNonStr, $F_1^\vartheta[z\backslash t]$ with $t \notin \mathsf{S}_\vartheta$ and $z \notin \vartheta$. If $z \in \mathsf{hrv}(F_1^\vartheta\langle\!\langle x\rangle\!\rangle)$, then we resort to the *i.h.*. If $z \in \mathsf{hrv}(F_1^\vartheta\langle\!\langle x\rangle\!\rangle)$ and $y \in \mathsf{hrv}(t)$, then we use the *i.h.* with respect to z and $F_1^\vartheta\langle\!\langle x\rangle\!\rangle$.

4. ESubLStr, $F_1^{\vartheta\cup\{z\}}[z\backslash s]$ with $s \in \mathsf{S}_\vartheta$. Similar to the previous case.

5. ESubsR, $F_1^\vartheta\langle\!\langle z\rangle\!\rangle[z\backslash \mathtt{I}^\vartheta]$. If $y \in \mathsf{hrv}(F_1^\vartheta\langle\!\langle z\rangle\!\rangle)$, the result is immediate. If $z \in \mathsf{hrv}(F_1^\vartheta\langle\!\langle z\rangle\!\rangle)$ and $y \in \mathtt{I}^\vartheta\langle\!\langle x\rangle\!\rangle$, then we resort to the *i.h.* on $\mathtt{I}^\vartheta$.

6. EAppRStr, $s\, F_1^\vartheta$ with $s \in \mathsf{S}_\vartheta$. Immediate.
7. ELam, $\lambda y.F_1^{\vartheta\cup\{y\}}$. This case is not possible since $\mathsf{hrv}(\lambda y.F_1^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle) = \varnothing$.

□

Lemma A.47. *Let $\mathbb{X}^\vartheta$ stand for either WNF_ϑ or $\mathsf{S}^{\mathsf{w}}_\vartheta$. If $x \notin \mathsf{hrv}(F^\vartheta[x])$ and $F^\vartheta[x] \in \mathbb{X}^\vartheta$, then $F^\vartheta[t] \in \mathbb{X}^\vartheta$.*

Proof. By simultaneous induction on $F^\vartheta[x] \in \mathsf{WNF}_\vartheta$ and $F^\vartheta[x] \in \mathsf{S}^{\mathsf{w}}_\vartheta$. The n$^{\mathsf{w}}$-var, n$^{\mathsf{w}}$-incl, and n$^{\mathsf{w}}$-lam cases are immediate.

1. n$^{\mathsf{w}}$-app: $F^\vartheta[x] = t_1 t_2$ with $t_1 \in \mathsf{S}^{\mathsf{w}}_\vartheta$. Then two cases are possible:
   - $t_1 t_2 = \mathtt{I}^\vartheta\, t_2$. We resort to the *i.h.*.
   - $t_1 t_2 = t_1\, F_1^\vartheta$. The result is immediate.
2. n$^{\mathsf{w}}$-sub$^\bullet$: $F^\vartheta[x] = t_1[y\backslash t_2]$ with $t_1 \in \mathbb{X}^{\vartheta\cup\{y\}}$ and $y \in \mathsf{hrv}(t_1)$ and $t_2 \in \mathsf{S}^{\mathsf{w}}_\vartheta$. The following further cases are considered:
   - $F^\vartheta[x] = F_1^\vartheta[x][y\backslash t_2]$ with $t_2 \notin \mathsf{S}_\vartheta$ and $y \notin \vartheta$. The result follows from the *i.h.* on F_1^ϑ and Lem. A.46.
   - $F^\vartheta[x] = F_1^{\vartheta\cup\{y\}}[x][y\backslash t_2]$ with $t_2 \in \mathsf{S}_\vartheta$. The result follows from the *i.h.* and Lem A.46.
   - $F^\vartheta[x] = F_1^\vartheta\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{I}^\vartheta[x]]$ and $t_2 = \mathtt{I}^\vartheta[x]$. Since $y \in \mathsf{hrv}(t_1)$, then $x \notin \mathsf{hrv}(\mathtt{I}^\vartheta[x])$. Therefore we conclude from the *i.h.* on $\mathtt{I}^\vartheta$.
3. n$^{\mathsf{w}}$-sub$^\circ$: $F^\vartheta[x] = t_1[y\backslash t_2]$ with $t_1 \in \mathbb{X}^\vartheta$ and $y \notin \mathsf{hrv}(t_1)$. The following cases are possible:
   - $F^\vartheta[x] = F_1^\vartheta[x][y\backslash t_2]$ with $t_2 \notin \mathsf{S}_\vartheta$ and $y \notin \vartheta$. The result follows from the *i.h.*
   - $F^\vartheta[x] = F_1^{\vartheta\cup\{y\}}[x][y\backslash t_2]$ with $t_2 \in \mathsf{S}_\vartheta$. The result follows from the *i.h.*
   - $F^\vartheta[x] = F_1^\vartheta\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{I}^\vartheta[x]]$ and $t_2 = \mathtt{I}^\vartheta[x]$. The result is immediate.

□

Lemma A.48. *Let r be a redex and $\mathbb{X}^\vartheta$ stand for either $\mathsf{S}^{\mathsf{w}}_\vartheta$ or WNF_ϑ. If $F^\vartheta[r] \in \mathbb{X}^\vartheta$, then $\mathsf{hrv}(F^\vartheta[r]) = \mathsf{hrv}(F^\vartheta[s])$, for any s. This also holds, in particular, if F^ϑ is an inert evaluation context.*

Proof. By induction on the derivation that $F^\vartheta[r] \in \mathbb{X}^\vartheta$. The n$^{\mathsf{w}}$-var, n$^{\mathsf{w}}$-incl, and n$^{\mathsf{w}}$-lam cases are immediate.

1. n$^{\mathsf{w}}$-app: $F^\vartheta[r] = t_1 t_2$ with $t_1 \in \mathsf{S}^{\mathsf{w}}_\vartheta$. Then two cases are possible:
   - $t_1 t_2 = \mathtt{I}^\vartheta\, t_2$. We resort to the *i.h.* with respect to item (2).
   - $t_1 t_2 = t_1\, F_1^\vartheta$. The result is immediate.

2. N^{w}-SUB$^{\bullet}$: $F^{\vartheta}[r] = t_1[y\backslash t_2]$ with $t_1 \in \mathbb{X}^{\vartheta\cup\{y\}}$ and $y \in \mathsf{hrv}(t_1)$ and $t_2 \in \mathsf{S}^{\mathsf{w}}_{\vartheta}$. The following further cases are considered:

   - $F^{\vartheta}[r] = F_1^{\vartheta}[r][y\backslash t_2]$ with $t_2 \notin \mathsf{S}_{\vartheta}$ and $y \notin \vartheta$. The result follows from the *i.h.* on $F_1^{\vartheta}[r]$.
   - $F^{\vartheta}[r] = F_1^{\vartheta\cup\{y\}}[r][y\backslash t_2]$ with $t_2 \in \mathsf{S}_{\vartheta}$. The result follows from the *i.h.* on $F_1^{\vartheta\cup\{y\}}[r]$.
   - $F^{\vartheta}[r] = F_1^{\vartheta}\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{I}^{\vartheta}[r]]$ and $t_2 = \mathtt{I}^{\vartheta}[x]$. The result follows from the *i.h.* on $\mathtt{I}^{\vartheta}[x]$.

3. N^{w}-SUB$^{\circ}$: $F^{\vartheta}[r] = t_1[y\backslash t_2]$ with $t_1 \in \mathbb{X}^{\vartheta}$ and $y \notin \mathsf{hrv}(t_1)$. The following cases are possible:

   - $F^{\vartheta}[r] = F_1^{\vartheta}[r][y\backslash t_2]$ with $t_2 \notin \mathsf{S}_{\vartheta}$ and $y \notin \vartheta$. The result follows from the *i.h.*
   - $F^{\vartheta}[r] = F_1^{\vartheta\cup\{y\}}[r][y\backslash t_2]$ with $t_2 \in \mathsf{S}_{\vartheta}$. The result follows from the *i.h.*
   - $F^{\vartheta}[r] = F_1^{\vartheta}\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{I}^{\vartheta}[r]]$ and $t_2 = \mathtt{I}^{\vartheta}[r]$. The result follows from the *i.h.*.

□

Lemma A.49. *Let $\mathbb{X}^{\vartheta}$ stand for either WNF_{ϑ} or $\mathsf{S}^{\mathsf{w}}_{\vartheta}$. If $t \overset{\vartheta}{\rightsquigarrow}_{\mathtt{r}} u$ and $t \in \mathbb{X}^{\vartheta}$, then $u \in \mathbb{X}^{\vartheta}$.*

Proof. By induction on the derivation that $t \in \mathbb{X}^{\vartheta}$. The N^{w}-VAR, N^{w}-INCL, and N^{w}-LAM cases are immediate.

1. N^{w}-APP: $t = t_1 t_2$ and $t_1 \in \mathsf{S}^{\mathsf{w}}_{\vartheta}$. Then $t_1\, t_2 = F^{\vartheta}[r]$ and we have two further cases. If $t_1 \notin \mathsf{S}_{\vartheta}$, then it must be the case that $F^{\vartheta}[r] = \mathtt{I}^{\vartheta}[r]\, t_2$ and the result follows from the *i.h.* with respect to item (2) and N^{w}-APP. If $t_1 \in \mathsf{S}_{\vartheta}$, then it must be the case that $F^{\vartheta}[r] = t_1\, F_1^{\vartheta}[r]$ and the result is immediate from N^{w}-APP.

2. N^{w}-SUB$^{\bullet}$: $t = t_1[x\backslash t_2]$ and $t_1 \in \mathbb{X}^{\vartheta\cup\{x\}}$, $x \in \mathsf{hrv}(t_1)$, $t_2 \in \mathsf{S}^{\mathsf{w}}_{\vartheta}$. Note that $F^{\vartheta} = \square$ is not possible since t_2 is not an answer. Thus one of the following holds:

   - $F^{\vartheta}[r] = F_1^{\vartheta}[r][x\backslash t_2]$ with $t_2 \notin \mathsf{S}_{\vartheta}$ and $x \notin \vartheta$. By Lem. A.37 this case is not possible.
   - $F^{\vartheta}[r] = F_1^{\vartheta\cup\{x_n\}}[r][x\backslash t_2]$ with $t_2 \in \mathsf{S}_{\vartheta}$. The result follows from the *i.h.* and N^{w}-SUB$^{\bullet}$or N^{w}-SUB$^{\circ}$.
   - $F^{\vartheta}[r] = F_1^{\vartheta}\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{I}^{\vartheta}[r]]$ and $t_2 = \mathtt{I}^{\vartheta}[r]$. We use the *i.h.* with respect to item (2) on t_2 and then conclude using either N^{w}-SUB$^{\bullet}$.

3. N^{w}-SUB$^{\circ}$: $t = t_1[x\backslash t_2]$ and $t_1 \in \mathbb{X}^{\vartheta}$ and $x \notin \mathsf{hrv}(t_1)$. One of the following holds:

   - $F^{\vartheta} = \square$. $t_1 = F_1^{\vartheta}\langle\!\langle x\rangle\!\rangle$ and t_2 is an answer. By Lem. A.47, $F_1^{\vartheta}\langle\!\langle t_2\rangle\!\rangle \in \mathsf{WNF}_{\vartheta}$. Thus we conclude using N^{w}-SUB$^{\circ}$. A similar argument applies if $\mathbb{X} = \mathsf{S}^{\mathsf{w}}$.
   - $F^{\vartheta}[r] = F_1^{\vartheta}[r][x\backslash t_2]$ with $t_2 \notin \mathsf{S}_{\vartheta}$ and $x \notin \vartheta$. The result follows from the *i.h.* and Lem A.48 (which guarantees that $x \notin \mathsf{hrv}(t_1')$, where t_1' is the reduct of $F_1^{\vartheta}[r]$).
   - $F^{\vartheta}[r] = F_1^{\vartheta\cup\{x\}}[r][x\backslash t_2]$ with $t_2 \in \mathsf{S}_{\vartheta}$. The result follows from the *i.h.*

- $F^{\vartheta}[r] = F_1^{\vartheta}\langle\!\langle x \rangle\!\rangle[x \backslash \mathtt{I}^{\vartheta}[r]]$ and $t_2 = \mathtt{I}^{\vartheta}[r]$. The result is immediate.

□

Theorem A.50 (Full proof of Thm. 4.23—Conservativity). *If* $t_0 \overset{\vartheta}{\rightsquigarrow} t_1 \overset{\vartheta}{\rightsquigarrow} \dots t_{n-1} \overset{\vartheta}{\rightsquigarrow} t_n$ *there exists an* $1 \leqslant i \leqslant n$ *such that the three following conditions hold:*

1. $t_0 \overset{\mathsf{W}}{\rightsquigarrow} t_1 \overset{\mathsf{W}}{\rightsquigarrow} \dots t_{n-1} \overset{\mathsf{W}}{\rightsquigarrow} t_i$
2. $t_i \overset{\vartheta \backslash \mathsf{W}}{\rightsquigarrow} t_{i+1} \overset{\vartheta \backslash \mathsf{W}}{\rightsquigarrow} \dots t_{n-1} \overset{\vartheta \backslash \mathsf{W}}{\rightsquigarrow} t_n$
3. *If* $i < n$, *then* $t_j \in \mathsf{N}_{\vartheta}^{\mathsf{w}}$ *for all* $i \leqslant j \leqslant n$.

Proof. Let $\mathtt{C}_i\langle\!\langle x \rangle\!\rangle$ be the context selected at step $t_i \overset{\vartheta}{\rightsquigarrow} t_{i+1}$. If $\mathtt{C}_i\langle\!\langle x \rangle\!\rangle \notin \mathsf{S}_{\vartheta}^{\mathsf{w}}$ and $\mathtt{C}_i\langle\!\langle x \rangle\!\rangle$ not an answer, then $\mathtt{C}_i \in \mathsf{WCtx}$ (*cf.* Lem A.45 in Appendix) and hence step $t_i \overset{\vartheta}{\rightsquigarrow} t_{i+1}$ is also a weak step. If, moreover, this happens for every step i with $i \in 1..n$, then we are done. Otherwise, let $j \in 1..n$ be the first index such that either $\mathtt{C}_j\langle\!\langle x \rangle\!\rangle \in \mathsf{S}_{\vartheta}^{\mathsf{w}}$ or $\mathtt{C}_j\langle\!\langle x \rangle\!\rangle$ not an answer. Then all subsequent steps are non-weak steps; this follows from the fact that weak call-by-need reduction preserves both answers and weak structures (*cf.* Lem. A.49). □

A.2.5 Commutation — proof of Lem. 4.49 and Lem. 4.50

In this section we give a proof of the *backward stability by internal steps* result stated in Lem. 4.49, and the *postponement of internal steps* result stated in Lem. 4.50. These proofs are long and technical. Before being able to give a complete proof, we need many auxiliary lemmas. The items in the statement of Lem. 4.49 are scattered throughout various lemmas: the first item of Lem. 4.49 is proved in Lem. A.51 (backward stability of answers) and Lem. A.52 (backward stability of db-redexes); the second item of Lem. 4.49 is proved in Lem. A.69 (backward stability of normal forms); the third item of Lem. 4.49 is proved in Lem. A.70 (backward stability of evaluation contexts).

The rest of this section is organized in subsections as follows. Sec. A.2.5 deals with backward stability of answers and db-redexes. Sections A.2.5–A.2.5 introduce auxiliary notions and results: the set of structural variables of an evaluation context (Sec. A.2.5), critical contexts (Sec. A.2.5), an analysis of the context that results from replacing a value by a variable in an evaluation context (Sec. A.2.5), a solution for a unification problem with evaluation contexts (Sec. A.2.5), and non-garbage contexts (Sec. A.2.5). Backward stability is then addressed for normal forms (Sec. A.2.5) and evaluation contexts (Sec. A.2.5). Finally we turn to the postponement result itself (Sec. A.2.5).

As a notational remark, in this section we use *anchor of a redex* to refer to the underlined subterm in each of the following cases:

1. db-**redex:** $\mathtt{C}\langle\underline{(\lambda x.t)\mathtt{L}\, s}\rangle$, *i.e.*, the anchor is the pattern of the db-redex.
2. lsv-**redex:** $\mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle \underline{x} \rangle\!\rangle[x \backslash \mathtt{vL}]\rangle$, *i.e.*, the anchor is the occurrence of x substituted by the lsv-step.

Backward stability of answers and db-redexes by internal steps

In this subsection we tackle the first item of Lem. 4.49. We recall the statement: if $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t$ and t is an answer (resp. a db redex), then t_0 must also be an answer (resp. a db redex). Backward stability of db redexes is necessary to show that internal steps can be postponed in a situation like $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (\lambda x.t)\mathtt{L}s \overset{\vartheta}{\rightsquigarrow} t[x\backslash s]\mathtt{L}$, to ensure that there is a db step at the root of t_0. Backward stability of answers is necessary to show that internal steps can be postponed in a situation like $x[x\backslash t_0] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} x[x\backslash \mathtt{vL}] \overset{\vartheta}{\rightsquigarrow} \mathtt{v}[x\backslash \mathtt{v}]\mathtt{L}$. In that case it can be argued that the step $t_0 \to \mathtt{vL}$ has to be internal, so t_0 is an answer and there is a lsv step at the root of t_0.

Lemma A.51 (Backward stability of answers)**.** *Let $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (\lambda x.s)\mathtt{L} = t$ be a ϑ-internal step. Then the source of the step is of the form $t_0 = (\lambda x.s_0)\mathtt{L}_0$. Moreover, the anchor of the step is not below a substitution context,* i.e. *it is inside s_0 or inside one of the arguments of $\mathtt{L}_0$.*

Proof. By induction on the context C under which the step takes place:

1. Empty, $\mathtt{C} = \Box$ Note that the step cannot be a db step, as it would then be a ϑ-external step, since $\Box \in \mathsf{E}_\vartheta$.

   So the step must be a lsv step, contracting the outermost substitution, that is, $t_0 = \mathtt{C}_1\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{vL}_2] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}_1\langle \mathtt{v}\rangle[y\backslash \mathtt{v}]\mathtt{L}_2 = t$. Note that $\mathtt{C}_1\langle \mathtt{v}\rangle = (\lambda x.s)\mathtt{L}_1$ where $\mathtt{L} = \mathtt{L}_1[y\backslash \mathtt{v}]\mathtt{L}_2$.

   We claim that $\mathtt{C}_1$ is not a substitution context. By contradiction, suppose that $\mathtt{C}_1$ is a substitution context. Then the lsv step $t_0 = y\mathtt{L}'[y\backslash \mathtt{vL}_2] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{vL}'[y\backslash \mathtt{v}]\mathtt{L}_2 = t$ is ϑ-external since $\mathtt{L}'[y\backslash \mathtt{vL}_2] \in \mathsf{E}_\vartheta$. This contradicts the assumption that the step is ϑ-internal.

   Now, since $\mathtt{C}_1\langle \mathtt{v}\rangle = (\lambda x.s)\mathtt{L}_1$, there are two cases, depending on the position of the hole of $\mathtt{C}_1$:

   1.1 The hole of $\mathtt{C}_1$ lies inside s Then $\mathtt{C}_1 = (\lambda x.\mathtt{C}_2)\mathtt{L}_1$, and the step is of the form $t_0 = (\lambda x.\mathtt{C}_2\langle\!\langle y \rangle\!\rangle)\mathtt{L}_1[y\backslash \mathtt{vL}_2] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (\lambda x.\mathtt{C}_2\langle \mathtt{v}\rangle)\mathtt{L}_1[y\backslash \mathtt{v}]\mathtt{L}_2 = t$. By taking $s_0 := \mathtt{C}_2\langle\!\langle y \rangle\!\rangle$ and $\mathtt{L}_0 := \mathtt{L}_1[y\backslash \mathtt{vL}_2]$ we conclude.

   1.2 The hole of $\mathtt{C}_1$ lies inside $\mathtt{L}_1$ Then $\mathtt{C}_1 = (\lambda x.s)\mathtt{L}_{11}[z\backslash \mathtt{C}_2]\mathtt{L}_{12}$ where $\mathtt{L}_1 = \mathtt{L}_{11}[z\backslash \mathtt{C}_2\langle \mathtt{v}\rangle]\mathtt{L}_{12}$, and the step is of the form $t_0 = (\lambda x.s)\mathtt{L}_{11}[z\backslash \mathtt{C}_2\langle\!\langle y \rangle\!\rangle]\mathtt{L}_{12}[y\backslash \mathtt{vL}_2] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (\lambda x.s)\mathtt{L}_{11}[z\backslash \mathtt{C}_2\langle \mathtt{v}\rangle]\mathtt{L}_{12}[y\backslash \mathtt{v}]\mathtt{L}$ t. By taking $s_0 := s$ and $\mathtt{L}_0 := \mathtt{L}_{11}[z\backslash \mathtt{C}_2\langle\!\langle y \rangle\!\rangle]\mathtt{L}_{12}[y\backslash \mathtt{vL}_2]$ we conclude.

   Note that, as already argued, in both cases, the anchor of the step is not below a substitution context.

2. Inside an abstraction, $\mathtt{C} = \lambda x.\mathtt{C}'$ The step is of the form $t_0 = \lambda x.\mathtt{C}'\langle r_0\rangle \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \lambda x.\mathtt{C}'\langle r\rangle = t$, so $\mathtt{L} = \mathtt{L}_0 = \Box$, with $s_0 = \mathtt{C}'\langle r_0\rangle$ and $s = \mathtt{C}'\langle r\rangle$. Note that the anchor of the step is inside s_0, hence not below a substitution context.

3. Left of an application, $\mathtt{C} = \mathtt{C}'\, u$ Impossible, since the step would be of the form $t_0 = \mathtt{C}'\langle r_0\rangle\, u \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}'\langle r\rangle\, u = t$ but t is not an application.

4. Right of an application, $\mathtt{C} = u\,\mathtt{C}'$ Impossible, analogous to the previous case.

5. Left of a substitution, $\mathtt{C} = \mathtt{C}'[y\backslash u]$ Then the step is of the form $t_0 = \mathtt{C}'\langle r\rangle[y\backslash u] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}'\langle r'\rangle[y\backslash u] = (\lambda x.s)\mathtt{L}'[y\backslash u] = t$, where $\mathtt{L} = \mathtt{L}'[y\backslash u]$. We consider two cases, depending on whether u is a strong ϑ-structure:

   5.1 If $u \in \mathsf{S}_\vartheta$ Note that the isomorphic step $\mathtt{C}'\langle r\rangle \to_{\mathtt{sh}\backslash\mathtt{gc}} (\lambda x.s)\mathtt{L}'$, taking place under the context $\mathtt{C}'$, cannot be $(\vartheta \cup \{y\})$-external, since then the fact that $\mathtt{C}' \in \mathsf{E}_{\vartheta\cup\{y\}}$ would imply that $\mathtt{C}'[y\backslash u] \in \mathsf{E}_\vartheta$, and the original step would be ϑ-external, contradicting the hypothesis.
   Hence the step $\mathtt{C}'\langle r\rangle \xrightarrow{\neg\vartheta\cup\{y\}}_{\mathtt{sh}} (\lambda x.s)\mathtt{L}'$ is $(\vartheta \cup \{y\})$-internal. By *i.h.* we have that $\mathtt{C}'\langle r\rangle = (\lambda x.s_0)\mathtt{L}'_0$, so the source of the original step is of the form $\mathtt{C}'\langle r\rangle[y\backslash u] = (\lambda x.s_0)\mathtt{L}'_0[y\backslash u]$. By *i.h.*, we also have that the anchor of the step is either inside s_0, or inside one of the arguments of $\mathtt{L}'_0$ By taking $\mathtt{L}_0 := \mathtt{L}'_0[y\backslash u]$ we conclude.

   5.2 If $u \notin \mathsf{S}_\vartheta$ Similar to the previous case: the isomorphic step $\mathtt{C}'\langle r\rangle \to_{\mathtt{sh}\backslash\mathtt{gc}} (\lambda x.s)\mathtt{L}'$, taking place under the context $\mathtt{C}'$, cannot be ϑ-external, as this would imply that the original step is ϑ-external.
   So it must be ϑ-internal and we may apply the *i.h.* to conclude that $\mathtt{C}'\langle r\rangle = (\lambda x.s_0)\mathtt{L}'_0$ and, moreover, that the anchor of the step is either inside s_0, or inside one of the arguments of $\mathtt{L}'_0$. This means that the source of the original step is of the form $\mathtt{C}'\langle r\rangle[y\backslash u] = (\lambda x.s_0)\mathtt{L}'_0[y\backslash u]$, as required.

6. Inside a substitution, $\mathtt{C} = u[y\backslash\mathtt{C}']$ Then it must be the case that $u = (\lambda x.s)\mathtt{L}'$ and the step is of the form $(\lambda x.s)\mathtt{L}'[y\backslash\mathtt{C}'\langle r\rangle] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (\lambda x.s)\mathtt{L}'[y\backslash\mathtt{C}'\langle r'\rangle]$, with $\mathtt{L} = \mathtt{L}'[y\backslash\mathtt{C}'\langle r'\rangle]$. By taking $s_0 := s$ and $\mathtt{L}_0 := \mathtt{L}'[y\backslash\mathtt{C}'\langle r\rangle]$ we conclude. Note that the anchor of the step is inside one of the arguments of $\mathtt{L}_0$, as required.

□

Lemma A.52 (Backward stability of db-redexes)**.** *Let $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (\lambda x.s)\mathtt{L}\,u = t$ be a ϑ-internal step. Then the source of the step is of the form $t_0 = (\lambda x.s_0)\mathtt{L}_0\,u_0$. Moreover, the anchor of the step is not below a context of the form $\mathtt{L}'\,u_0$,* i.e. *it is inside s_0, inside one of the arguments of $\mathtt{L}_0$, or inside u_0.*

Proof. By case analysis on the shape of the context $\mathtt{C}$ under which the step takes place. The interesting case is when going to the left of an application, that is $\mathtt{C} = \mathtt{C}'\,u$. Then the step is of the form $\mathtt{C}'\langle r\rangle\,u \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}'\langle r'\rangle\,u$. Consider the isomorphic step $\mathtt{C}'\langle r\rangle \to_{\mathtt{sh}\backslash\mathtt{gc}} \mathtt{C}'\langle r'\rangle$ takes place under the context $\mathtt{C}'$. We consider two cases, depending on whether $\mathtt{C}'$ is an generalized evaluation context over ϑ:

1. If $\mathtt{C}' \in \mathsf{E}_\vartheta$ Note that $\mathtt{C}'$ is not an inert evaluation context, *i.e.* $\mathtt{C}' \notin \mathsf{E}^\circ_\vartheta$, since otherwise we would have $\mathtt{C}'\,u \in \mathsf{E}_\vartheta$, which means that the original step is ϑ-external. So $\mathtt{C}' \in \mathsf{E}_\vartheta\backslash\mathsf{E}^\circ_\vartheta$ and by Lem. A.23 we know that $\mathtt{C}'$ has the form of an answer. More precisely, there are two subcases:

1.1 The context $\mathtt{C}'$ is of the form $(\lambda x.\mathtt{C}'')\mathtt{L}$ Then the original step is $t_0 = (\lambda x.\mathtt{C}''\langle r\rangle)\mathtt{L}\, s \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (\lambda x.\mathtt{C}''\langle r'\rangle)\mathtt{L}\, s = t$. Taking $s_0 := \mathtt{C}''\langle r\rangle$, with $\mathtt{L}_0 = \mathtt{L}$ and $u_0 = u$ we conclude.

1.2 The context $\mathtt{C}'$ is of the form $(\lambda x.s)\mathtt{L}_1[y\backslash\mathtt{C}'']\mathtt{L}_2$ Then the original step is of the form $t_0 = (\lambda x.s)\mathtt{L}_1[y\backslash\mathtt{C}''\langle r\rangle]\mathtt{L}_2\, s \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (\lambda x.s)\mathtt{L}_1[y\backslash\mathtt{C}''\langle r'\rangle]\mathtt{L}_2\, s = t$. By taking $\mathtt{L}_0 := \mathtt{L}_1[y\backslash\mathtt{C}''\langle r\rangle]\mathtt{L}_2$, with $s_0 = s$ and $u_0 = u$ we conclude.

2. If $\mathtt{C}' \notin \mathsf{E}_\vartheta$ Then the step $\mathtt{C}'\langle r\rangle \to_{\mathtt{sh}\backslash\mathtt{gc}} \mathtt{C}'\langle r'\rangle = (\lambda x.s)\mathtt{L}$ is ϑ-internal, so since answers are backward stable by internal steps (Lem. A.51) we have that $\mathtt{C}'\langle r\rangle = (\lambda x.s_0)\mathtt{L}_0$ and the anchor of the step is not below a substitution context. Hence $t = \mathtt{C}'\langle r\rangle\, u = (\lambda x.s_0)\mathtt{L}_0\, u$ and the anchor of the original step is not below a context of the form $\mathtt{L}\, u$, as required.

□

Structural variables

In this subsection we introduce *structural variables*. Intuitively, the structural variables $\mathsf{sv}(\mathtt{C})$ of an evaluation context $\mathtt{C}$ are the free variables that *must* be frozen in order for $\mathtt{C}$ to be an evaluation context. For instance $\mathsf{sv}((x\Box)[y\backslash z]) = \{x\}$ and $\mathsf{sv}((x\Box)[x\backslash z]) = \{x, z\}$. To understand the definition of structural variables, it might be helpful to observe that a term in the process of being evaluated has *already frozen* subterms, which have been normalized and are morally to the left of the focus of evaluation, and *still pending* subterms, which are yet to be evaluated and are morally to the right of the focus of evaluation. Structural variables are defined to be the non-garbage variables that occur in the already frozen subterms.

Structural variables are required as tools to reason over arbitrary evaluation contexts. In particular, the strengthening lemma for evaluation contexts (Lem. A.54) allows obtaining a ϑ-evaluation context from a $(\vartheta \cup \{x\})$-evaluation context $\mathtt{C}$ depending on whether $x \in \mathsf{sv}(\mathtt{C})$. This mimicks the strengthening lemma for normal forms that we have already stated (Lem. A.25) which allows obtaining a ϑ-normal form from a $(\vartheta \cup \{x\})$-normal form t depending on whether $x \in \mathsf{ngv}(t)$.

In this subsection we also introduce a "proof tactic" (Tactic A.55) that will be used later.

Definition A.53 (Structural variables)**.** Let $\mathtt{C}$ be a generalized evaluation context over ϑ. More precisely, let $\mathtt{C} \in \mathbb{X}^\vartheta$ where $\mathbb{X}^\vartheta$ is either the set E_ϑ or the set $\mathsf{E}^\circ_\vartheta$. The set of *structural variables*

of $\mathtt{C}$, written $\mathsf{sv}(\mathtt{C})$ is defined by induction on the derivation that $\mathtt{C} \in \mathbb{X}^{\vartheta}$ as follows:

$$
\begin{array}{lrcl}
(\textsc{EBox}) & \mathsf{sv}(\Box) & \stackrel{\text{def}}{=} & \varnothing \\
(\textsc{EAppL}) & \mathsf{sv}(\mathtt{C}_1\, t) & \stackrel{\text{def}}{=} & \mathsf{sv}(\mathtt{C}_1) \\
 & & & \text{where } \mathtt{C}_1 \in \mathsf{E}^{\circ}_{\vartheta} \\
(\textsc{ESubLNonStr}) & \mathsf{sv}(\mathtt{C}_1[x\backslash t]) & \stackrel{\text{def}}{=} & \mathsf{sv}(\mathtt{C}_1)\backslash\{x\} \\
 & & & \text{where } \mathtt{C}_1 \in \mathbb{X}^{\vartheta}, x \notin \vartheta, \text{ and } t \notin \mathsf{S}_{\vartheta} \\
(\textsc{ESubLStr}) & \mathsf{sv}(\mathtt{C}_1[x\backslash t]) & \stackrel{\text{def}}{=} & (\mathsf{sv}(\mathtt{C}_1)\backslash\{x\}) \cup \begin{cases} \mathsf{ngv}(t) & \text{if } x \in \mathsf{sv}(\mathtt{C}_1) \\ \varnothing & \text{otherwise} \end{cases} \\
 & & & \text{where } \mathtt{C}_1 \in \mathbb{X}^{\vartheta\cup\{x\}} \text{ and } t \in \mathsf{S}_{\vartheta} \\
(\textsc{ESubsR}) & \mathsf{sv}(\mathtt{C}_1\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{C}_2]) & \stackrel{\text{def}}{=} & (\mathsf{sv}(\mathtt{C}_1)\backslash\{x\}) \cup \mathsf{sv}(\mathtt{C}_2) \\
 & & & \text{where } \mathtt{C}_1 \in \mathbb{X}^{\vartheta} \text{ and } \mathtt{C}_2 \in \mathsf{E}^{\circ}_{\vartheta} \\
(\textsc{EAppRStr}) & \mathsf{sv}(t\, \mathtt{C}_1) & \stackrel{\text{def}}{=} & \mathsf{ngv}(t) \cup \mathsf{sv}(\mathtt{C}_1) \\
 & & & \text{where } t \in \mathsf{S}_{\vartheta} \text{ and } \mathtt{C}_1 \in \mathsf{E}_{\vartheta} \\
(\textsc{ELam}) & \mathsf{sv}(\lambda x.\mathtt{C}_1) & \stackrel{\text{def}}{=} & \mathsf{sv}(\mathtt{C}_1)\backslash\{x\} \\
 & & & \text{where } \mathtt{C}_1 \in \mathsf{E}_{\vartheta\cup\{x\}}
\end{array}
$$

Lemma A.54 (Strengthening for evaluation contexts)**.** *Let $\mathtt{C}$ be a (resp. inert) $(\vartheta \cup \{x\})$-evaluation context.*

1. *If $x \notin \mathsf{sv}(\mathtt{C})$ then $\mathtt{C}$ is a (resp. inert) ϑ-evaluation context.*
2. *If $x \in \mathsf{sv}(\mathtt{C})$ and $x \notin \vartheta$ then for any term q there is a (resp. inert) ϑ-evaluation context $\mathtt{C}_2$ such that $\mathtt{C}\langle q \rangle = \mathtt{C}_2\langle\!\langle x \rangle\!\rangle$.*

Proof. For the first item of the lemma, let $\mathbb{X}^{\vartheta}$ denote either the set E_{ϑ} or the set $\mathsf{E}^{\circ}_{\vartheta}$. Let us show that if $\mathtt{C} \in \mathbb{X}^{\vartheta\cup\{x\}}$ and $x \notin \mathsf{sv}(\mathtt{C})$, then $\mathtt{C} \in \mathbb{X}^{\vartheta}$. Proceed by induction on the size of the context $\mathtt{C}$, and then by case analysis on the last step of the derivation that $\mathtt{C} \in \mathbb{X}^{\vartheta\cup\{x\}}$. The interesting case is when $\mathtt{C}$ is built by applying ESubLStr, that is, $\mathtt{C} = \mathtt{C}_1[y\backslash t] \in \mathbb{X}^{\vartheta\cup\{x\}}$ with $t \in \mathsf{S}_{\vartheta\cup\{x\}}$ and $\mathtt{C}_1 \in \mathbb{X}^{\vartheta\cup\{x,y\}}$. Then $x \notin \mathsf{sv}(\mathtt{C}_1[y\backslash t]) \supseteq \mathsf{sv}(\mathtt{C}_1)\backslash\{y\}$, so $x \notin \mathsf{sv}(\mathtt{C}_1)\backslash\{y\}$. Observe that $x \neq y$ by the variable convention, so actually $x \notin \mathsf{sv}(\mathtt{C}_1)$. Hence we can apply the *i.h.*, obtaining that $\mathtt{C}_1 \in \mathbb{X}^{\vartheta\cup\{y\}}$. We consider two cases, depending on whether y is structural in $\mathtt{C}_1$:

1. If $y \in \mathsf{sv}(\mathtt{C}_1)$ Then, by definition of the structural variables, we have that $\mathsf{sv}(\mathtt{C}) = (\mathsf{sv}(\mathtt{C}_1)\backslash\{y\}) \cup \mathsf{ngv}(t)$. In particular, $x \notin \mathsf{ngv}(t)$. By the fact that garbage variables are not required in "ϑ" (Lem. A.25) we have that $t \in \mathsf{S}_{\vartheta}$. Now we can apply the formation rule for generalized contexts adding a structural substitution (ESubLStr), and conclude $\mathtt{C}_1[y\backslash t] \in \mathbb{X}^{\vartheta}$, as required.
2. If $y \notin \mathsf{sv}(\mathtt{C}_1)$ Then we may apply the *i.h.* again on the fact that $\mathtt{C}_1 \in \mathbb{X}^{\vartheta\cup\{y\}}$ to obtain that $\mathtt{C}_1 \in \mathbb{X}^{\vartheta}$. By the fact that adding an arbitrary substitution preserves evaluation contexts (Lem. A.26) we have that $\mathtt{C}_1[x\backslash t] \in \mathbb{X}^{\vartheta}$, as required.

For the second item of the lemma, let $\mathbb{X}^{\vartheta}$ stand for either E_{ϑ} or $\mathsf{E}^{\circ}_{\vartheta}$, and let $\mathtt{C}_1 \in \mathbb{X}^{\vartheta\cup\{x\}}$ where $x \in \mathsf{sv}(\mathtt{C}_1)$ and $x \notin \vartheta$. Let us show that for any term q there is a context $\mathtt{C}_2 \in \mathbb{X}^{\vartheta}$ such that $\mathtt{C}_1\langle q\rangle = \mathtt{C}_2\langle\!\langle x\rangle\!\rangle$. Proceed by induction on the size of the term $\mathtt{C}_1\langle q\rangle$, and then by case analysis on the last step of the derivation that $\mathtt{C}_1 \in \mathbb{X}^{\vartheta\cup\{x\}}$. The interesting cases are the rules ESubLStr, ESubsR, and EAppRStr:

1. ESubLStr, $\mathtt{C}_1 = \mathtt{C}_{11}[y\backslash t] \in \mathbb{X}^{\vartheta\cup\{x\}}$ with $t \in \mathsf{S}_{\vartheta\cup\{x\}}$ and $\mathtt{C}_{11} \in \mathbb{X}^{\vartheta\cup\{x,y\}}$ We consider two cases, depending on whether x is a structural variable in $\mathtt{C}_{11}$:

   1.1 If $x \in \mathsf{sv}(\mathtt{C}_{11})$ Then by *i.h.* there is a context $\mathtt{C}_{21} \in \mathbb{X}^{\vartheta\cup\{y\}}$ such that $\mathtt{C}_{11}\langle q\rangle = \mathtt{C}_{21}\langle\!\langle x\rangle\!\rangle$. We consider two further subcases, depending on whether y is a structural variable in $\mathtt{C}_{21}$:

      1.1.1 If $y \in \mathsf{sv}(\mathtt{C}_{21})$ We consider two more cases, depending on whether x is garbage in the structure t:

         1.1.1.1 If $x \in \mathsf{ngv}(t)$ Since $y \in \mathsf{sv}(\mathtt{C}_{21})$ we may apply the *i.h.* again, to obtain that there exists a context $\mathtt{C}_{31} \in \mathbb{X}^{\vartheta}$ such that $\mathtt{C}_{21}\langle\!\langle x\rangle\!\rangle = \mathtt{C}_{31}\langle\!\langle y\rangle\!\rangle$. Note that we are able to apply the *i.h.* since the term $\mathtt{C}_{21}\langle\!\langle x\rangle\!\rangle = \mathtt{C}_{11}\langle q\rangle$ is smaller than the original term, namely $\mathtt{C}_1\langle q\rangle = \mathtt{C}_{11}\langle q\rangle[y\backslash t]$.
         Since non-garbage variables are below evaluation contexts (Lem. A.25) and $t \in \mathsf{S}_{\vartheta\cup\{x\}}$ we know that there exists a context $\mathtt{C}_{22} \in \mathsf{E}^{\circ}_{\vartheta}$ such that $t = \mathtt{C}_{22}\langle\!\langle x\rangle\!\rangle$. So we have $\mathtt{C}_1\langle q\rangle = \mathtt{C}_{11}\langle q\rangle[y\backslash t] = \mathtt{C}_{11}\langle q\rangle[y\backslash\mathtt{C}_{22}\langle\!\langle x\rangle\!\rangle] = \mathtt{C}_{21}\langle\!\langle x\rangle\!\rangle[y\backslash\mathtt{C}_{22}\langle\!\langle x\rangle\!\rangle] = \mathtt{C}_{31}\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{C}_{22}\langle\!\langle x\rangle\!\rangle]$ with $\mathtt{C}_{31} \in \mathbb{X}^{\vartheta}$ and $\mathtt{C}_{22} \in \mathsf{E}^{\circ}_{\vartheta}$. By applying the rule for building evaluation contexts by going inside substitutions (ESubsR), we have that $\mathtt{C}_{31}\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{C}_{22}] \in \mathbb{X}^{\vartheta}$.

         1.1.1.2 If $x \notin \mathsf{ngv}(t)$ By the fact that garbage variables are not needed in "ϑ" (Lem. A.25) we have that $t \in \mathsf{S}_{\vartheta}$. So $\mathtt{C}_1\langle q\rangle = \mathtt{C}_{11}\langle q\rangle[y\backslash t] = \mathtt{C}_{21}\langle\!\langle x\rangle\!\rangle[y\backslash t]$ with $\mathtt{C}_{21} \in \mathbb{X}^{\vartheta\cup\{y\}}$ and $t \in \mathsf{S}_{\vartheta}$. By applying the rule for building evaluation contexts with structural substitutions (ESubLStr), we have that $\mathtt{C}_{21}[y\backslash t] \in \mathbb{X}^{\vartheta}$.

      1.1.2 If $y \notin \mathsf{sv}(\mathtt{C}_{21})$ Then since non-structural variables are not required in "ϑ" (Lem. A.54), $\mathtt{C}_{21} \in \mathbb{X}^{\vartheta}$. So $\mathtt{C}_1\langle q\rangle = \mathtt{C}_{11}\langle q\rangle[y\backslash t] = \mathtt{C}_{21}\langle\!\langle x\rangle\!\rangle[y\backslash t]$ with $\mathtt{C}_{21} \in \mathbb{X}^{\vartheta}$. By the fact that adding an arbitrary substitution preserves evaluation contexts (Lem. A.26), we have $\mathtt{C}_{21}[y\backslash t] \in \mathbb{X}^{\vartheta}$, as required.

   1.2 If $x \notin \mathsf{sv}(\mathtt{C}_{11})$ Recall that, by hypothesis, $x \in \mathsf{sv}(\mathtt{C}_1) = \mathsf{sv}(\mathtt{C}_{11}[y\backslash t])$ and that by definition of structural variables: $\mathsf{sv}(\mathtt{C}_{11}[y\backslash t]) = \mathsf{sv}(\mathtt{C}_{11}) \cup A$ where $A = \mathsf{ngv}(t)$ if $y \in \mathsf{sv}(\mathtt{C}_{11})$, and $A = \varnothing$ otherwise. Since $x \notin \mathsf{sv}(\mathtt{C}_{11})$, we must have that $y \in \mathsf{sv}(\mathtt{C}_{11})$ and $x \in \mathsf{ngv}(t)$.

   By applying the lemma that non-structural variables are not required in "ϑ" (Lem. A.54) on the fact that $\mathtt{C}_{11} \in \mathbb{X}^{\vartheta\cup\{x,y\}}$ we have $\mathtt{C}_{11} \in \mathbb{X}^{\vartheta\cup\{y\}}$. Since $y \in \mathsf{sv}(\mathtt{C}_{11})$, by the *i.h.* we have that there exists a context $\mathtt{C}_{21} \in \mathbb{X}^{\vartheta}$ such that $\mathtt{C}_{11}\langle q\rangle = \mathtt{C}_{21}\langle\!\langle y\rangle\!\rangle$. Moreover, $x \in \mathsf{ngv}(t)$, so since non-garbage variables are below evaluation contexts (Lem. A.25), there exists a context $\mathtt{C}_{22} \in \mathsf{E}^{\circ}_{\vartheta}$ such that $t = \mathtt{C}_{22}\langle\!\langle x\rangle\!\rangle$. So we have

$\mathtt{C}_1\langle q\rangle = \mathtt{C}_{11}\langle q\rangle[y\backslash t] = \mathtt{C}_{21}\langle\!\langle y\rangle\!\rangle[y\backslash t] = \mathtt{C}_{21}\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{C}_{22}\langle\!\langle x\rangle\!\rangle]$ with $\mathtt{C}_{21} \in \mathbb{X}^{\vartheta}$ and $\mathtt{C}_{22} \in \mathsf{E}^{\circ}_{\vartheta}$. By applying the rule for building evaluation contexts by going inside substitutions (ESubsR), we obtain that $\mathtt{C}_{21}\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{C}_{22}] \in \mathbb{X}^{\vartheta}$, as required.

2. ESubsR, $\mathtt{C}_1 = \mathtt{C}_{11}\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{C}_{12}] \in \mathbb{X}^{\vartheta\cup\{x\}}$ with $\mathtt{C}_{11} \in \mathbb{X}^{\vartheta\cup\{x\}}$ and $\mathtt{C}_{12} \in \mathsf{E}^{\circ}_{\vartheta\cup\{x\}}$ Then $\mathsf{sv}(\mathtt{C}_1) = \mathsf{sv}(\mathtt{C}_{11}\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{C}_{12}]) = (\mathsf{sv}(\mathtt{C}_{11})\backslash\{y\}) \cup \mathsf{sv}(\mathtt{C}_{12})$. Observe that $x \neq y$ by the variable convention. We consider two cases, depending on whether x is a structural variable in $\mathtt{C}_{11}$:

   2.1 If $x \in \mathsf{sv}(\mathtt{C}_{11})$ Then by *i.h.* there is a context $\mathtt{C}_{21} \in \mathbb{X}^{\vartheta}$ such that $\mathtt{C}_{11}\langle\!\langle y\rangle\!\rangle = \mathtt{C}_{21}\langle\!\langle x\rangle\!\rangle$. By the fact that adding an arbitrary substitution preserves evaluation contexts (Lem. A.26), we obtain that $\mathtt{C}_{21}[y\backslash \mathtt{C}_{12}\langle q\rangle] \in \mathbb{X}^{\vartheta}$, as required.

   2.2 If $x \notin \mathsf{sv}(\mathtt{C}_{11})$ Since non-structural variables are not required in "ϑ" (Lem. A.54), $\mathtt{C}_{11} \in \mathbb{X}^{\vartheta}$. Moreover, it must be the case that $x \in \mathsf{sv}(\mathtt{C}_{12})$, so by *i.h.* we have that there is a context $\mathtt{C}_{22} \in \mathsf{E}^{\circ}_{\vartheta}$ such that $\mathtt{C}_{12}\langle q\rangle = \mathtt{C}_{22}\langle\!\langle x\rangle\!\rangle$. By applying the formation rule for generalized evaluation contexts going inside substitutions (ESubsR) we conclude that $\mathtt{C}_{11}\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{C}_{22}] \in \mathbb{X}^{\vartheta}$, as required.

3. EAppRStr, $\mathtt{C}_1 = t\,\mathtt{C}_{11} \in \mathbb{X}^{\vartheta\cup\{x\}}$ with $t \in \mathsf{S}_{\vartheta\cup\{x\}}$ and $\mathtt{C}_{11} \in \mathsf{E}_{\vartheta\cup\{x\}}$ Then $\mathsf{sv}(\mathtt{C}_1) = \mathsf{ngv}(t) \cup \mathsf{sv}(\mathtt{C}_{11})$ We consider two cases, depending on whether x is non-garbage in t:

   3.1 If $x \in \mathsf{ngv}(t)$ Since non-garbage variables are below evaluation contexts (Lem. A.25), the structure t can be written as of the form $\mathtt{C}_{21}\langle\!\langle x\rangle\!\rangle$, with $\mathtt{C}_{21} \in \mathsf{E}^{\circ}_{\vartheta}$. By applying the formation rule for generalized evaluation contexts going to the left of an application (EAppL) we conclude that $\mathtt{C}_{21}\,\mathtt{C}_{11}\langle q\rangle \in \mathsf{E}^{\circ}_{\vartheta}$.
   If $\mathbb{X}^{\vartheta}$ is $\mathsf{E}^{\circ}_{\vartheta}$, we are done. If $\mathbb{X}^{\vartheta}$ is E_{ϑ}, we are also done, since $\mathsf{E}^{\circ}_{\vartheta} \subseteq \mathsf{E}_{\vartheta}$.

   3.2 If $x \notin \mathsf{ngv}(t)$ Since garbage variables are not required in "ϑ" (Lem. A.25), $t \in \mathsf{S}_{\vartheta}$. Moreover, $x \in \mathsf{sv}(\mathtt{C}_{11})$, so by *i.h.* there must exist a context $\mathtt{C}_{21} \in \mathsf{E}_{\vartheta}$ such that $\mathtt{C}_{11}\langle q\rangle = \mathtt{C}_{21}\langle\!\langle x\rangle\!\rangle$. By applying the formation rule for generalized evaluation contexts going to the right of a structure (EAppRStr) we conclude that $t\,\mathtt{C}_{21} \in \mathbb{X}^{\vartheta}$ as required.

□

The following result will be useful many times throughout the proofs in the remainder of this section. We call it a "proof tactic", rather than a "lemma", following the nomenclature usual in proof assistants such as Coq. The exact way in which this result has to be instantiated in each case may vary slightly.

Tactic A.55 (Strengthening ϑ)**.** *Consider a* $\mathtt{lsv}$ *step* $\mathtt{C}\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{vL}] \to \mathtt{C}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\mathtt{L}$.

1. **Strengthening ϑ for normal forms.** *Let* $\mathbb{X}^{\vartheta}$ *stand for either the set* $\mathtt{nf}\vartheta$ *or the set* S_{ϑ}. *If* $\mathtt{C}\langle\!\langle x\rangle\!\rangle \in \mathbb{X}^{\hat{\vartheta}}$, *where* $\hat{\vartheta} = \mathsf{fz}^{\vartheta}([x\backslash \mathtt{v}]\mathtt{L})$ *then* $\mathtt{C}\langle\!\langle x\rangle\!\rangle \in \mathbb{X}^{\vartheta}$.

2. **Strengthening ϑ for evaluation contexts.** *Let* $\mathbb{X}^{\vartheta}$ *stand for either the set* E_{ϑ} *or the set* $\mathsf{E}^{\circ}_{\vartheta}$. *If* $\mathtt{C}[x\backslash \mathtt{v}]\mathtt{L} \in \mathbb{X}^{\vartheta}$ *then* $\mathtt{C}\langle\!\langle x\rangle\!\rangle \in \mathbb{X}^{\vartheta}$ *and* $\mathtt{C}\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{vL}] \in \mathbb{X}^{\vartheta}$.

Proof. For the first item, note that $x \notin \hat{\vartheta}$ since x is bound to an answer, so it is not frozen. So we have that $\hat{\vartheta} \subseteq \vartheta \cup \mathsf{domL}$. The variables in domL do not occur free in the term $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ by Barendregt's convention, since $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ is outside the scope of $\mathtt{L}$ on the left-hand side of the $\mathtt{lsv}$ step. In particular, all the variables in domL are garbage variables in the term $\mathtt{C}\langle\!\langle x \rangle\!\rangle$. Hence, by repeatedly applying the fact that garbage variables are not required in "ϑ" (Lem. A.25), we obtain that $\mathtt{C}\langle\!\langle x \rangle\!\rangle \in \mathbb{X}^{\vartheta}$.

For the second item, by the decomposition of evaluation contexts lemma (Lem. A.20) the context $\mathtt{C}$ must be an evaluation context in $\mathbb{X}^{\hat{\vartheta}}$ where $\hat{\vartheta} = \mathsf{fz}^{\vartheta}([x\backslash \mathtt{v}]\mathtt{L})$. Note that $x \notin \hat{\vartheta}$ since x is bound to an answer, so it is not frozen. So we have that $\hat{\vartheta} \subseteq \vartheta \cup \mathsf{domL}$. Note that the variables in domL do not occur free in the context $\mathtt{C}$ by Barendregt's convention, since $\mathtt{C}$ is outside the scope of $\mathtt{L}$ on the left-hand side of the $\mathtt{lsv}$ step. In particular, all the variables in domL are not structural variables in the context $\mathtt{C}$. Hence, by repeatedly applying the fact that non-structural variables are not required in "ϑ" (Lem. A.54), we obtain that $\mathtt{C} \in \mathbb{X}^{\vartheta}$. Moreover, since adding arbitrary substitutions preserves evaluation contexts (Lem. A.26), $\mathtt{C}[x\backslash \mathtt{vL}] \in \mathbb{X}^{\vartheta}$, as required. □

Critical contexts

Consider a context like $\mathtt{C} = y[y\backslash z\Box]$ and the set of frozen variables $\vartheta = \{z\}$. Note that $\mathtt{C}\langle I \rangle = y[y\backslash z\, I]$ is a ϑ-normal form since $z\, I$ is a ϑ-structure. On the other hand, $\mathtt{C}\langle\!\langle x \rangle\!\rangle = y[y\backslash z\, x]$ is **not** a ϑ-normal form because $z\, x$ is not a ϑ-structure, as $x \notin \vartheta$. Remark that $\mathtt{C}$ is a ϑ-evaluation context. In the following Lem. A.57 we show that this is not just a coincidence. Indeed, we will show that if a context is such that $\mathtt{C}\langle t \rangle$ is a normal form but $\mathtt{C}\langle x \rangle$ is not a normal form, then $\mathtt{C}$ must be an evaluation context.

This result will be a useful tool to prove that normal forms and evaluation contexts are backward stable by internal steps. For example, if $\mathtt{C}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{v}] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle \mathtt{v} \rangle[x\backslash \mathtt{v}]$ is an internal step and the right-hand side is a normal form, then $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ must be a normal form. Otherwise we would have that $\mathtt{C}\langle \mathtt{v} \rangle$ is a normal form while $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ is not, hence $\mathtt{C}$ would be an evaluation context, contradicting the fact that the original step was internal.

Definition A.56 (Critical contexts)**.** Let $\mathbb{X}^{\vartheta}$ be a set of terms depending on a set of variables ϑ. A context $\mathtt{C}$ is said to be $\mathbb{X}^{\vartheta}$*-critical* if the following conditions hold:

1. $\mathtt{C}\langle q \rangle \in \mathbb{X}^{\vartheta}$ for some term q; and
2. $\mathtt{C}\langle\!\langle x \rangle\!\rangle \notin \mathbb{X}^{\vartheta}$ for some variable $x \notin \vartheta$ that is not bound by $\mathtt{C}$.

Lemma A.57 (Critical contexts are evaluation contexts)**.** *The following inclusions between sets hold:*

1. *The set of* $\mathtt{nf}\vartheta$*-critical contexts is included in* E_{ϑ}*.*
2. *The set of* S_{ϑ}*-critical contexts is included in* $\mathsf{E}^{\circ}_{\vartheta}$*.*

Proof. Let $\mathbb{X}^{\vartheta}$ denote the set $\mathtt{nf}\vartheta$ (resp. S_{ϑ}), and let $\mathbb{Y}^{\vartheta}$ denote the set E_{ϑ} (resp. $\mathsf{E}^{\circ}_{\vartheta}$). Suppose that $\mathtt{C}$ is a $\mathbb{X}^{\vartheta}$-critical context, and let us show that $\mathtt{C} \in \mathbb{Y}^{\vartheta}$. Since $\mathtt{C}$ is $\mathbb{X}^{\vartheta}$-critical, there is a

term q and a variable x not bound by $\mathtt{C}$ such that $\mathtt{C}\langle q\rangle \in \mathbb{X}^{\vartheta}$ and $\mathtt{C}\langle\!\langle x\rangle\!\rangle \notin \mathbb{X}^{\vartheta}$. We proceed by induction on the derivation that $\mathtt{C}\langle q\rangle \in \mathbb{X}^{\vartheta}$. The interesting cases are rules nfSubG and nfSub:

1. nfSubG, $\mathtt{C}\langle q\rangle = t[y\backslash s] \in \mathbb{X}^{\vartheta}$ with $t \in \mathbb{X}^{\vartheta}$ and $y \notin \mathsf{ngv}(t)$ Let us check that $\mathtt{C} \in \mathbb{Y}^{\vartheta}$. If $\mathtt{C}$ is empty, *i.e.* $\mathtt{C} = \Box$, we trivially have $\mathtt{C} \in \mathbb{Y}^{\vartheta}$. Otherwise, $\mathtt{C}$ is non-empty and there are two possibilities:

   1.1 The hole of $\mathtt{C}$ is to the left, *i.e.* $\mathtt{C} = \mathtt{C}'[y\backslash s]$ Then $\mathtt{C}'\langle q\rangle \in \mathbb{X}^{\vartheta}$ by formation of $\mathtt{C}\langle q\rangle \in \mathbb{X}^{\vartheta}$. Moreover we claim that $\mathtt{C}'\langle\!\langle x\rangle\!\rangle \notin \mathbb{X}^{\vartheta}$. To see this, note that the fact that $y \notin \mathsf{ngv}(\mathtt{C}'\langle q\rangle) \cup \{x\}$ implies that $y \notin \mathsf{ngv}(\mathtt{C}'\langle\!\langle x\rangle\!\rangle)$. So, by contradiction, if we suppose $\mathtt{C}'\langle\!\langle x\rangle\!\rangle \notin \mathbb{X}^{\vartheta}$ we can apply the same formation rule and obtain that $\mathtt{C}'\langle\!\langle x\rangle\!\rangle[y\backslash s] \in \mathbb{X}^{\vartheta}$, contradicting the hypothesis that $\mathtt{C}\langle\!\langle x\rangle\!\rangle \notin \mathbb{X}^{\vartheta}$.
   Therefore we are able to apply the *i.h.* on the facts that $\mathtt{C}'\langle q\rangle \in \mathbb{X}^{\vartheta}$ and $\mathtt{C}'\langle\!\langle x\rangle\!\rangle \notin \mathbb{X}^{\vartheta}$ to conclude that $\mathtt{C}' \in \mathbb{Y}^{\vartheta}$. This in turn implies that $\mathtt{C}'[y\backslash s] \in \mathbb{Y}^{\vartheta}$ since adding an arbitrary substitution preserves evaluation contexts (Lem. A.26).

   1.2 The hole of $\mathtt{C}$ is to the right, *i.e.* $\mathtt{C} = t[y\backslash \mathtt{C}']$ This case is not possible, since $t \in \mathbb{X}^{\vartheta}$ and $y \notin \mathsf{ngv}(t)$ by formation, and this implies that $t[y\backslash \mathtt{C}'\langle\!\langle x\rangle\!\rangle] \in \mathbb{X}^{\vartheta}$, contradicting the hypothesis that $\mathtt{C}\langle\!\langle x\rangle\!\rangle \notin \mathbb{X}^{\vartheta}$.

2. nfSub, $\mathtt{C}\langle q\rangle = t[y\backslash M^{\vartheta}] \in \mathbb{X}^{\vartheta}$ with $t \in \mathbb{X}^{\vartheta\cup\{y\}}$ and $M^{\vartheta} \in \mathsf{S}_{\vartheta}$ Let us check that $\mathtt{C} \in \mathbb{Y}^{\vartheta}$. If $\mathtt{C}$ is empty, *i.e.* $\mathtt{C} = \Box$, we trivially have $\mathtt{C} \in \mathbb{Y}^{\vartheta}$. Otherwise, $\mathtt{C}$ is non-empty and there are two possibilities:

   2.1 The hole of $\mathtt{C}$ is to the left, *i.e.* $\mathtt{C} = \mathtt{C}'[y\backslash M^{\vartheta}]$ Then $\mathtt{C}'\langle q\rangle \in \mathbb{X}^{\vartheta\cup\{y\}}$ by formation. Moreover, we claim that $\mathtt{C}'\langle\!\langle x\rangle\!\rangle \notin \mathbb{X}^{\vartheta\cup\{x\}}$. By contradiction, suppose that $\mathtt{C}'\langle\!\langle x\rangle\!\rangle \in \mathbb{X}^{\vartheta\cup\{x\}}$. Then $\mathtt{C}'\langle\!\langle x\rangle\!\rangle[y\backslash M^{\vartheta}] \in \mathbb{X}^{\vartheta}$, contradicting the hypothesis that $\mathtt{C}\langle\!\langle x\rangle\!\rangle \notin \mathbb{X}^{\vartheta}$.
   So by *i.h.* we obtain that $\mathtt{C}' \in \mathbb{Y}^{\vartheta\cup\{x\}}$ and, applying the context forming rule for structural substitutions (ESubLStr), we get $\mathtt{C}'[y\backslash M^{\vartheta}] \in \mathbb{Y}^{\vartheta}$, that is to say $\mathtt{C} \in \mathbb{Y}^{\vartheta}$, as required.

   2.2 The hole of $\mathtt{C}$ is to the right, *i.e.* $\mathtt{C} = t[y\backslash \mathtt{C}']$ Then $t \in \mathbb{X}^{\vartheta\cup\{y\}}$ and $y \in \mathsf{ngv}(t)$ by formation. This implies that $t = \mathtt{C}_1\langle\!\langle y\rangle\!\rangle$ with $\mathtt{C}_1 \in \mathbb{Y}^{\vartheta}$ by Lem. A.25
   Note that $\mathtt{C}'\langle q\rangle = M^{\vartheta} \in \mathsf{S}_{\vartheta}$. Moreover, we claim that $\mathtt{C}'\langle\!\langle x\rangle\!\rangle \notin \mathsf{S}_{\vartheta}$. By contradiction, suppose that $\mathtt{C}'\langle\!\langle x\rangle\!\rangle \in \mathsf{S}_{\vartheta}$. Then $t[y\backslash \mathtt{C}'\langle\!\langle x\rangle\!\rangle] \in \mathbb{X}^{\vartheta}$, contradicting the hypothesis that $\mathtt{C}\langle\!\langle x\rangle\!\rangle \notin \mathbb{X}^{\vartheta}$. So by *i.h.* we have that $\mathtt{C}' \in \mathsf{E}^{\circ}_{\vartheta}$.
   Combining the facts that $\mathtt{C}_1 \in \mathbb{Y}^{\vartheta}$ and $\mathtt{C}' \in \mathsf{E}^{\circ}_{\vartheta}$, by applying the formation rule for generalized evaluation contexts going inside substitutions, we conclude that $\mathtt{C}_1\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{C}'] \in \mathbb{Y}^{\vartheta}$, as required.

□

Replacing a value by a variable in an evaluation context

To prove that internal steps can be postponed we need to deal with situations such as $\mathtt{C}\langle t, x\rangle[x\backslash \mathtt{v}] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle t, \mathtt{v}\rangle[x\backslash \mathtt{v}] \overset{\vartheta}{\rightsquigarrow} \mathtt{C}\langle t', \mathtt{v}\rangle$, where $\mathtt{C}$ is a two-hole context. Note that $\mathtt{C}\langle\Box, \mathtt{v}\rangle$ is an evalua-

tion context since the second step is external. To postpone the internal step we would like that $\mathtt{C}\langle\Box, x\rangle$ is also an evaluation context. Unfortunately this is not always the case. As an example, consider the two-hole context $\mathtt{C} = (z\,\Box_1)[z\backslash y\,\Box_2]$ with $\vartheta = \{y\}$ and note that $\mathtt{C}\langle\Box, I\rangle = (z\,\Box)[z\backslash y\,I]$ is a $\{y\}$-evaluation context, since z is bound to a strong $\{y\}$-structure, but $\mathtt{C}\langle\Box, x\rangle = (z\,\Box)[z\backslash y\,x]$ is not a $\{y\}$-evaluation context, since z is bound to $y\,x$, which is not a strong $\{y\}$-structure. In such a situation, evaluation should focus on x, that is, what we *do* have is that $\mathtt{C}\langle t, \Box\rangle = (z\,t)[z\backslash y\,\Box]$ is a $\{y\}$-evaluation context. The following lemma deals with this situation in full generality.

Lemma A.58 (Replacing a value by a variable in an evaluation context)**.** *Let $\widehat{\mathtt{C}}$ be a two-hole context, $x \notin \vartheta$ a variable, $\mathtt{v}$ any value, and q be a term such that x is not bound by $\widehat{\mathtt{C}}\langle q, \Box\rangle$. If $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle \in \mathbb{X}^\vartheta$ then either $\widehat{\mathtt{C}}\langle\Box, x\rangle \in \mathbb{X}^\vartheta$ (***left branch***) or $\widehat{\mathtt{C}}\langle q, \Box\rangle \in \mathbb{X}^\vartheta$ (***right branch***) where $\mathbb{X}^\vartheta$ is either the set E_ϑ or the set $\mathsf{E}^\circ_\vartheta$.*

Proof. Let us write $\Box$ and $\boxtimes$ to distinguish the two holes of $\widehat{\mathtt{C}}$. The proof goes by induction on the derivation that $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle$ is a generalized evaluation context over ϑ.

1. EBox, $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle = \Box \in \mathbb{X}^\vartheta$ Impossible, as $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle$ must contain a value $\mathtt{v}$ as a subterm.

2. EAppL, $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle = \mathtt{I}^\vartheta\,t \in \mathbb{X}^\vartheta$ with $\mathtt{I}^\vartheta \in \mathsf{E}^\circ_\vartheta$ If the value $\mathtt{v}$ is inside t, *i.e.* $\widehat{\mathtt{C}}\langle\Box, \boxtimes\rangle = \mathtt{I}^\vartheta\mathtt{C}\langle\boxtimes\rangle$ then the left branch of the disjunction holds as $\mathtt{I}^\vartheta\mathtt{C}\langle x\rangle \in \mathbb{X}^\vartheta$.
   Otherwise, the value $\mathtt{v}$ is inside $\mathtt{I}^\vartheta$, *i.e.* there is a two-hole context $\widehat{\mathtt{C}}_1$ such that $\widehat{\mathtt{C}}\langle\Box, \boxtimes\rangle = \widehat{\mathtt{C}}_1\langle\Box, \boxtimes\rangle\,t$ and $\widehat{\mathtt{C}}_1\langle\Box, \mathtt{v}\rangle = \mathtt{I}^\vartheta \in \mathsf{E}^\circ_\vartheta$. Then it is straightforward to conclude by *i.h.*.

3. ESubLNonStr, $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle = F^\vartheta[y\backslash t] \in \mathbb{X}^\vartheta$ where $F^\vartheta \in \mathbb{X}^\vartheta$ and $t \notin \mathsf{S}_\vartheta$ If the value $\mathtt{v}$ is inside t, *i.e.* $\widehat{\mathtt{C}}\langle\Box, \boxtimes\rangle = F^\vartheta[y\backslash\mathtt{C}\langle\boxtimes\rangle]$, we may apply the fact that adding an arbitrary substitution preserves evaluation contexts (Lem. A.26), obtaining that $F^\vartheta[y\backslash\mathtt{C}\langle x\rangle] \in \mathbb{X}^\vartheta$, that is $\widehat{\mathtt{C}}\langle\Box, x\rangle \in \mathbb{X}^\vartheta$ and the left branch of the disjunction holds. Otherwise the value $\mathtt{v}$ is inside F^ϑ, *i.e.* there is a two-hole context $\widehat{\mathtt{C}}_1$ such that $\widehat{\mathtt{C}}\langle\Box, \boxtimes\rangle = \widehat{\mathtt{C}}_1\langle\Box, \boxtimes\rangle[y\backslash t]$ and $\widehat{\mathtt{C}}_1\langle\Box, \mathtt{v}\rangle = F^\vartheta \in \mathbb{X}^\vartheta$. Then it is straightforward to conclude by *i.h.*.

4. ESubLStr, $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle = F^{\vartheta\cup\{y\}}[y\backslash M^\vartheta] \in \mathbb{X}^\vartheta$ where $F^{\vartheta\cup\{y\}} \in \mathbb{X}^{\vartheta\cup\{y\}}$ and $M^\vartheta \in \mathsf{S}_\vartheta$ If the value $\mathtt{v}$ is inside M^ϑ, *i.e.* $M^\vartheta = \mathtt{C}\langle\mathtt{v}\rangle$, we consider two further subcases, depending on whether $\mathtt{C}\langle\!\langle x\rangle\!\rangle$ is a strong ϑ-structure:

   4.1 If $\mathtt{C}\langle\!\langle x\rangle\!\rangle \in \mathsf{S}_\vartheta$ Applying the formation rule for generalized ϑ-evaluation contexts, using a structural substitution (ESubLStr), we conclude that $F^{\vartheta\cup\{y\}}[y\backslash\mathtt{C}\langle\!\langle x\rangle\!\rangle] \in \mathbb{X}^\vartheta$, that is $\widehat{\mathtt{C}}\langle\Box, x\rangle \in \mathbb{X}^\vartheta$, and the left branch of the disjunction holds.

   4.2 If $\mathtt{C}\langle\!\langle x\rangle\!\rangle \notin \mathsf{S}_\vartheta$ Then, since $\mathtt{C}\langle\mathtt{v}\rangle \in \mathsf{S}_\vartheta$ but $\mathtt{C}\langle\!\langle x\rangle\!\rangle \notin \mathsf{S}_\vartheta$, we have that $\mathtt{C}$ is S_ϑ-critical. By Lem. A.57 we have that every S_ϑ-critical context is a $\mathsf{E}^\circ_\vartheta$ context, so $\mathtt{C} \in \mathsf{E}^\circ_\vartheta$.
   We consider two further subcases, depending on whether y is a structural variable in $F^{\vartheta\cup\{y\}}$:

      4.2.1 If $y \in \mathsf{sv}(F^{\vartheta\cup\{y\}})$ Since structural variables are below evaluation contexts (Lem. A.54), there is a context $F_2^\vartheta \in \mathbb{X}^\vartheta$ such that $F^{\vartheta\cup\{y\}}\langle q\rangle = F_2^\vartheta\langle\!\langle y\rangle\!\rangle$. This means that $F^{\vartheta\cup\{y\}}\langle q\rangle[y\backslash\mathtt{C}] = F_2^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{C}] \in \mathbb{X}^\vartheta$ since $\mathtt{C} \in \mathsf{E}^\circ_\vartheta$ is a inert context. So the right branch holds.

4.2.2 If $y \notin \mathsf{sv}(F^{\vartheta\cup\{y\}})$ Since non-structural variables are not required in "ϑ" (Lem. A.54), $F^{\vartheta\cup\{y\}} \in \mathbb{X}^{\vartheta}$. Since adding an arbitrary substitution preserves evaluation contexts (Lem. A.26), we conclude that $F^{\vartheta\cup\{y\}}[y\backslash \mathtt{C}\langle\!\langle x \rangle\!\rangle] \in \mathbb{X}^{\vartheta}$, and the left branch holds.

Otherwise, the value $\mathtt{v}$ is inside $F^{\vartheta\cup\{y\}}$, *i.e.* there is a two-hole context $\widehat{\mathtt{C}}_1$ such that $\widehat{\mathtt{C}}\langle\Box,\boxtimes\rangle = \widehat{\mathtt{C}}_1\langle\Box,\boxtimes\rangle[x\backslash M^{\vartheta}]$ and $\widehat{\mathtt{C}}_1\langle\Box,\mathtt{v}\rangle = F^{\vartheta\cup\{y\}} \in \mathbb{X}^{\vartheta\cup\{y\}}$. Then it is straightforward to conclude by *i.h.*.

5. ESubsR, $\widehat{\mathtt{C}}\langle\Box,\mathtt{v}\rangle = F^{\vartheta}\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{I}^{\vartheta}] \in \mathbb{X}^{\vartheta}$ with $F^{\vartheta} \in \mathbb{X}^{\vartheta}$ and $\mathtt{I}^{\vartheta} \in \mathsf{E}^{\circ}_{\vartheta}$ If the value $\mathtt{v}$ is inside $F^{\vartheta}\langle\!\langle y \rangle\!\rangle$, there are two cases, depending on whether the hole of F^{ϑ} lies inside the value $\mathtt{v}$ or not:

   5.1 If the hole of F^{ϑ} lies inside $\mathtt{v}$ Then $F^{\vartheta} = \mathtt{C}_1\langle\lambda z.\mathtt{C}_2\rangle$ where $\mathtt{v} = \lambda z.\mathtt{C}_2\langle\!\langle y \rangle\!\rangle$ and $\widehat{\mathtt{C}}\langle\Box,\boxtimes\rangle = \mathtt{C}_1\langle\boxtimes\rangle[y\backslash \mathtt{I}^{\vartheta}\langle\Box\rangle]$. By the decomposition of evaluation contexts lemma (Lem. A.20) we have that $\mathtt{C}_1 \in \mathbb{X}^{\vartheta}$. By the fact that adding an arbitrary substitution preserves evaluation contexts (Lem. A.26), $\mathtt{C}_1[x\backslash \mathtt{I}^{\vartheta}\langle q\rangle] \in \mathbb{X}^{\vartheta}$. This means that $\widehat{\mathtt{C}}\langle q,\Box\rangle = \mathtt{C}_1[x\backslash \mathtt{I}^{\vartheta}\langle q\rangle] \in \mathbb{X}^{\vartheta}$, so the right branch holds.

   5.2 If the hole of F^{ϑ} and the position of $\mathtt{v}$ are disjoint Then there is a two-hole context $\widehat{\mathtt{C}}_1$ such that $\widehat{\mathtt{C}}_1\langle\Box,\mathtt{v}\rangle = F^{\vartheta}$. Note, in particular, that $\widehat{\mathtt{C}}_1\langle y,\mathtt{v}\rangle = F^{\vartheta}\langle\!\langle y \rangle\!\rangle$, and $\widehat{\mathtt{C}}\langle\Box,\boxtimes\rangle = \widehat{\mathtt{C}}_1\langle y,\boxtimes\rangle[y\backslash \mathtt{I}^{\vartheta}\langle\Box\rangle]$. By *i.h.* there are two possibilities. Then it is straightforward to conclude by *i.h.*; using Lem. A.26 in the right branch case.

   Otherwise, the value $\mathtt{v}$ is inside $\mathtt{I}^{\vartheta}$. This means that there is a two-hole context $\widehat{\mathtt{C}}_1$ such that $\widehat{\mathtt{C}} = F^{\vartheta}\langle\!\langle y \rangle\!\rangle[y\backslash \widehat{\mathtt{C}}_1]$ and $\widehat{\mathtt{C}}_1\langle\Box,\mathtt{v}\rangle = \mathtt{I}^{\vartheta}$. Then it is straightforward to conclude by *i.h.*.

6. EAppRStr, $\widehat{\mathtt{C}}\langle\Box,\mathtt{v}\rangle = M^{\vartheta}\, F^{\vartheta} \in \mathbb{X}^{\vartheta}$, with $M^{\vartheta} \in \mathsf{S}_{\vartheta}$ and $F^{\vartheta} \in \mathsf{E}_{\vartheta}$ If the value $\mathtt{v}$ is inside M^{ϑ} *i.e.* $M^{\vartheta} = \mathtt{C}\langle\mathtt{v}\rangle$ and $\widehat{\mathtt{C}}\langle\Box,\boxtimes\rangle = \mathtt{C}\langle\boxtimes\rangle\, F^{\vartheta}\langle\Box\rangle$, we consider two further subcases, depending on whether $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ is a strong ϑ-structure:

   6.1 If $\mathtt{C}\langle\!\langle x \rangle\!\rangle \in \mathsf{S}_{\vartheta}$ Applying the formation rule for generalized ϑ-evaluation contexts, going to the right of a structure (EAppRStr), we conclude that $\widehat{\mathtt{C}}\langle\Box, x\rangle = \mathtt{C}\langle\!\langle x \rangle\!\rangle\, F^{\vartheta} \in \mathbb{X}^{\vartheta}$, so the left branch holds.

   6.2 If $\mathtt{C}\langle\!\langle x \rangle\!\rangle \notin \mathsf{S}_{\vartheta}$ Then, since $\mathtt{C}\langle\mathtt{v}\rangle \in \mathsf{S}_{\vartheta}$ but $\mathtt{C}\langle\!\langle x \rangle\!\rangle \notin \mathsf{S}_{\vartheta}$, we have that $\mathtt{C}$ is S_{ϑ}-critical. By Lem. A.57 we have that every S_{ϑ}-critical context is a $\mathsf{E}^{\circ}_{\vartheta}$ context, so $\mathtt{C} \in \mathsf{E}^{\circ}_{\vartheta}$. Applying the formation rule for generalized ϑ-evaluation contexts, going to the left of an application (EAppL), we conclude that $\widehat{\mathtt{C}}\langle q,\Box\rangle = \mathtt{C}\, F^{\vartheta}\langle q\rangle \in \mathbb{X}^{\vartheta}$, so the right branch holds.

   Otherwise, the value $\mathtt{v}$ is inside F^{ϑ}, that is, there is a two-hole context $\widehat{\mathtt{C}}_1$ such that $\widehat{\mathtt{C}} = M^{\vartheta}\,\widehat{\mathtt{C}}_1$ and $F^{\vartheta} = \widehat{\mathtt{C}}_1\langle\Box,\mathtt{v}\rangle$. Then it is straightforward to conclude by *i.h.*.

7. ELam, $\widehat{\mathtt{C}}\langle\Box,\mathtt{v}\rangle = \lambda y.F^{\vartheta\cup\{y\}} \in \mathsf{E}_{\vartheta}$ with $F^{\vartheta\cup\{y\}} \in \mathsf{E}_{\vartheta\cup\{y\}}$ Immediate by *i.h.*.

□

Stripping substitutions

Consider the following unification problem: if $\mathtt{C}$ is an evaluation context and we know that $\mathtt{C}\langle t\rangle = s\mathtt{L}$, then what is the shape of $\mathtt{C}$? We call this the problem of *stripping substitutions* out of an evaluation context. It might simply be the case that $\mathtt{C} = \mathtt{C}'\mathtt{L}$ where $\mathtt{C}'$ is in turn an evaluation context. But it may also be the case that $\mathtt{C}$ goes inside a substitution, for example, $\mathtt{C} = x[x\backslash y\Box]$ and $\mathtt{L} = [x\backslash yt]$. The relation between $\mathtt{C}$ and $\mathtt{L}$ can actually get quite hairy: $\mathtt{C}$ can take a number of "jumps" inside $\mathtt{L}$. For instance, $\mathtt{C} = x_1[x_1\backslash y[y\backslash x_2]][z\backslash u][x_2\backslash\Box]$ with $\mathtt{L} = [x_1\backslash y[y\backslash x_2]][z\backslash u][x_2\backslash t]$. We characterize the solution to this problem by defining an auxiliary sort of *chain contexts*. A chain context $\mathscr{L}$ is intuitively a context with two holes, and $t_1\mathscr{L}\{t_2\}$ stands for the result of plugging t_1 and t_2 in each of its holes. For instance, in the example above we would have $\Box_1\mathscr{L}\{\Box_2\} = \Box_1[x_1\backslash y[y\backslash x_2]][z\backslash t][x_2\backslash\Box_2]$. Thus $\mathtt{C}$ can be recovered as $s\mathscr{L}\{\Box\}$ and $\mathtt{L}$ can be recovered as $\Box\mathscr{L}\{t\}$.

Definition A.59 (Chain context)**.** The sets of (ϑ, x)-chain contexts, ranged over by $\mathscr{L}, \mathscr{L}'$, etc., are defined inductively with the two following rules:

$$\frac{\begin{array}{c}\vartheta' = \mathsf{fz}^{\vartheta}(\mathtt{L}_2)\\ \mathtt{C} \in \mathsf{E}^{\circ}_{\vartheta'} \text{ is a inert evaluation context}\\ \mathtt{L}_1, \mathtt{L}_2 \text{ are substitution contexts}\end{array}}{\langle \mathtt{L}_1, x, \mathtt{C}, \mathtt{L}_2\rangle \text{ is a } (\vartheta, x)\text{-chain context}} \qquad \frac{\begin{array}{c}\vartheta' = \mathsf{fz}^{\vartheta}(\mathtt{L})\\ \mathtt{C} \in \mathsf{E}^{\circ}_{\vartheta'} \text{ is a inert evaluation context}\\ \mathtt{L} \text{ is a substitution context}\\ \mathscr{L} \text{ is a } (\vartheta' \cup \{y\}, x)\text{-chain context}\end{array}}{\langle \mathscr{L}, y, \mathtt{C}, \mathtt{L}\rangle \text{ is a } (\vartheta, x)\text{-chain context}}$$

Given a (ϑ, x)-chain context $\mathscr{L}$, its instantiation on two terms t_1, t_2, written $t_1\mathscr{L}\{t_2\}$, is defined inductively as follows:

$$\begin{array}{rcl} t_1\langle \mathtt{L}_1, x, \mathtt{C}, \mathtt{L}_2\rangle\{t_2\} & \stackrel{\text{def}}{=} & t_1\mathtt{L}_1[x\backslash \mathtt{C}\langle t_2\rangle]\mathtt{L}_2 \\ t_1\langle \mathscr{L}, y, \mathtt{C}, \mathtt{L}\rangle\{t_2\} & \stackrel{\text{def}}{=} & (t_1\mathscr{L}\{y\})[y\backslash \mathtt{C}\langle t_2\rangle]\mathtt{L} \end{array}$$

Sometimes we write $\mathscr{L}^{\vartheta}_x$ to stress that $\mathscr{L}$ is a (ϑ, x)-chain context. The number of rules required to build a chain context $\mathscr{L}$ is called the number of *jumps* of $\mathscr{L}$.

Lemma A.60 (Weakening for chain contexts)**.** *If $\mathscr{L}$ is a (ϑ, x)-chain context, and $\vartheta \subseteq \vartheta'$ then $\mathscr{L}$ is a (ϑ', x)-chain context.*

Proof. By induction on the formation rules for chain contexts, using the weakening lemma for evaluation contexts (Lem. A.24). □

Definition A.61 (Adding substitutions to chain contexts)**.** If $\mathtt{L}$ is a substitution context, $\vartheta' = \mathsf{fz}^{\vartheta}(\mathtt{L})$ and $\mathscr{L}$ is a (ϑ', x)-chain context then we write $\mathscr{L}\mathtt{L}$ for the (ϑ, x)-chain context defined as follows:

1. $\langle \mathtt{L}_1, x, \mathtt{C}, \mathtt{L}_2\rangle\mathtt{L} \stackrel{\text{def}}{=} \langle \mathtt{L}_1, x, \mathtt{C}, \mathtt{L}_2\mathtt{L}\rangle$
2. $\langle \mathscr{L}', x, \mathtt{C}, \mathtt{L}'\rangle\mathtt{L} \stackrel{\text{def}}{=} \langle \mathscr{L}', x, \mathtt{C}, \mathtt{L}'\mathtt{L}\rangle$

Note that $t_1(\mathscr{L}\mathtt{L})\{t_2\} = (t_1\mathscr{L}\{t_2\})\mathtt{L}$.

Lemma A.62 (Stripping substitutions from a context using chain contexts)**.** *Let $F^\vartheta \in \mathbb{X}^\vartheta$ be a generalized evaluation context, where $\mathbb{X}^\vartheta$ stands for either E_ϑ or $\mathsf{E}^\circ_\vartheta$. Suppose that $F^\vartheta\langle t\rangle = s\mathtt{L}$ where all the substitution nodes in the spine of $\mathtt{L}$ belong to the context F^ϑ (rather than to the subterm t), that is, one of the following holds:*

- **A.** *$F^\vartheta = \mathtt{CL}$ and $s = \mathtt{C}\langle t\rangle$.*
- **B.** *$F^\vartheta = s\mathtt{L}_1[x\backslash\mathtt{C}]\mathtt{L}_2$ and $\mathtt{L} = \mathtt{L}_1[x\backslash\mathtt{C}\langle t\rangle]\mathtt{L}_2$.*

Then in each case the following more precise conditions hold:

- **A.** *There is an evaluation context $F_1^{\vartheta'} \in \mathbb{X}^{\vartheta'}$ where $\vartheta' = \mathsf{fz}^\vartheta(\mathtt{L})$ such that $F^\vartheta = F_1^{\vartheta'}\mathtt{L}$ and $s = F_1^{\vartheta'}\langle t\rangle$.*
- **B.** *There is an evaluation context $F_1^{\vartheta'} \in \mathbb{X}^{\vartheta'}$ where $\vartheta' = \mathsf{fz}^\vartheta(\mathtt{L})$, and a (ϑ, x)-chain context $\mathscr{L}$ such that $F^\vartheta = F_1^{\vartheta'}\langle\!\langle x\rangle\!\rangle\mathscr{L}\{\Box\}$ and $\mathtt{L} = \Box\mathscr{L}\{t\}$.*

Proof. By induction on $\mathtt{L}$, using the fact that if $\mathtt{CL}$ is a ϑ-evaluation context then $\mathtt{C}$ is a $\mathsf{fz}^\vartheta(\mathtt{L})$-evaluation context (Lem. A.20). □

Lemma A.63 (Stripping substitutions from a $\mathtt{lsv}$ redex using chain contexts)**.** *Let $\mathbb{X}^\vartheta$ denote either the set E_ϑ or the set $\mathsf{E}^\circ_\vartheta$. If $F_1^\vartheta\langle F_2^{\vartheta'}\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}']\rangle = t\mathtt{L}$ where $F_1^\vartheta\langle F_2^{\vartheta'}[x\backslash\mathtt{vL}']\rangle \in \mathbb{X}^\vartheta$ is an evaluation context then at least one of the following four possibilities holds:*

1. *A $F_1^\vartheta = F_{11}^{\vartheta''}\mathtt{L}$ where $\vartheta'' = \mathsf{fz}^\vartheta(\mathtt{L})$ and $F_{11}^{\vartheta''} \in \mathbb{X}^{\vartheta''}$.*
2. *B $F_1^\vartheta = F_{11}^{\vartheta''}\langle\!\langle y\rangle\!\rangle\mathscr{L}\{\Box\}$ such that $\mathtt{L} = \Box\mathscr{L}\{F_2^{\vartheta'}\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}']\}$, where $\vartheta'' = \mathsf{fz}^\vartheta(\mathtt{L})$, the evaluation context $F_{11}^{\vartheta''}$ is in $\mathbb{X}^{\vartheta''}$ and $\mathscr{L}$ is a (ϑ, y)-chain context.*
3. *C $F_2^{\vartheta'} = F_{21}^{\vartheta''}\tilde{\mathtt{L}}$ such that $\mathtt{L} = F_1^\vartheta\langle\tilde{\mathtt{L}}[x\backslash\mathtt{vL}']\rangle$, where $\vartheta'' = \mathsf{fz}^\vartheta(\tilde{\mathtt{L}})$, the context F_1^ϑ is a substitution context, and the evaluation context $F_{21}^{\vartheta''}$ is in $\mathsf{E}_{\vartheta''}$.*
4. *D $F_2^{\vartheta'} = F_{21}^{\vartheta''}\langle\!\langle y\rangle\!\rangle\{\Box\}$ such that $\mathtt{L} = F_1^\vartheta\langle\Box\mathscr{L}\{x\}[x\backslash\mathtt{vL}']\rangle$, where $\vartheta'' = \mathsf{fz}^\vartheta(\mathtt{L})$, the context F_1^ϑ is a substitution context, the evaluation context $F_{21}^{\vartheta''}$ is in $\mathsf{E}_{\vartheta''}$, and $\mathscr{L}$ is a (ϑ'', y)-chain context.*

Proof. We know that $F_1^\vartheta\langle F_2^{\vartheta'}\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}']\rangle = t\mathtt{L}$. We consider two cases, depending on whether $\mathtt{L}$ is "contained" in F_1^ϑ, that is, all the substitution nodes in the spine of $\mathtt{L}$ belong to the context F_1^ϑ, or otherwise:

1. If all the substitution nodes in the spine of $\mathtt{L}$ belong to the context F_1^ϑ That is, the substitution nodes in $\mathtt{L}$ do not come from the subterm $F_2^{\vartheta'}\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}']$. Then we may strip the substitution $\mathtt{L}$ from F_1^ϑ using Lem. A.62, which means that we are either in case **A** or case **B**, and we are done.
2. Otherwise Then some of the substitution nodes in $\mathtt{L}$ come from the subterm $F_2^{\vartheta'}\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}']$. So we have that F_1^ϑ is a substitution context and that $\mathtt{L} = F_1^\vartheta\langle\mathtt{L}_1\rangle$ for some substitution context $\mathtt{L}_1$. Note that $\mathtt{L}_1$ is non-empty since otherwise $\mathtt{L}$ would be subsumed in F_1^ϑ,

which has already been considered in the previous case. Since $\mathtt{L}_1$ is non-empty we have that $\mathtt{L}_1 = \tilde{\mathtt{L}}[x\backslash \mathtt{vL}']$. So $F_2^{\vartheta'}\langle\langle x\rangle\rangle[x\backslash \mathtt{vL}'] = t\tilde{\mathtt{L}}[x\backslash \mathtt{vL}']$. Then we may strip the substitution $\tilde{\mathtt{L}}$ from $F_2^{\vartheta'}[x\backslash \mathtt{vL}']$ using Lem. A.62. This gives us two possibilities, which correspond to cases **C** and **D** respectively.

□

Non-garbage contexts

This subsection deals with non-garbage contexts, which are used in the next subsection to prove backward stability for normal forms.

Definition A.64 (Non-garbage contexts)**.** The set of non-garbage contexts is given by the following grammar:

$$R ::= \Box \mid R\,t \mid t\,R \mid \lambda x.R \mid R[x\backslash t] \mid R\langle\langle x\rangle\rangle[x\backslash R]$$

Lemma A.65 (Non-garbage variables are variables below non-garbage contexts)**.** *The following equivalence holds:*

$$\mathsf{ngv}(t) = \{x \mid \exists R.\ R \text{ is a non-garbage context and } t = R\langle\langle x\rangle\rangle\}$$

Proof. By induction on t. □

Lemma A.66 (Generalized evaluation contexts are non-garbage)**.** *Let $F^{\vartheta} \in \mathsf{E}_{\vartheta}$ be a generalized ϑ-evaluation context. Then F^{ϑ} is non-garbage.*

Proof. By induction on the derivation that $F^{\vartheta} \in \mathsf{E}_{\vartheta}$. □

Lemma A.67 (Replacing a variable in a non-garbage context yields a non-garbage context)**.** *Let $\widehat{\mathsf{C}}$ be a two-hole context such that $\widehat{\mathsf{C}}\langle\Box, y\rangle$ is a non-garbage context and y is not bound by the context $\widehat{\mathsf{C}}\langle q, \Box\rangle$ (for an arbitrary term q). Then for any term s the context $\widehat{\mathsf{C}}\langle\Box, s\rangle$ is non-garbage.*

Proof. By induction on the derivation that $\widehat{\mathsf{C}}\langle\Box, y\rangle$ is a non-garbage context. □

Lemma A.68 (Preservation of non-garbage variables by internal steps when going to normal form)**.** *Let $t \xrightarrow{\neg\vartheta}_{\mathtt{sh}} s$ be a ϑ-internal step, such that $s \in \mathtt{nf}\vartheta$ is a strong ϑ-normal form. Then* $\mathsf{ngv}(t) \subseteq \mathsf{ngv}(s)$.

Proof. Let $\mathtt{r} : t \xrightarrow{\neg\vartheta}_{\mathtt{sh}} s$ be the internal step. The proof goes by induction on t. If t is a variable or an abstraction it is immediate. We consider the cases for application and substitution:

1. Application, $t = t_1\,t_2$ Note that $\mathtt{r}$ cannot be a step at the root, since it would be a $\mathtt{db}$ step, and it would be external. Hence there are two cases, depending on whether the step $\mathtt{r}$ is internal to t_1 or internal to t_2:

   1.1 If $\mathtt{r}$ is internal to t_1 Let $\mathtt{r}_1 : t_1 \to_{\mathtt{sh}\backslash\mathtt{gc}} s_1$ be the step isomorphic to $\mathtt{r}$ but going under the context $\Box\,t_2$. Then $s = s_1\,t_2$. Note that $\mathtt{r}_1$ cannot be ϑ-external, for otherwise $\mathtt{r}$ would be ϑ-external. So $\mathsf{ngv}(t_1\,t_2) = \mathsf{ngv}(t_1) \cup \mathsf{ngv}(t_2) \subseteq^{\textit{i.h.}} \mathsf{ngv}(s_1) \cup \mathsf{ngv}(t_2) = \mathsf{ngv}(s_1\,t_2)$.

1.2 If $\mathtt{r}$ is internal to t_2 Let $\mathtt{r}_1 : t_2 \to_{\mathtt{sh}\setminus\mathtt{gc}} s_2$ be the step isomorphic to $\mathtt{r}$ but going under the context $t_1\,\Box$. Then $s = t_1\,s_2$. Recall that by hypothesis $s \in \mathtt{nf}\vartheta$ is a normal form, so t_1 must be a strong ϑ-structure, *i.e.* $t_1 \in \mathsf{S}_\vartheta$. The step $\mathtt{r}_1$ cannot be ϑ-external, for otherwise $\mathtt{r}$ would be ϑ-external (note that this depends on the fact that t_1 is a structure). So $\mathsf{ngv}(t_1\,t_2) = \mathsf{ngv}(t_1) \cup \mathsf{ngv}(t_2) \subseteq^{\textit{i.h.}} \mathsf{ngv}(t_1) \cup \mathsf{ngv}(s_2) = \mathsf{ngv}(t_1\,s_2)$.

2. Substitution, $t = t_1[x\backslash t_2]$ We consider three cases, depending on whether (1) the step $\mathtt{r}$ is at the root of t, (2) $\mathtt{r}$ is internal to t_1, (3) $\mathtt{r}$ is internal to t_2.

2.1 If $\mathtt{r}$ is at the root of t Then $\mathtt{r}$ is a $\mathtt{lsv}$ step, which means that $t_1 = \mathtt{C}\langle\!\langle x\rangle\!\rangle$ and $t_2 = \mathtt{vL}$ in such a way that $\mathtt{r} : t = \mathtt{C}\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle\mathtt{v}\rangle[x\backslash\mathtt{v}]\mathtt{L} = s$. Since $s = \mathtt{C}\langle\mathtt{v}\rangle[x\backslash\mathtt{v}]\mathtt{L} \in \mathtt{nf}\vartheta$ we may strip the substitution context $[x\backslash\mathtt{v}]\mathtt{L}$ (by Lem. A.27) to obtain that $\mathtt{C}\langle\mathtt{v}\rangle \in \mathtt{nf}\hat{\vartheta}$ where $\hat{\vartheta} \subseteq \mathsf{fz}^\vartheta([x\backslash\mathtt{v}]\mathtt{L}) = \mathsf{fz}^\vartheta(\mathtt{L})$. We consider two cases, depending on whether $\mathtt{C}\langle\!\langle x\rangle\!\rangle$ is a normal form in $\mathtt{nf}\hat{\vartheta}$:

2.1.1 If $\mathtt{C}\langle\!\langle x\rangle\!\rangle \in \mathtt{nf}\hat{\vartheta}$ We consider two further subcases, depending on whether x is a non-garbage variable in $\mathtt{C}\langle\!\langle x\rangle\!\rangle$:

2.1.1.1 If $x \in \mathsf{ngv}(\mathtt{C}\langle\!\langle x\rangle\!\rangle)$ Recall that $\hat{\vartheta} \subseteq \vartheta \cup \mathsf{domL}$. Moreover, observe that $\mathtt{C}\langle\!\langle x\rangle\!\rangle$ is outside the scope of $\mathtt{L}$ in the original term $\mathtt{C}\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}]$, so by Barendregt's convention we may suppose that variables in domL do not occur in $\mathtt{C}\langle\!\langle x\rangle\!\rangle$. In particular, variables in domL are garbage in $\mathtt{C}\langle\!\langle x\rangle\!\rangle$, so since garbage variables are not required in "ϑ" (Lem. A.25), $\mathtt{C}\langle\!\langle x\rangle\!\rangle \in \mathtt{nf}\vartheta$. Since $x \in \mathsf{ngv}(\mathtt{C}\langle\!\langle x\rangle\!\rangle)$ and $\mathtt{C}\langle\!\langle x\rangle\!\rangle$ is a normal form in $\mathtt{nf}\vartheta$, by the fact that non-garbage variables in normal forms are below evaluation contexts (Lem. A.25), we have that there exists an evaluation context $F^\vartheta \in \mathsf{E}_\vartheta$ such that $\mathtt{C}\langle\!\langle x\rangle\!\rangle = F^\vartheta\langle\!\langle x\rangle\!\rangle$. There are two subcases, depending on whether $\mathtt{C} = F^\vartheta$ or $\mathtt{C} \neq F^\vartheta$:

- If $\mathtt{C} = F^\vartheta$ Then $\mathtt{C}[x\backslash\mathtt{vL}]$ is an evaluation context in E_ϑ, contradicting the fact that $\mathtt{r}$ is ϑ-internal.
- If $\mathtt{C} \neq F^\vartheta$ Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box, x\rangle = F^\vartheta$ and $\widehat{\mathtt{C}}\langle x, \Box\rangle = \mathtt{C}$, and the step is of the form: $\mathtt{r} : t = \widehat{\mathtt{C}}\langle\underline{x}, x\rangle[x\backslash\mathtt{vL}] \to_{\mathtt{sh}\setminus\mathtt{gc}} \widehat{\mathtt{C}}\langle\underline{x}, \mathtt{v}\rangle[x\backslash\mathtt{v}]\mathtt{L} = s$. Note that the underlined occurrence of x is non-garbage on the left-hand side, so it is also non-garbage on the right-hand side.
  More precisely, $\widehat{\mathtt{C}}\langle\Box, x\rangle = F^\vartheta$ is an evaluation context so by Lem. A.66 it is also a non-garbage context. Recall that replacing a variable by an arbitrary term in a non-garbage context is still a non-garbage context (Lem. A.67), so $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle$ is also non-garbage. Moreover, since non-garbage variables coincide with variables below non-garbage contexts (Lem. A.65) we have that $x \in \mathsf{ngv}(\widehat{\mathtt{C}}\langle x, \mathtt{v}\rangle)$.
  This contradicts the fact that s is a normal form, since to conclude that $\widehat{\mathtt{C}}\langle x, \mathtt{v}\rangle[x\backslash\mathtt{vL}]$ is a normal form, given that $x \in \mathsf{ngv}(\widehat{\mathtt{C}}\langle x, \mathtt{v}\rangle)$, we would require that x is bound to a structure, but it is bound to a value $\mathtt{v}$.

2.1.1.2 If $x \notin \mathsf{ngv}(\mathtt{C}\langle\!\langle x \rangle\!\rangle)$ Let us show that $\mathsf{ngv}(t) \subseteq \mathsf{ngv}(s)$. Consider an arbitrary variable $y \in \mathsf{ngv}(t) = \mathsf{ngv}(\mathtt{C}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}])$, and let us show that $y \in \mathsf{ngv}(s)$. Since x is garbage in $\mathtt{C}\langle\!\langle x \rangle\!\rangle$, it must be the case that $y \in \mathsf{ngv}(\mathtt{C}\langle\!\langle x \rangle\!\rangle)$. Moreover, since $x \neq y$ and y is non-garbage in $\mathtt{C}\langle\!\langle x \rangle\!\rangle$, by the fact that non-garbage variables are below non-garbage contexts (Lem. A.65) there must exist a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle \Box, x\rangle$ is non-garbage and $\widehat{\mathtt{C}}\langle y, \Box\rangle = \mathtt{C}$. By replacing a variable in a non-garbage context (Lem. A.67) we obtain that $\widehat{\mathtt{C}}\langle \Box, \mathtt{v}\rangle = \mathtt{C}$ is also non-garbage. So $y \in \mathsf{ngv}(\widehat{\mathtt{C}}\langle y, \mathtt{v}\rangle) = \mathsf{ngv}(\mathtt{C}\langle \mathtt{v}\rangle)$. Hence $y \in \mathsf{ngv}(\mathtt{C}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\mathtt{L}) = \mathsf{ngv}(s)$, as required.

2.1.2 If $\mathtt{C}\langle\!\langle x \rangle\!\rangle \notin \mathtt{nf}\hat{\vartheta}$ Then by definition (Def. A.56) $\mathtt{C}$ is a $\mathtt{nf}\hat{\vartheta}$-critical context. By Lem. A.57 since $\mathtt{C}$ is $\mathbb{X}^{\hat{\vartheta}}$-critical, it is an evaluation context, $\mathtt{C} \in \mathsf{E}_{\hat{\vartheta}}$. By strengthening ϑ (Tactic A.55) $\mathtt{C}[x\backslash \mathtt{vL}] \in \mathsf{E}_{\vartheta}$, contradicting the fact that the step $\mathtt{r}$ is ϑ-internal.

2.2 If $\mathtt{r}$ is internal to t_1 Let $\mathtt{r}_1 : t_1 \to s_1$ be the step isomorphic to $\mathtt{r}$ but going under the context $[x\backslash t_2]$. Then $s = s_1[x\backslash t_2]$. Note that $\mathtt{r}_1$ cannot be ϑ-external, since then $\mathtt{r}$ would be ϑ-external. There are two cases, depending on whether x is garbage in t_1 or not:

2.2.1 If $x \in \mathsf{ngv}(t_1)$ Note that by *i.h.* $x \in \mathsf{ngv}(s_1)$. Then $\mathsf{ngv}(t) = \mathsf{ngv}(t_1) \cup \mathsf{ngv}(t_2) \subseteq^{\textit{i.h.}} \mathsf{ngv}(s_1) \cup \mathsf{ngv}(t_2) = \mathsf{ngv}(s_1[x\backslash t_2]) = \mathsf{ngv}(s)$.

2.2.2 If $x \notin \mathsf{ngv}(t_1)$ Then $\mathsf{ngv}(t) = \mathsf{ngv}(t_1) \subseteq^{\textit{i.h.}} \mathsf{ngv}(s_1) \subseteq \mathsf{ngv}(s_1[x\backslash t_2]) = \mathsf{ngv}(s)$.

2.3 If $\mathtt{r}$ is internal to t_2 Let $\mathtt{r}_1 : t_2 \to s_2$ be the step isomorphic to $\mathtt{r}$ but going under the context $t_1[x\backslash \Box]$. Then $s = t_1[x\backslash s_2]$. We consider two subcases, depending on whether x is garbage in t_1 or not:

2.3.1 If $x \in \mathsf{ngv}(t_1)$ We consider two subcases, depending on whether $\mathtt{r}_1$ is ϑ-external or ϑ-internal:

2.3.1.1 If $\mathtt{r}$ is ϑ-external Since $t_1[x\backslash s_2]$ is a normal form, we have that $t_1 \in \mathtt{nf}\vartheta \cup \{x\}$. By the fact that non-garbage variables in normal forms are below evaluation contexts (Lem. A.25) there must exist an evaluation context $F_1^{\vartheta} \in \mathsf{E}_{\vartheta}$ such that $t_1 = F_1^{\vartheta}\langle\!\langle x \rangle\!\rangle$. Moreover, since the step $\mathtt{r}_1$ is external, we have that $t_2 = F_2^{\vartheta}\langle\!\langle \Sigma \rangle\!\rangle$ where $F_2^{\vartheta} \in \mathsf{E}_{\vartheta}$ and Σ is the anchor of a redex. If we let Σ' denote its contractum, we have that the step $\mathtt{r}$ is of the form $\mathtt{r} : t = F_1^{\vartheta}\langle\!\langle x \rangle\!\rangle[x\backslash F_2^{\vartheta}\langle \Sigma\rangle] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_1^{\vartheta}\langle\!\langle x \rangle\!\rangle[x\backslash F_2^{\vartheta}\langle \Sigma'\rangle] = s$. Note that F_2^{ϑ} cannot be a inert ϑ-evaluation context, since otherwise the step $\mathtt{r}$ would be ϑ-external.
Hence we have that $F_2^{\vartheta} \notin \mathsf{E}_{\vartheta}^{\circ}$. Recall that evaluation contexts which are not inert evaluation contexts have the shape of an answer (Lem. A.23). In particular, the subterm $F_2^{\vartheta}\langle \Sigma'\rangle$ is an answer $(\lambda y.r)\mathtt{L}$. This contradicts the hypothesis that $s = F_1^{\vartheta}\langle\!\langle x \rangle\!\rangle[x\backslash(\lambda y.r)\mathtt{L}]$ is in normal form, since x is below an evaluation context and bound to an answer.

2.3.1.2 If $\mathtt{r}$ is ϑ-internal Then $\mathsf{ngv}(t) = \mathsf{ngv}(t_1) \cup \mathsf{ngv}(t_2) \subseteq^{\textit{i.h.}} \mathsf{ngv}(s_1) \cup \mathsf{ngv}(t_2) = \mathsf{ngv}(s)$ as required.

2.3.2 If $x \notin \mathsf{ngv}(t_1)$ Then $\mathsf{ngv}(t) = \mathsf{ngv}(t_1) = \mathsf{ngv}(s)$ and we are done.

□

Backward stability of normal forms

To prove that internal steps can be postponed, we need to deal with situations such as $ts \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t's \overset{\vartheta}{\rightsquigarrow} t's'$. Here $t'\Box$ must be an evaluation context, since the second step is external, so t' is a structure. We would like to obtain that $t\Box$ is also an evaluation context, *i.e.* we would like to show that t is a structure. This is where the following lemma comes into play.

Lemma A.69 (Backward stability of normal forms)**.** *Let $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t$ be an internal step with $t \in \mathbb{X}^\vartheta$ where $\mathbb{X}^\vartheta$ stands for either $\mathtt{nf}\vartheta$ or S_ϑ. Then $t_0 \in \mathbb{X}^\vartheta$.*

Proof. By induction on the derivation that $t \in \mathbb{X}^\vartheta$. The interesting cases are N-APP, NFSUBG, and NFSUB.

1. N-APP, $t = M^\vartheta\, N^\vartheta \in \mathsf{S}_\vartheta$ with $M^\vartheta \in \mathsf{S}_\vartheta$ and $N^\vartheta \in \mathtt{nf}\vartheta$ Note that the step $\mathtt{r}$ cannot be at the root of t_0, since the right-hand side of both $\mathtt{db}$ and $\mathtt{lsv}$ steps is a substitution, rather than an application.

   So t_0 is an application $t_1\, t_2$, and we consider two cases depending on whether the step $\mathtt{r}$ is internal to t_1 or internal to t_2:

   1.1 If $\mathtt{r}$ is to the left of $t_0 = t_1\, N^\vartheta$ Let $\mathtt{r}_1 : t_1 \to_{\mathtt{sh}\setminus\mathtt{gc}} M^\vartheta$ be the step isomorphic to $\mathtt{r}$ but going under the context $\Box\, N^\vartheta$. Note that $\mathtt{r}_1$ cannot be ϑ-external, since this would imply that $\mathtt{r}$ is ϑ-external. So $\mathtt{r}_1$ is ϑ-internal and by *i.h.* we have that $t_1 \in \mathsf{S}_\vartheta$. Hence $t_0 = t_1\, N^\vartheta \in \mathsf{S}_\vartheta$, as required.

   1.2 If $\mathtt{r}$ is to the right of $t_0 = M^\vartheta\, t_2$ Let $\mathtt{r}_1 : t_2 \to_{\mathtt{sh}\setminus\mathtt{gc}} N^\vartheta$ be the step isomorphic to $\mathtt{r}$ but going under the context $M^\vartheta\, \Box$. Note that $\mathtt{r}_1$ cannot be ϑ-external, since this would imply that $\mathtt{r}$ is ϑ-external. So $\mathtt{r}_1$ is ϑ-internal and by *i.h.* we have that $t_2 \in \mathtt{nf}\vartheta$. Hence $t_0 = M^\vartheta\, t_2 \in \mathsf{S}_\vartheta$, as required.

2. NFSUBG, $t = s[x\backslash u] \in \mathbb{X}^\vartheta$ with $x \notin \mathsf{ngv}(s)$ and $s \in \mathbb{X}^\vartheta$ We consider three cases, depending on whether (1) $\mathtt{r}$ is a step at the root of t_0, (2) t_0 is a substitution $s_0[x\backslash u_0]$ and $\mathtt{r}$ is internal to t_1, (3) t_0 is a substitution $s_0[x\backslash u_0]$ and $\mathtt{r}$ is internal to t_2.

   2.1 If $\mathtt{r}$ is at the root Note that $\mathtt{r}$ cannot be a $\mathtt{db}$ step since it would be external, it must be a $\mathtt{lsv}$ step $\mathtt{r} : t_0 = \mathtt{C}\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{v}\mathtt{L}] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle \mathtt{v}\rangle[y\backslash \mathtt{v}]\mathtt{L} = s[x\backslash u]$. So s is of the form $s = s'\mathtt{L}_1$ with $\mathtt{L}_1[x\backslash u] = [y\backslash \mathtt{v}]\mathtt{L}$ and $\mathtt{C}\langle \mathtt{v}\rangle = s'$. Note that since $s = s'\mathtt{L}_1 \in \mathbb{X}^\vartheta$ by Lem. A.27 we must have $s' \in \mathbb{X}^{\hat{\vartheta}}$ where $\hat{\vartheta} \subseteq \mathsf{fz}^\vartheta(\mathtt{L}_1[x\backslash u]) = \mathsf{fz}^\vartheta([y\backslash \mathtt{v}]\mathtt{L}) = \mathsf{fz}^\vartheta(\mathtt{L})$. We consider two subcases, depending on whether $\mathtt{C}\langle\!\langle y \rangle\!\rangle \in \mathbb{X}^{\hat{\vartheta}}$.

      2.1.1 If $\mathtt{C}\langle\!\langle y \rangle\!\rangle \in \mathbb{X}^{\hat{\vartheta}}$ By strengthening ϑ (Tactic A.55), $\mathtt{C}\langle\!\langle y \rangle\!\rangle \in \mathbb{X}^\vartheta$. Consider two further subcases, depending on whether y is a garbage variable in $\mathtt{C}\langle\!\langle y \rangle\!\rangle$:

2.1.1.1 If $y \in \mathsf{ngv}(\mathtt{C}\langle\!\langle y \rangle\!\rangle)$ Recall that non-garbage variables in a normal form are below evaluation contexts (Lem. A.25). Then since $\mathtt{C}\langle\!\langle y \rangle\!\rangle$ is a normal form in $\mathbb{X}^{\vartheta}$ and $y \in \mathsf{ngv}(\mathtt{C}\langle\!\langle y \rangle\!\rangle)$, we have that $\mathtt{C}\langle\!\langle y \rangle\!\rangle$ may be written as $F^{\vartheta}\langle\!\langle y \rangle\!\rangle$, where $F^{\vartheta} \in \mathsf{E}_{\vartheta}$.
If $\mathtt{C}$ and F^{ϑ} are the same context, then $\mathtt{C}[y\backslash \mathtt{vL}] \in \mathsf{E}_{\vartheta}$, which contradicts the hypothesis that the step $\mathtt{r}$ is ϑ-internal. So we may suppose that $\mathtt{C} \neq F^{\vartheta}$. Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle \Box, y \rangle = F^{\vartheta}$ and $\widehat{\mathtt{C}}\langle y, \Box \rangle = \mathtt{C}$, and the step $\mathtt{r}$ is of the form: $\mathtt{r} : \widehat{\mathtt{C}}\langle \underline{y}, y \rangle[y\backslash \mathtt{vL}] \to_{\mathtt{sh}\backslash\mathtt{gc}} \widehat{\mathtt{C}}\langle \underline{y}, \mathtt{v} \rangle[y\backslash \mathtt{v}]\mathtt{L} = t$. Note that the underlined occurrence of y is non-garbage on the left-hand side, so it is also non-garbage on the right-hand side.
More precisely, $\widehat{\mathtt{C}}\langle \Box, y \rangle = F^{\vartheta}$ is an evaluation context so by Lem. A.66 it is also a non-garbage context. Recall that replacing a variable by an arbitrary term in a non-garbage context is still a non-garbage context (Lem. A.67), so $\widehat{\mathtt{C}}\langle \Box, \mathtt{v} \rangle$ is also non-garbage. Moreover, since non-garbage variables coincide with variables below non-garbage contexts (Lem. A.65) we have that $y \in \mathsf{ngv}(\widehat{\mathtt{C}}\langle y, \mathtt{v} \rangle)$.
This contradicts the fact that t is a normal form, since to conclude that $\widehat{\mathtt{C}}\langle y, \mathtt{v} \rangle[y\backslash \mathtt{v}]$ is a normal form, given that $y \in \mathsf{ngv}(\widehat{\mathtt{C}}\langle y, \mathtt{v} \rangle)$ we would require that y is bound to a structure, but it is bound to a value $\mathtt{v}$.

2.1.1.2 If $y \notin \mathsf{ngv}(\mathtt{C}\langle\!\langle y \rangle\!\rangle)$ Then we are done, as $\mathtt{C}\langle\!\langle y \rangle\!\rangle \in \mathbb{X}^{\vartheta}$, so by applying the nfSubG rule we obtain that $\mathtt{C}\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{vL}] \in \mathbb{X}^{\vartheta}$, as wanted.

2.1.2 If $\mathtt{C}\langle\!\langle y \rangle\!\rangle \notin \mathbb{X}^{\hat{\vartheta}}$ Note that $\mathtt{C}\langle \mathtt{v} \rangle = s' \in \mathbb{X}^{\hat{\vartheta}}$. So by definition (Def. A.56) $\mathtt{C}$ is a $\mathbb{X}^{\hat{\vartheta}}$-critical context. By Lem. A.57 since $\mathtt{C}$ is $\mathbb{X}^{\hat{\vartheta}}$-critical, it is an evaluation context, $\mathtt{C} \in \mathbb{X}^{\hat{\vartheta}}$. By strengthening ϑ (Tactic A.55), $\mathtt{C}[y\backslash \mathtt{vL}] \in \mathbb{X}^{\vartheta}$, contradicting the fact that the step $\mathtt{r}$ is ϑ-internal.

2.2 If $\mathtt{r}$ is to the left of $t_0 = s_0[x\backslash u]$ Let $\mathtt{r}_1 : s_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} s$ be the step isomorphic to $\mathtt{r}$ but going under the context $\Box[x\backslash u]$. Note that $\mathtt{r}_1$ cannot be ϑ-external, since then $\mathtt{r}$ would be ϑ-external. So $\mathtt{r}_1$ is ϑ-internal and by *i.h.* we have that $s_0 \in \mathbb{X}^{\vartheta}$. Moreover, since non-garbage variables are preserved by internal steps (Lem. A.68), by the contrapositive we have that $x \notin \mathsf{ngv}(s_0)$, hence $t_0 = s_0[x\backslash u] \in \mathbb{X}^{\vartheta}$ as required.

2.3 If $\mathtt{r}$ is to the right of $t_0 = s[x\backslash u_0]$ Then by applying the rule nfSubG it is immediate that $t_0 = s[x\backslash u_0] \in \mathbb{X}^{\vartheta}$

3. nfSub, $t = s[x\backslash M^{\vartheta}] \in \mathbb{X}^{\vartheta}$ with $x \in \mathsf{ngv}(s)$, $s \in \mathbb{X}^{\vartheta \cup \{x\}}$ and $M^{\vartheta} \in \mathsf{S}_{\vartheta}$ We consider three cases, depending on whether (1) $\mathtt{r}$ is a step at the root of t_0, (2) t_0 is a substitution $s_0[x\backslash u_0]$ and $\mathtt{r}$ is internal to t_1, (3) t_0 is a substitution $s_0[x\backslash u_0]$ and $\mathtt{r}$ is internal to t_2.

3.1 If $\mathtt{r}$ is at the root Note that $\mathtt{r}$ cannot be a $\mathtt{db}$ step since it would be external, it must be a $\mathtt{lsv}$ step: $\mathtt{r} : t_0 = \mathtt{C}\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{vL}] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle \mathtt{v} \rangle[y\backslash \mathtt{v}]\mathtt{L} = s[x\backslash M^{\vartheta}]$. So let us write s as of the form $s = s'\mathtt{L}_1$ in such a way that $\mathtt{L}_1[x\backslash M^{\vartheta}] = [y\backslash \mathtt{v}]\mathtt{L}$. By Lem. A.27 we have that $s' \in \mathbb{X}^{\hat{\vartheta}}$ where $\hat{\vartheta} \subseteq \mathsf{fz}^{\vartheta}(\mathtt{L}_1[x\backslash M^{\vartheta}]) = \mathsf{fz}^{\vartheta}([y\backslash \mathtt{v}]\mathtt{L}) = \mathsf{fz}^{\vartheta}(\mathtt{L})$. Then the remainder of this case is analogous to case 2.1 of this lemma.

3.2 If $\mathtt{r}$ is to the left of $t_0 = s_0[x\backslash M^\vartheta]$ Let $\mathtt{r}_1 : s_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} s$ be the step isomorphic to $\mathtt{r}$ but going under the context $\Box[x\backslash M^\vartheta]$. Note that $\mathtt{r}_1$ cannot be $(\vartheta \cup \{x\})$-external, since then $\mathtt{r}$ would be ϑ-external. So $\mathtt{r}_1$ is $(\vartheta \cup \{x\})$-internal, and since $s \in \mathbb{X}^{\vartheta\cup\{x\}}$ by *i.h.* we have that $s_0 \in \mathbb{X}^{\vartheta\cup\{x\}}$. We consider two further subcases, depending on whether x is garbage in s_0:

3.2.1 If $x \in \mathsf{ngv}(s_0)$ Then $s_0[x\backslash M^\vartheta] \in \mathbb{X}^\vartheta$ since $s_0 \in \mathbb{X}^{\vartheta\cup\{x\}}$, by the rule nfSub.

3.2.2 If $x \notin \mathsf{ngv}(s_0)$ Then since garbage variables are not required in "ϑ" (Lem. A.25), we have that $s_0 \in \mathbb{X}^\vartheta$. Hence $s_0[x\backslash M^\vartheta] \in \mathbb{X}^\vartheta$, by the rule nfSubG.

3.3 If $\mathtt{r}$ is to the right of $t_0 = s[x\backslash u_0]$ Let $\mathtt{r}_1 : u_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} M^\vartheta$ be the step isomorphic to $\mathtt{r}$ but going under the context $s[x\backslash\Box]$. We consider two cases, depending on whether $\mathtt{r}_1$ is ϑ-external or ϑ-internal:

3.3.1 If $\mathtt{r}_1$ is ϑ-external First note that, since $x \in \mathsf{ngv}(s)$ and $s \in \mathbb{X}^{\vartheta\cup\{x\}}$, by the fact that non-garbage variables in normal forms are below evaluation contexts (Lem. A.25) there must exist an evaluation context $F_1^\vartheta \in \mathsf{E}_\vartheta$ such that $s = F_1^\vartheta\langle\!\langle x \rangle\!\rangle$.
Moreover, since $\mathtt{r}_1$ is a ϑ-external step, the term u_0 can be written as $F_2^\vartheta\langle\Sigma\rangle$, where F_2^ϑ is an evaluation context in E_ϑ and Σ is the anchor of a redex. If we let Σ' denote the contractum of Σ, the internal step is $\mathtt{r} : F_1^\vartheta\langle\!\langle x \rangle\!\rangle[x\backslash F_2^\vartheta\langle\Sigma\rangle] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_1^\vartheta\langle\!\langle x \rangle\!\rangle[x\backslash F_2^\vartheta\langle\Sigma'\rangle] = s[x\backslash M^\vartheta] = t$. Since the step $\mathtt{r}$ is ϑ-internal, the context F_2^ϑ cannot be a inert evaluation context, *i.e.* $F_2^\vartheta \notin \mathsf{E}_\vartheta^\circ$. Recall that evaluation contexts which are not inert evaluation contexts have the shape of an answer (Lem. A.23). This means that $F_2^\vartheta\langle\Sigma'\rangle = (\lambda y.r)\mathtt{L}$ is an answer. But we also had that $F_2^\vartheta\langle\Sigma'\rangle = M^\vartheta$, so it is both an answer and a structure, which is impossible.

3.3.2 If $\mathtt{r}_1$ is ϑ-internal Then by *i.h.* u_0 is a structure, *i.e.* $u_0 \in M^\vartheta$. Hence $s[x\backslash u_0] \in \mathbb{X}^\vartheta$, as required.

□

Backward preservation of evaluation contexts

To prove that internal steps can be postponed, we need to deal with situations such as $t[x\backslash I] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F^\vartheta\langle\!\langle x \rangle\!\rangle[x\backslash I] \overset{\vartheta}{\rightsquigarrow} F^\vartheta\langle I\rangle[x\backslash I]$. We would like to obtain that t can also be written as $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ for some evaluation context $\mathtt{C}$, to swap the external step before the internal one. This is precisely the situation addressed by the following lemma.

Lemma A.70 (Backward stability of evaluation contexts)**.** *Let $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F^\vartheta\langle\!\langle x \rangle\!\rangle$ be an internal step with $F^\vartheta\langle\!\langle x \rangle\!\rangle \in \mathbb{X}^\vartheta$, where $\mathbb{X}^\vartheta$ stands for either E_ϑ or $\mathsf{E}_\vartheta^\circ$, and such that F^ϑ does not bind x. Then there exists an evaluation context $F_0^\vartheta \in \mathbb{X}^\vartheta$ such that $t_0 = F_0^\vartheta\langle\!\langle x \rangle\!\rangle$.*

Proof. Let $\mathtt{r}$ be the name of the ϑ-internal step $\mathtt{r} : t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F^\vartheta\langle\!\langle x \rangle\!\rangle$. The proof goes by induction on the derivation that $F^\vartheta \in \mathbb{X}^\vartheta$. The cases for rules EBox, EAppL, and ELam are easy. We deal with the other rules:

1. ESubLNonStr, $F^{\vartheta} = F_1^{\vartheta}[y\backslash t]$ with $F_1^{\vartheta} \in \mathbb{X}^{\vartheta}$ and $t \notin \mathtt{S}_{\vartheta}$ We consider three cases, depending on whether (1) the internal step $\mathtt{r}$ is at the root of t_0, (2) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to t_0', (3) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to r_0.

   1.1 The internal step $\mathtt{r}$ is at the root of t_0 Note that $\mathtt{r}$ cannot be a db step, since it would be external. So it is an lsv step of the form: $\mathtt{r} : t_0 = \mathtt{C}\langle\!\langle z \rangle\!\rangle[z\backslash \mathtt{vL}] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle \mathtt{v}\rangle[z\backslash \mathtt{v}]\mathtt{L} = F_1^{\vartheta}\langle\!\langle x \rangle\!\rangle[y\backslash t] = t_1$. Let $\mathtt{L}_1$ be the substitution context such that $\mathtt{L}_1[y\backslash t] = [z\backslash \mathtt{v}]\mathtt{L}$, and using Lem. A.62 let us strip the substitution $\mathtt{L}_1$ from $F_1^{\vartheta}\langle\!\langle x \rangle\!\rangle$. This gives us two possibilities, **A** and **B**:

      1.1.1 Case A Then $F_1^{\vartheta} = F_{11}^{\hat{\vartheta}}\mathtt{L}_1$ and $\mathtt{C}\langle \mathtt{v}\rangle = F_{11}^{\hat{\vartheta}}\langle\!\langle x \rangle\!\rangle$ where $\hat{\vartheta} = \mathsf{fz}^{\vartheta}(\mathtt{L}_1)$ and $F_{11}^{\hat{\vartheta}} \in \mathbb{X}^{\hat{\vartheta}}$. We consider three further subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\hat{\vartheta}}$.

         1.1.1.1 The hole of $\mathtt{C}$ and the hole of $F_{11}^{\hat{\vartheta}}$ are disjoint Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle \Box, \mathtt{v}\rangle = F_{11}^{\hat{\vartheta}}$ and $\widehat{\mathtt{C}}\langle x, \Box\rangle = \mathtt{C}$. By Lem. A.58 there are two possibilities: the left and the right branch of the disjunction. The right branch case is impossible since it contradicts that $\mathtt{r}$ is ϑ-internal (by strengthening ϑ, Tactic A.55). In the left branch case, $\widehat{\mathtt{C}}\langle \Box, z\rangle \in \mathbb{X}^{\hat{\vartheta}}$ so by strengthening ϑ (Tactic A.55), $\widehat{\mathtt{C}}\langle \Box, z\rangle[z\backslash \mathtt{vL}] \in \mathbb{X}^{\vartheta}$. Hence $t_0 = \widehat{\mathtt{C}}\langle x, z\rangle[z\backslash \mathtt{vL}]$ and by taking $F_0^{\vartheta} := \widehat{\mathtt{C}}\langle \Box, z\rangle[z\backslash \mathtt{vL}]$ we conclude.

         1.1.1.2 The context $\mathtt{C}$ is a prefix of the context $F_{11}^{\hat{\vartheta}}$ By strengthening ϑ (Tactic A.55), $\mathtt{C}[z\backslash \mathtt{vL}] \in \mathbb{X}^{\hat{\vartheta}}$. This contradicts the fact that $\mathtt{r}$ is ϑ-internal.

         1.1.1.3 The context $F_{11}^{\hat{\vartheta}}$ is a prefix of the context $\mathtt{C}$ Then $\mathtt{C} = F_{11}^{\hat{\vartheta}}\langle \mathtt{C}_1\rangle$, so $\mathtt{C}_1\langle \mathtt{v}\rangle = x$, which is impossible.

      1.1.2 Case B Then $F_1^{\vartheta} = F_{11}^{\hat{\vartheta}}\langle\!\langle w \rangle\!\rangle\mathscr{L}\{\Box\}$, $\mathtt{C}\langle \mathtt{v}\rangle = F_{11}^{\hat{\vartheta}}\langle\!\langle w \rangle\!\rangle$, and $\mathtt{L}_1 = \Box\mathscr{L}\{x\}$ where $\hat{\vartheta} = \mathsf{fz}^{\vartheta}(\mathtt{L}_1)$, the evaluation context $F_{11}^{\hat{\vartheta}}$ is in $\mathbb{X}^{\hat{\vartheta}}$, and $\mathscr{L}$ is a (ϑ, w)-chain context. We consider three further subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\hat{\vartheta}}$. They remainder of this case is similar to case 1.1.1, except when the hole of $\mathtt{C}$ and the hole of $F_{11}^{\hat{\vartheta}}$ are disjoint. Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle \Box, \mathtt{v}\rangle = F_{11}^{\hat{\vartheta}}$ and $\widehat{\mathtt{C}}\langle w, \Box\rangle = \mathtt{C}$, and the internal step $\mathtt{r}$ is of the form: $\mathtt{r} : t_0 = \widehat{\mathtt{C}}\langle w, z\rangle[z\backslash \mathtt{vL}] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \widehat{\mathtt{C}}\langle w, \mathtt{v}\rangle[z\backslash \mathtt{v}]\mathtt{L} = t_1$. Note that w is bound by $[z\backslash \mathtt{v}]\mathtt{L} = \Box\mathscr{L}\{x\}[y\backslash t]$ on the right-hand side of the step $\mathtt{r}$ since $\mathscr{L}$ is a (ϑ, w)-chain context. So w must be bound by $[z\backslash \mathtt{vL}]$ on the left-hand side of the step $\mathtt{r}$, for otherwise it would be free, and free variables cannot become bound by reduction.

         Hence it must be the case that $w = z$. Note that w is bound to a term of the form $\mathtt{I}_1^{\vartheta_1}\langle\!\langle w_1 \rangle\!\rangle$ on the right-hand side of the step $\mathtt{r}$, and we have just argued that $w = z$, so $\mathtt{I}_1^{\vartheta_1}\langle\!\langle w_1 \rangle\!\rangle = \mathtt{v}$. This is impossible since answers do not have variables below inert evaluation contexts (Lem. A.21).

   1.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'[y\backslash t]$ Let $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} F_1^{\vartheta}\langle\!\langle x \rangle\!\rangle$ be the step isomorphic to $\mathtt{r}$ but going under the context $[y\backslash t]$. Then by *i.h.* there is an

evaluation context $F_{10}^{\vartheta} \in \mathbb{X}^{\vartheta}$ such that $t_0' = F_{10}^{\vartheta}\langle\!\langle x \rangle\!\rangle$. By taking $F_0^{\vartheta} := F_{10}^{\vartheta}[y\backslash t] \in \mathbb{X}^{\vartheta}$ we conclude that $t_0 = F_{10}^{\vartheta}\langle\!\langle x \rangle\!\rangle[y\backslash t]$, as required.

1.3 The internal step $\mathtt{r}$ is to the right of $t_0 = F_1^{\vartheta}\langle\!\langle x \rangle\!\rangle[y\backslash t_0']$ By taking $F_0^{\vartheta} := F_{10}^{\vartheta}[y\backslash t_0'] \in \mathbb{X}^{\vartheta}$ we conclude that $t_0 = F_{10}^{\vartheta}\langle\!\langle x \rangle\!\rangle[y\backslash t]$, as required.

2. ESUBLSTR, $F^{\vartheta} = F_1^{\vartheta\cup\{y\}}[y\backslash M^{\vartheta}]$ with $F^{\vartheta\cup\{y\}} \in \mathbb{X}^{\vartheta\cup\{y\}}$ and $M^{\vartheta} \in \mathsf{S}_{\vartheta}$ We consider three cases, depending on whether (1) the internal step $\mathtt{r}$ is at the root of t_0, (2) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to t_0', (3) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to r_0.

2.1 The internal step $\mathtt{r}$ is at the root of t_0 Note that $\mathtt{r}$ cannot be a $\mathtt{db}$ step, since it would be external. So $\mathtt{r}$ is an $\mathtt{lsv}$ step of the form: $\mathtt{r} : t_0 = \mathtt{C}\langle\!\langle z \rangle\!\rangle[z\backslash \mathtt{vL}] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle\!\langle \mathtt{v} \rangle\!\rangle[z\backslash \mathtt{v}]\mathtt{L} = F_1^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle[y\backslash M^{\vartheta}] = t_1$. Let $\mathtt{L}_1$ be the substitution context such that $[z\backslash \mathtt{v}]\mathtt{L} = \mathtt{L}_1[y\backslash M^{\vartheta}]$, and using Lem. A.62 let us strip the substitution $\mathtt{L}_1$ from $F_1^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle$. This gives us two possibilities, **A** and **B**:

2.1.1 Case A Then $F_1^{\vartheta\cup\{y\}} = F_{11}^{\hat{\vartheta}\cup\{y\}}\mathtt{L}_1$ and $\mathtt{C}\langle \mathtt{v} \rangle = F^{\hat{\vartheta}\cup\{y\}}\langle\!\langle x \rangle\!\rangle$ where $\hat{\vartheta} \cup \{y\} = \mathsf{fz}^{\vartheta\cup\{y\}}(\mathtt{L}_1)$ and $F_{11}^{\hat{\vartheta}\cup\{y\}} \in \mathbb{X}^{\hat{\vartheta}\cup\{y\}}$. We consider three further subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\hat{\vartheta}\cup\{y\}}$:

2.1.1.1 The hole of $\mathtt{C}$ and $F_{11}^{\hat{\vartheta}\cup\{y\}}$ are disjoint Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle \Box, \mathtt{v} \rangle = F_{11}^{\hat{\vartheta}\cup\{y\}}$ and $\widehat{\mathtt{C}}\langle x, \Box \rangle = \mathtt{C}$. By Lem. A.58 there are two possibilities: the left and the right branch of the disjunction. The right branch case is impossible, ssible since it contradicts that $\mathtt{r}$ is ϑ-internal (by strengthening ϑ, Tactic A.55).
In the left branch case, by strengthening ϑ (Tactic A.55), $\widehat{\mathtt{C}}\langle \Box, z \rangle[z\backslash \mathtt{vL}] \in \mathbb{X}^{\vartheta}$. Then it is immediate to conclude, since by taking $F_0^{\vartheta} := \widehat{\mathtt{C}}\langle \Box, z \rangle[z\backslash \mathtt{vL}] \in \mathbb{X}^{\vartheta}$, we have that $t_0 = \widehat{\mathtt{C}}\langle x, z \rangle[z\backslash \mathtt{vL}]$, as required.

2.1.1.2 The context $\mathtt{C}$ is a prefix of $F_{11}^{\hat{\vartheta}\cup\{y\}}$ By strengthening ϑ (Tactic A.55), $\mathtt{C}[z\backslash \mathtt{vL}] \in \mathbb{X}^{\vartheta\cup\{y\}}$ Moreover, y is bound by $\mathtt{L}_1[y\backslash M^{\vartheta}] = [z\backslash \mathtt{v}]\mathtt{L}$, and $y \neq z$, since y is bound to M^{ϑ} and z is bound to $\mathtt{v}$. Hence y cannot occur free in the subterm $\mathtt{C}\langle\!\langle z \rangle\!\rangle$ on the left-hand side of the step $\mathtt{r}$. In particular, y does not occur as a structural variable in $\mathtt{C}$. So applying the fact that non-structural variables are not required in "ϑ" (Lem. A.54) we obtain that $\mathtt{C}[z\backslash \mathtt{vL}] \in \mathbb{X}^{\vartheta}$. This contradicts that the step $\mathtt{r}$ is ϑ-internal.

2.1.1.3 The context $F_{11}^{\hat{\vartheta}\cup\{y\}}$ is a prefix of $\mathtt{C}$ Then $\mathtt{C} = F_{11}^{\hat{\vartheta}\cup\{y\}}\langle \mathtt{C}_1 \rangle$, so $\mathtt{C}_1\langle \mathtt{v} \rangle = x$, which is impossible.

2.1.2 Case B Then $F_1^{\vartheta} = F_{11}^{\vartheta\cup\{y\}}\langle\!\langle w \rangle\!\rangle\mathscr{L}\{\Box\}$, $\mathtt{C}\langle \mathtt{v} \rangle = F_{11}^{\hat{\vartheta}\cup\{y\}}\langle\!\langle w \rangle\!\rangle$, and $\mathtt{L}_1 = \Box\mathscr{L}\{x\}$, where $\hat{\vartheta} \cup \{y\} = \mathsf{fz}^{\vartheta\cup\{y\}}(\mathtt{L}_1)$, the evaluation context $F^{\hat{\vartheta}\cup\{y\}}$ is in $\mathbb{X}^{\hat{\vartheta}\cup\{y\}}$, and $\mathscr{L}$ is a (ϑ, w)-chain context. We consider three further subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\hat{\vartheta}\cup\{y\}}$.
The remainder of this case is similar to 2.1.1 except when the hole of $\mathtt{C}$ and $F_{11}^{\hat{\vartheta}\cup\{y\}}$ are disjoint. Then there is a two hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle \Box, \mathtt{v} \rangle =$

$F_{11}^{\hat{\vartheta}\cup\{y\}}$ and $\widehat{\mathtt{C}}\langle w, \Box\rangle = \mathtt{C}$. The step $\mathtt{r}$ is of the form: $\mathtt{r} : \widehat{\mathtt{C}}\langle w, z\rangle[z\backslash \mathtt{vL}] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \widehat{\mathtt{C}}\langle w, \mathtt{v}\rangle[z\backslash \mathtt{v}]\mathtt{L}$. Note that w is bound by $\Box\mathscr{L}\{x\}[y\backslash M^{\vartheta}] = [z\backslash \mathtt{v}]\mathtt{L}$, since $\mathscr{L}$ is a (ϑ, w)-chain context. Hence it must be the case that $w = z$, for otherwise, if it were the case that $w \in \mathsf{dom}\mathtt{L}$, w would occur free on the left-hand side of the step $\mathtt{r}$, since it occurs outside the scope of $\mathtt{L}$. This is impossible since free variables cannot become bound after a reduction step.
Note that w must be bound to a term of the form $\mathtt{I}_1^{\vartheta_1}\langle\!\langle w_1 \rangle\!\rangle$ and, since we have just argued that $w = z$, we have that $\mathtt{I}_1^{\vartheta_1}\langle\!\langle w_1 \rangle\!\rangle = \mathtt{v}$. This is impossible since answers do not have variables below inert evaluation contexts (Lem. A.21).

2.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'[y\backslash M^{\vartheta}]$ Let $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} F_1^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle$ be the step isomorphic to $\mathtt{r}$ but going under the context $[y\backslash M^{\vartheta}]$. Note that $\mathtt{r}_1$ must be $(\vartheta\cup\{y\})$-internal, otherwise $\mathtt{r}$ would be $(\vartheta\cup\{y\})$-external. By *i.h.* there is an evaluation context $F_{10}^{\vartheta\cup\{y\}} \in \mathbb{X}^{\vartheta\cup\{y\}}$ such that $t_0' = F_{10}^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle$. It is immediate to conclude by taking $F_0^{\vartheta} := F_{10}^{\vartheta\cup\{y\}}[y\backslash M^{\vartheta}] \in \mathbb{X}^{\vartheta}$, since then $t_0 = F_{10}^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle[y\backslash M^{\vartheta}]$.

2.3 The internal step $\mathtt{r}$ is to the right of $t_0 = F_1^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle[y\backslash t_0']$ Let $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} M^{\vartheta}$. We consider two cases, depending on whether the step $\mathtt{r}_1$ is ϑ-internal or ϑ-internal:

2.3.1 If $\mathtt{r}_1$ is ϑ-external Two further subcases, depending on whether y is a structural variable in $F_1^{\vartheta\cup\{y\}}$ or not:

2.3.1.1 If $y \in \mathsf{sv}(F_1^{\vartheta\cup\{y\}})$ Since $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} M^{\vartheta}$ is a ϑ-external step, we can write $t_0' = F_3^{\vartheta}\langle\Sigma\rangle$ and $M^{\vartheta} = F_3^{\vartheta}\langle\Sigma'\rangle$ where Σ is the anchor of a redex, Σ' its contractum, and F_3^{ϑ} is an evaluation context $F_3^{\vartheta} \in \mathsf{E}_{\vartheta}$. Moreover, since structural variables are below evaluation contexts (Lem. A.54), there exists an evaluation context $F_2^{\vartheta} \in \mathbb{X}^{\vartheta}$ such that $F_1^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle = F_2^{\vartheta}\langle\!\langle y \rangle\!\rangle$. Hence the step $\mathtt{r}$ is of the form: $\mathtt{r} : F_2^{\vartheta}\langle\!\langle y \rangle\!\rangle[y\backslash F_3^{\vartheta}\langle\Sigma\rangle] \to_{\mathtt{sh}\backslash\mathtt{gc}} F_2^{\vartheta}\langle\!\langle y \rangle\!\rangle[y\backslash F_3^{\vartheta}\langle\Sigma'\rangle]$. If F_3^{ϑ} happens to be a inert evaluation context, *i.e.* $F_3^{\vartheta} \in \mathsf{E}_{\vartheta}^{\circ}$ then the composition $F_2^{\vartheta}\langle\!\langle y \rangle\!\rangle[y\backslash F_3^{\vartheta}]$ is a ϑ-evaluation context and $\mathtt{r}$ is a ϑ-external step, contradicting the hypothesis that it was internal.
So we may suppose that F_3^{ϑ} is *not* a inert evaluation context. By Lem. A.23 we know that evaluation contexts which are not inert evaluation contexts have the shape of an answer, that is, $F_3^{\vartheta}\langle * \rangle$ is an answer when filling the hole with an arbitrary term. In particular, $F_3^{\vartheta}\langle\Sigma'\rangle = M^{\vartheta}$ is both an answer and a structure, which is impossible.

2.3.1.2 If $y \notin \mathsf{sv}(F_1^{\vartheta\cup\{y\}})$ By the fact that non-structural variables are not required in "ϑ" (Lem. A.54), we have that $F_1^{\vartheta\cup\{y\}} \in \mathbb{X}^{\vartheta}$. Then, regardless of whether t_0' is a structure or not, adding an arbitrary substitution (Lem. A.26) we have $F_1^{\vartheta\cup\{y\}}[y\backslash t_0'] \in \mathbb{X}^{\vartheta}$. It is then immediate to conclude by taking $F_0^{\vartheta} := F_1^{\vartheta\cup\{y\}}[y\backslash t_0'] \in \mathbb{X}^{\vartheta}$, since indeed $t_0 = F_1^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle[y\backslash t_0']$.

2.3.2 If $\mathtt{r}_1$ is ϑ-internal Since structures are backward preserved by internal steps (Lem. A.69), $t_0' \in \mathsf{S}_{\vartheta}$. We conclude by taking $F_0^{\vartheta} := F_1^{\vartheta\cup\{y\}}[y\backslash t_0'] \in \mathbb{X}^{\vartheta}$, since $t_0 = F_1^{\vartheta\cup\{y\}}\langle\!\langle x \rangle\!\rangle[y\backslash t_0']$, as required.

3. ESubsR, $F^{\vartheta} = F_1^{\vartheta}\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{I}^{\vartheta}]$ where $F_1^{\vartheta} \in \mathbb{X}^{\vartheta}$ and $\mathtt{I}^{\vartheta} \in \mathsf{E}_{\vartheta}^{\circ}$ We consider three cases, depending on whether (1) the internal step $\mathtt{r}$ is at the root of t_0, (2) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to t_0', (3) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to r_0.

   3.1 The internal step $\mathtt{r}$ is at the root of t_0 Note that $\mathtt{r}$ cannot be a $\mathtt{db}$ redex, since it would be external. So $\mathtt{r}$ is a $\mathtt{lsv}$ redex of the form: $\mathtt{r} : t_0 = \mathtt{C}\langle\!\langle z \rangle\!\rangle[z\backslash \mathtt{vL}] \to \mathtt{C}\langle \mathtt{v} \rangle[z\backslash \mathtt{v}]\mathtt{L} = F_1^{\vartheta}\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{I}^{\vartheta}\langle\!\langle x \rangle\!\rangle] = t_1$. Let $\mathtt{L}_1$ be the substitution context such that $\mathtt{L}_1[y\backslash \mathtt{I}^{\vartheta}\langle\!\langle x \rangle\!\rangle] = [z\backslash \mathtt{v}]\mathtt{L}$, and using Lem. A.62 let us strip the substitution $\mathtt{L}_1$ from $F_1^{\vartheta}\langle\!\langle y \rangle\!\rangle$. This gives us two possibilities, **A** and **B**:

      3.1.1 Case A Then: $F_1^{\vartheta} = F_{11}^{\hat{\vartheta}}\mathtt{L}_1$ and $\mathtt{C}\langle \mathtt{v} \rangle = F_{11}^{\hat{\vartheta}}\langle\!\langle y \rangle\!\rangle$ where $\hat{\vartheta} = \mathsf{fz}^{\vartheta}(\mathtt{L}_1)$ and $F_{11}^{\hat{\vartheta}} \in \mathbb{X}^{\hat{\vartheta}}$.

      We consider three further subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\hat{\vartheta}}$.

         3.1.1.1 The hole of $\mathtt{C}$ and the hole of $F_{11}^{\hat{\vartheta}}$ are disjoint Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle \Box, \mathtt{v} \rangle = F_{11}^{\hat{\vartheta}}$ and $\widehat{\mathtt{C}}\langle y, \Box \rangle = \mathtt{C}$. Note that y is bound by the substitution context $\mathtt{L}_1[y\backslash \mathtt{I}^{\vartheta}\langle\!\langle x \rangle\!\rangle] = [z\backslash \mathtt{v}]\mathtt{L}$ on the right-hand side of the step $\mathtt{r}$. So it must be the case that $y = z$, for if we had $y \in \mathsf{dom}\mathtt{L}$, we would have that y is free on the left-hand side of the step $\mathtt{r}$, since it occurs outside the scope of the substitution $\mathtt{L}$. This is impossible, since a free variable cannot become bound along reduction.
         Also note that y is bound to $\mathtt{I}^{\vartheta}\langle\!\langle x \rangle\!\rangle$ and, since $y = z$, we have $\mathtt{I}^{\vartheta}\langle\!\langle x \rangle\!\rangle = \mathtt{v}$. This is impossible, since answers do not have variables below inert evaluation contexts (Lem. A.21).

         3.1.1.2 The context $\mathtt{C}$ is a prefix of $F_{11}^{\hat{\vartheta}}$ Then by the decomposition of evaluation contexts lemma (Lem. A.20) the context $\mathtt{C}$ must be an evaluation context in $\mathbb{X}^{\hat{\vartheta}}$. By strengthening ϑ (Tactic A.55), $\mathtt{C}[z\backslash \mathtt{vL}] \in \mathbb{X}^{\vartheta}$, contradicting the fact that the step $\mathtt{r}$ is ϑ-internal.

         3.1.1.3 The context $F_{11}^{\hat{\vartheta}}$ is a prefix of $\mathtt{C}$ Then $\mathtt{C} = F_{11}^{\hat{\vartheta}}\langle \mathtt{C}_1 \rangle$, so $\mathtt{C}_1\langle \mathtt{v} \rangle = y$, which is impossible.

      3.1.2 Case B Then: $F_1^{\vartheta} = F_{11}^{\hat{\vartheta}}\langle\!\langle w \rangle\!\rangle\mathscr{L}\{\Box\}$, $\mathtt{C}\langle \mathtt{v} \rangle = F_{11}^{\hat{\vartheta}}\langle\!\langle w \rangle\!\rangle$, and $\mathtt{L}_1 = \Box\mathscr{L}\{y\}$, where $\hat{\vartheta} = \mathsf{fz}^{\vartheta}(\mathtt{L}_1)$, the evaluation context $F_{11}^{\hat{\vartheta}}$ is in $\mathbb{X}^{\hat{\vartheta}}$, and $\mathscr{L}$ is a (ϑ, w)-chain context. We consider three further subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\hat{\vartheta}}$. The remainder of this case is similar to 3.1.1.

   3.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'[y\backslash \mathtt{I}^{\vartheta}]$ Let $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} F_1^{\vartheta}\langle\!\langle y \rangle\!\rangle$ be the step isomorphic to $\mathtt{r}$ but going under the context $[y\backslash \mathtt{I}^{\vartheta}\langle\!\langle x \rangle\!\rangle]$. Note that $\mathtt{r}_1$ must be ϑ-internal, for if it were ϑ-external, by adding an arbitrary substitution (Lem. A.26) it would contradict the fact that $\mathtt{r}$ is ϑ-internal.

   So we may apply the *i.h.* to obtain that there exists an evaluation context $F_{10}^{\vartheta} \in \mathbb{X}^{\vartheta}$ such that $t_0' = F_{10}^{\vartheta}\langle\!\langle y \rangle\!\rangle$. Applying the ESubsR rule and taking $F_0^{\vartheta} := F_{10}^{\vartheta}\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{I}^{\vartheta}] \in \mathbb{X}^{\vartheta}$, we have that $t_0 = F_{10}^{\vartheta}\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{I}^{\vartheta}\langle\!\langle x \rangle\!\rangle]$, as required.

3.3 The internal step $\mathtt{r}$ is to the right of $t_0 = F_1^\vartheta\langle\!\langle y \rangle\!\rangle[y\backslash t_0']$ Let $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} \mathtt{I}^\vartheta\langle\!\langle x \rangle\!\rangle$ be the step isomorphic to $\mathtt{r}$ but going under the context $F_1^\vartheta\langle\!\langle y \rangle\!\rangle[y\backslash\Box]$. Note that $\mathtt{r}_1$ cannot be ϑ-external, since then $\mathtt{r}$ would be ϑ-external. Hence $\mathtt{r}_1$ is ϑ-external, and we may apply the *i.h.* to obtain that there is a inert evaluation context $\mathtt{I}_0^\vartheta \in \mathsf{E}_\vartheta^\circ$ such that $t_0' = \mathtt{I}_0^\vartheta\langle\!\langle x \rangle\!\rangle$. Taking $F_0^\vartheta := F_1^\vartheta\langle\!\langle y \rangle\!\rangle[y\backslash\mathtt{I}_0^\vartheta] \in \mathbb{X}^\vartheta$, we have that $t_0 = F_1^\vartheta\langle\!\langle y \rangle\!\rangle[y\backslash\mathtt{I}_0^\vartheta\langle\!\langle x \rangle\!\rangle]$, as required.

4. EAppRStr, $F^\vartheta = M^\vartheta\, F_1^\vartheta$ where $M^\vartheta \in \mathsf{S}_\vartheta$ and $F_1^\vartheta \in \mathsf{E}_\vartheta$ We consider three cases, depending on whether (1) the internal step $\mathtt{r}$ is at the root of t_0, (2) t_0 is an application $t_0'\, r_0$ and the step $\mathtt{r}$ is internal to t_0', (3) t_0 is an application $t_0'\, r_0$ and the step $\mathtt{r}$ is internal to r_0.

4.1 The internal step $\mathtt{r}$ is at the root of t_0 This case is impossible: $\mathtt{r}$ cannot be a $\mathtt{db}$ step or a $\mathtt{lsv}$ step, since the right-hand side of both $\mathtt{db}$ and $\mathtt{lsv}$ steps is a substitution, not an application.

4.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'\, F_1^\vartheta\langle\!\langle x \rangle\!\rangle$ Let $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} M^\vartheta$ be the step isomorphic to $\mathtt{r}$ but going under the context $\Box\, F_1^\vartheta\langle\!\langle x \rangle\!\rangle$. Note that $\mathtt{r}_1$ must be ϑ-internal, otherwise $\mathtt{r}$ would be ϑ-external. Then since normal forms are backward preserved by internal steps (Lem. A.69), t_0' must be a strong ϑ-structure M_0^ϑ. By taking $F_0^\vartheta := M_0^\vartheta\, F_1^\vartheta \in \mathbb{X}^\vartheta$ we have that $t_0 = M_0^\vartheta\, F_1^\vartheta\langle\!\langle x \rangle\!\rangle$, as required.

4.3 The internal step $\mathtt{r}$ is to the right of $t_0 = M^\vartheta\, t_0'$ Let $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} F_1^\vartheta\langle\!\langle x \rangle\!\rangle$ be the step isomorphic to $\mathtt{r}$ but going under the context $M^\vartheta\, \Box$. By *i.h.* there is an evaluation context $F_{10}^\vartheta \in \mathsf{E}_\vartheta$ such that $t_0' = F_{10}^\vartheta\langle\!\langle x \rangle\!\rangle$. Taking $F_0^\vartheta := M^\vartheta\, F_{10}^\vartheta \in \mathbb{X}^\vartheta$ we have that $t_0 = M^\vartheta\, F_{10}^\vartheta\langle\!\langle x \rangle\!\rangle$, as required.

□

Postponement of internal steps

We turn to the proof of postponement of internal steps after external steps. The proof is long and by a heavy case analysis. For organizational purposes we split the proof in two lemmas: the first one (Lem. A.71) deals with the case in which the external step is a $\mathtt{db}$ step; the second one (Lem. A.72) deals with the case in which the external step is a $\mathtt{lsv}$ step. Finally in Lem. 4.50 we conclude and give the proof of Postponement itself (Lem. 4.50 in the main body).

Lemma A.71 (Postponement of internal steps after external $\mathtt{db}$ steps)**.** *Given any set of variables ϑ such that $\mathtt{fv}(t_0) \subseteq \vartheta$, if $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_1 \overset{\vartheta}{\rightsquigarrow} t_3$ where the second step is a $\mathtt{db}$ step, there exists a term t_2 such that $t_0 \overset{\vartheta}{\rightsquigarrow} t_2 \twoheadrightarrow_{\mathtt{sh}\backslash\mathtt{gc}} t_3$ where the first step is a $\mathtt{db}$ step. An explicit construction for the diagrams is given.*

Proof. Let us call $\mathtt{r}$ to the internal step $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_1$ and $\mathtt{r}'$ to the external $\mathtt{db}$ step $t_1 \overset{\vartheta}{\rightsquigarrow} t_3$. Then $t_1 = F^\vartheta\langle(\lambda x.s)\mathtt{L}\, u\rangle$ and $t_3 = F^\vartheta\langle s[x\backslash u]\mathtt{L}\rangle$. Throughout the proof we write Δ for the $\mathtt{db}$ redex $(\lambda x.s)\mathtt{L}\, u$ and Δ' for its contractum $s[x\backslash u]\mathtt{L}$. Let $\mathbb{X}^\vartheta$ denote either the set E_ϑ or the set $\mathsf{E}_\vartheta^\circ$. By induction on the derivation that $F^\vartheta \in \mathbb{X}^\vartheta$, the term t_0 will be shown to be of the form

$F_0^\vartheta\langle\Delta_0\rangle$, where $F_0^\vartheta \in \mathbb{X}^\vartheta$, and Δ_0 is a db redex, and $t_2 = F_0^\vartheta\langle\Delta_0'\rangle$, where Δ_0' is the contractum of Δ_0, in such a way that the diagram is closed as required by the statement.

1. EBox, $F^\vartheta = \Box \in \mathbb{X}^\vartheta$ Then there is a db redex at the root of t_1. By Lem. A.52, the internal step $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_1$ must be of the form $\mathtt{r} : t_0 = (\lambda x.s_0)\mathtt{L}_0\, u_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (\lambda x.s)\mathtt{L}\, u = t_1$ and the anchor of $\mathtt{r}$ must lie either inside s_0, inside u_0, or inside one of the arguments of $\mathtt{L}_0$. By taking $F_0^\vartheta := \Box$ we conclude.

2. EAppL, $F^\vartheta = \mathtt{I}^\vartheta\, r \in \mathbb{X}^\vartheta$ with $\mathtt{I}^\vartheta \in \mathsf{E}_\vartheta^\circ$ The situation is $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{I}^\vartheta\langle\Delta\rangle r \overset{\vartheta}{\rightsquigarrow} \mathtt{I}^\vartheta\langle\Delta'\rangle r$. We consider three cases: (1) the step $\mathtt{r}$ is at the root of t_0; (2) t_0 is an application $t_0 = t_0'\, r_0$ and the step $\mathtt{r}$ takes place inside t_0'; (3) t_0 is an application $t_0 = t_0'\, r_0$ and the step $\mathtt{r}$ takes place inside r_0.

   2.1 The internal step $\mathtt{r}$ is at the root of t_0 Impossible: $\mathtt{r}$ cannot be a db step, since it would be external, and it cannot be a lsv step.

   2.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'\, r_0$ Then there is a step $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\setminus\mathtt{gc}} \mathtt{I}^\vartheta\langle\Delta\rangle$. We consider two subcases, depending on whether $\mathtt{r}_1$ is ϑ-external or ϑ-internal.

      2.2.1 If $\mathtt{r}_1$ is ϑ-external Then t_0' is of the form $\widetilde{F}^\vartheta\langle\Sigma\rangle$ where $\widetilde{F}^\vartheta$ is an evaluation context in E_ϑ and Σ is the anchor of a redex. Note that $\widetilde{F}^\vartheta \notin \mathsf{E}_\vartheta^\circ$, *i.e.* it is not a inert evaluation context, since that would imply that $\widetilde{F}^\vartheta\, r \in \mathbb{X}^\vartheta$ and we would have that the step $\mathtt{r} : \widetilde{F}^\vartheta\langle\Sigma\rangle r \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{I}^\vartheta\langle\Delta\rangle r$ is external. Hence since $\widetilde{F}^\vartheta \in \mathsf{E}_\vartheta \setminus \mathsf{E}_\vartheta^\circ$ by Lem. A.23 we conclude that t_0' is of the form $\mathtt{v}_0\mathtt{L}_0$, *i.e.* an answer. Moreover, since answers are stable by reduction (Lem. A.22) we have that $\mathtt{I}^\vartheta\langle\Delta\rangle$ is an answer, and this is impossible since answers do not have redexes below inert evaluation contexts (Lem. A.21).

      2.2.2 If $\mathtt{r}_1$ is ϑ-internal Immediate by *i.h.*.

   2.3 The internal step $\mathtt{r}$ is to the right of $t_0 = t_0'\, r_0$ Then $t_0' = \mathtt{I}^\vartheta\langle\Delta\rangle$ and $r_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} r$. By taking $F_0^\vartheta := \mathtt{I}^\vartheta\, r_0 \in \mathbb{X}^\vartheta$ closing the diagram is immediate.

3. ESubLNonStr, $F^\vartheta = F_1^\vartheta[y\backslash r]$ with $y \notin \vartheta$, $r \notin \mathsf{S}_\vartheta$ and $F_1^\vartheta \in \mathbb{X}^\vartheta$ The situation is $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_1^\vartheta\langle\Delta\rangle[y\backslash r] \overset{\vartheta}{\rightsquigarrow} F_1^\vartheta\langle\Delta'\rangle[y\backslash r]$. There are three cases: (1) the step $\mathtt{r}$ is at the root of t_0; (2) t_0 is a substitution $t_0 = t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ takes place inside t_0'; (3) t_0 is a substitution $t_0 = t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ takes place inside r_0.

   3.1 The internal step $\mathtt{r}$ is at the root of t_0 Note that $\mathtt{r}$ cannot be a db step as it would be external. Suppose that $\mathtt{r}$ is a lsv step. Then $t_0 = \mathtt{C}\langle\!\langle z\rangle\!\rangle[z\backslash \mathtt{v}\mathtt{L}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle\mathtt{v}\rangle[z\backslash\mathtt{v}]\mathtt{L}' = F_1^\vartheta\langle\Delta\rangle[y\backslash r] \overset{\vartheta}{\rightsquigarrow} F_1^\vartheta\langle\Delta'\rangle[y\backslash r]$. We know that $\mathtt{C}\langle\mathtt{v}\rangle[z\backslash\mathtt{v}]\mathtt{L}' = F_1^\vartheta\langle\Delta\rangle[y\backslash r]$. The outermost substitution $[y\backslash r]$ is either $[z\backslash\mathtt{v}]$ (if $\mathtt{L}'$ is empty) or it is the outermost substitution in $\mathtt{L}'$. In any case, the substitution $[y\backslash r]$ is not part of $\mathtt{C}$.

   Let $\mathtt{L}_1$ be a substitution context such that $[z\backslash\mathtt{v}]\mathtt{L}' = \mathtt{L}_1[y\backslash r]$ and using Lem. A.62 let us strip the substitution $\mathtt{L}_1$ from $F_1^\vartheta\langle\Delta\rangle$. This gives us two possibilities, case **A** and case **B** in the statement of Lem. A.62:

3.1.1 Case A Then $F_1^{\vartheta} = F_{11}^{\vartheta'}\mathtt{L}_1$ in such a way that: $\mathtt{C}\langle \mathtt{v}\rangle = F_{11}^{\vartheta'}\langle\Delta\rangle$, where $\vartheta' = \mathsf{fz}^{\vartheta}(\mathtt{L}_1[y\backslash r]) = \mathsf{fz}^{\vartheta}([z\backslash \mathtt{v}]\mathtt{L}')$ and $F_{11}^{\vartheta'} \in \mathbb{X}^{\vartheta'}$. We consider three subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\vartheta'}$.

3.1.1.1 The hole of $\mathtt{C}$ and the hole of $F_{11}^{\vartheta'}$ are disjoint Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle = F_{11}^{\vartheta'}$ and $\widehat{\mathtt{C}}\langle\Delta, \Box\rangle = \mathtt{C}$ and the situation is $t_0 = \widehat{\mathtt{C}}\langle\Delta, z\rangle[z\backslash \mathtt{v}\mathtt{L}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \widehat{\mathtt{C}}\langle\Delta, \mathtt{v}\rangle[z\backslash\mathtt{v}]\mathtt{L}' = t_1 \overset{\vartheta}{\rightsquigarrow} \widehat{\mathtt{C}}\langle\Delta', \mathtt{v}\rangle[z\backslash\mathtt{v}]\mathtt{L}' = t_3$. Recall that $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle = F_{11}^{\vartheta'} \in \mathbb{X}^{\vartheta'}$ where $\vartheta' = \mathsf{fz}^{\vartheta}([z\backslash\mathtt{v}]\mathtt{L}')$. Note that $z \notin \vartheta'$ since the value $\mathtt{v}$ is not a strong structure. By Lem. A.58 there are two possibilities: the left and the right branch of the disjunction. The right branch case, $\widehat{\mathtt{C}}\langle\Delta, \Box\rangle \in \mathbb{X}^{\vartheta'}$, is impossible, as we would have that $\widehat{\mathtt{C}}\langle\Delta, \Box\rangle[z\backslash\mathtt{v}]\mathtt{L}' \in \mathbb{X}^{\vartheta}$, since $\mathsf{fz}^{\vartheta}([z\backslash\mathtt{v}]\mathtt{L}) = \vartheta'$. This implies that there are two different steps of the generalized call-by-need evaluation strategy under ϑ outgoing from t_1: one is the $\mathtt{db}$ step $t_1 = \widehat{\mathtt{C}}\langle\Delta, x\rangle[x\backslash\mathtt{v}]\mathtt{L}' \overset{\vartheta}{\rightsquigarrow} \widehat{\mathtt{C}}\langle\Delta', x\rangle[x\backslash\mathtt{v}]\mathtt{L}' = t_3$ and the other one is the $\mathtt{lsv}$ step: $t_1 = \widehat{\mathtt{C}}\langle\Delta, x\rangle[x\backslash\mathtt{v}]\mathtt{L}' \overset{\vartheta}{\rightsquigarrow} \widehat{\mathtt{C}}\langle\Delta, \mathtt{v}\rangle[x\backslash\mathtt{v}]\mathtt{L}'$. The coexistence of two different steps contradicts the fact that $\overset{\vartheta}{\rightsquigarrow}$ is a strategy (as shown in Lem. 4.17).
In the left branch case, $\widehat{\mathtt{C}}\langle\Box, z\rangle \in \mathbb{X}^{\vartheta'}$. Then we also have that $\widehat{\mathtt{C}}\langle\Box, z\rangle[z\backslash\mathtt{v}\mathtt{L}'] \in \mathbb{X}^{\vartheta}$, and it is immediate to close the diagram.

3.1.1.2 The context $\mathtt{C}$ is a prefix of $F_{11}^{\vartheta'}$ Let $F_{11}^{\vartheta'} = \mathtt{C}\langle\mathtt{C}'\rangle$. By the decomposition of evaluation contexts (Lem. A.20) $\mathtt{C} \in \mathbb{X}^{\vartheta'}$. By strengthening ϑ (Tactic A.55) we have that $\mathtt{C} \in \mathbb{X}^{\vartheta}$. Hence the step $\mathtt{r}$ is external, which is a contradiction.

3.1.1.3 The context $F_{11}^{\vartheta'}$ is a prefix of $\mathtt{C}$ Let $\mathtt{C} = F_{11}^{\vartheta'}\langle\mathtt{C}'\rangle$. Hence $\mathtt{C}'\langle\mathtt{v}\rangle = (\lambda x.s)\mathtt{L}u$. There are four possibilities for the position of the hole of $\mathtt{C}'$: inside s, inside one of the substitutions in $\mathtt{L}$, inside u, or right above $\lambda x.s$ (*i.e.* $\mathtt{C}' = \Box\mathtt{L}u$). In the first three cases it is easy to close the diagram. For example, if the hole of $\mathtt{C}'$ is inside s, *i.e.* $\mathtt{C}' = (\lambda x.\mathtt{C}'')\mathtt{L}u$ and $s = \mathtt{C}''\langle\mathtt{v}\rangle$, closing the diagram is straightforward noting that $F_{11}^{\vartheta'}[z\backslash\mathtt{v}\mathtt{L}']$ is a ϑ-evaluation context, by strengthening ϑ (Tactic A.55).
The last case is impossible: if the context $\mathtt{C}'$ is of the form $\mathtt{C}' = \Box\mathtt{L}u$, then by strengthening ϑ (Tactic A.55) we have that $F_{11}^{\vartheta'}\langle\Box\mathtt{L}s\rangle[z\backslash\mathtt{v}\mathtt{L}'] \in \mathbb{X}^{\vartheta}$. and we obtain that $\mathtt{r}$ is an external step.

3.1.2 Case B . Then $F_1^{\vartheta} = F_{11}^{\vartheta'}\langle\!\langle x'\rangle\!\rangle\mathscr{L}\{\Box\}$ in such a way that $\mathtt{C}\langle\mathtt{v}\rangle = F_{11}^{\vartheta'}\langle\!\langle x'\rangle\!\rangle$ and $\mathtt{L}_1 = \Box\mathscr{L}\{\Delta\}$, where $\vartheta' = \mathsf{fz}^{\vartheta}(\mathtt{L}_1[y\backslash r]) = \mathsf{fz}^{\vartheta}([z\backslash\mathtt{v}]\mathtt{L}')$, the evaluation context is $F_{11}^{\vartheta'} \in \mathbb{X}^{\vartheta'}$, and $\mathscr{L}$ is a (ϑ, x')-chain context. The remainder of this case is similar to case 3.1.1. Namely, we consider three subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\vartheta'}$.
The only difference with respect to case 3.1.1. is when the context $\mathtt{C}$ is a prefix of $F_{11}^{\vartheta'}$. Then $F_{11}^{\vartheta'} = \mathtt{C}\langle\mathtt{C}'\rangle$, so by Lem. A.20 $\mathtt{C} \in \mathbb{X}^{\vartheta'}$, and by strengthening ϑ (Tactic A.55) $\mathtt{C} \in \mathbb{X}^{\vartheta}$ which means that $\mathtt{r}$ is external.

3.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'[y\backslash r_0]$ Note that isomorphic step $t_0' \rightarrow_{\mathtt{sh}\backslash\mathtt{gc}} t_1'$ must be internal, so this case is immediate by *i.h.*.

3.3 The internal step $\mathtt{r}$ is to the right of $t_0 = t'_0[y\backslash r_0]$ Then it is immediate to close the diagram, recalling that adding an arbitrary substitution preserves evaluation contexts (Lem. A.26).

4. ESubLStr, $F^{\vartheta} = F_1^{\vartheta\cup\{y\}}[y\backslash r]$ with $r \in \mathsf{S}_{\vartheta}$ and $F^{\vartheta\cup\{y\}} \in \mathbb{X}^{\vartheta\cup\{y\}}$ The situation is $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_1^{\vartheta\cup\{y\}}\langle\Delta\rangle[y\backslash r] = t_1 \overset{\vartheta}{\rightsquigarrow} F_1^{\vartheta\cup\{y\}}\langle\Delta'\rangle[y\backslash r] = t_3$. There are three cases: (1) the step $\mathtt{r}$ is at the root of t_0; (2) t_0 is a substitution $t'_0[y\backslash r_0]$ and the step $\mathtt{r}$ takes place inside t'_0; (3) t_0 is a substitution $t'_0[y\backslash r_0]$ and the step $\mathtt{r}$ takes place inside r_0.

4.1 The internal step $\mathtt{r}$ is at the root of t_0 Note that $\mathtt{r}$ cannot be a $\mathtt{db}$ step, as that would be an external step. Suppose then that $\mathtt{r}$ is a $\mathtt{lsv}$ step. The situation is $t_0 = \mathtt{C}\langle\!\langle z\rangle\!\rangle[z\backslash \mathtt{v}\mathtt{L}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle\mathtt{v}\rangle[z\backslash\mathtt{v}]\mathtt{L}' = F_1^{\vartheta\cup\{y\}}\langle\Delta\rangle[y\backslash r] = t_1 \overset{\vartheta}{\rightsquigarrow} F_1^{\vartheta\cup\{y\}}\langle\Delta'\rangle[y\backslash r] = t_3$. We know that $\mathtt{C}\langle\mathtt{v}\rangle[z\backslash\mathtt{v}]\mathtt{L}' = F_1^{\vartheta\cup\{y\}}\langle(\lambda x.s)\mathtt{L}\,u\rangle[y\backslash r]$. Note that $\mathtt{L}'$ cannot be empty since the outermost substitution $[y\backslash r]$ cannot coincide with $[z\backslash\mathtt{v}]$, given that $r \in \mathsf{S}_{\vartheta}$ is a structure, and therefore it cannot be a value like $\mathtt{v}$.

Let $\mathtt{L}_1$ be a substitution context such that $[z\backslash\mathtt{v}]\mathtt{L}' = \mathtt{L}_1[y\backslash r]$, and using Lem. A.62 let us strip the substitution $\mathtt{L}_1$ from $F_1^{\vartheta\cup\{y\}}\langle\Delta\rangle$. This gives us two possibilities, case **A** and case **B** in the statement of Lem. A.62:

4.1.1 Case A Then $F_1^{\vartheta\cup\{y\}} = F_{11}^{\vartheta'\cup\{y\}}\mathtt{L}_1$ in such a way that $[z\backslash\mathtt{v}]\mathtt{L}' = \mathtt{L}_1[y\backslash r]$ and $\mathtt{C}\langle\mathtt{v}\rangle = F_{11}^{\vartheta'\cup\{y\}}\langle(\lambda x.s)\mathtt{L}\,u\rangle$, where $\vartheta'\cup\{y\} = \mathsf{fz}^{\vartheta\cup\{y\}}(\mathtt{L}_1[y\backslash r]) = \mathsf{fz}^{\vartheta\cup\{y\}}([z\backslash\mathtt{v}]\mathtt{L}')$ and $F_{11}^{\vartheta'\cup\{y\}} \in \mathbb{X}^{\vartheta'\cup\{y\}}$. We consider three subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\vartheta'\cup\{y\}}$. These are similar to the three subcases of 3.1.1.

4.1.2 Case B Then $F_1^{\vartheta\cup\{y\}} = F_{11}^{\vartheta'\cup\{y\}}\langle\!\langle x'\rangle\!\rangle\mathscr{L}\{\Box\}$ such that $\mathtt{C}\langle\mathtt{v}\rangle = F_{11}^{\vartheta'\cup\{y\}}\langle\!\langle x'\rangle\!\rangle$ and $\Box\mathscr{L}\{\Delta\} = \mathtt{L}_1$, where $\vartheta'\cup\{y\} = \mathsf{fz}^{\vartheta\cup\{y\}}(\mathtt{L}_1)$, the evaluation context $F_{11}^{\vartheta'\cup\{y\}}$ is in $\mathbb{X}^{\vartheta'\cup\{y\}}$, and $\mathscr{L}$ is a $(\vartheta\cup\{y\}, x')$-chain context. We consider three subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\vartheta'\cup\{y\}}$. These are similar to the three subcases of 3.1.2.

4.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t'_0[y\backslash r]$ Note that the step $t'_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} t'_1$ must be $(\vartheta\cup\{y\})$-internal, for otherwise the step at the top of the diagram $\mathtt{r} : t'_0[y\backslash r] \to_{\mathtt{sh}\backslash\mathtt{gc}} t'_1[y\backslash r]$ would be a ϑ-external step. Then it is straightforward to conclude by *i.h.*.

4.3 The internal step $\mathtt{r}$ is to the right of $t_0 = t'_0[y\backslash r_0]$ Here $r_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} r$ and $t'_0 = F_1^{\vartheta\cup\{y\}}\langle\Delta\rangle$. We consider two cases, depending on whether the step $\mathtt{r}_1 : r_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} r$ is ϑ-external or ϑ-internal:

4.3.1 If $\mathtt{r}_1$ is ϑ-external Two further subcases, depending on whether $y \in \mathsf{sv}(F_1^{\vartheta\cup\{y\}})$ or not:

4.3.1.1 If $y \in \mathsf{sv}(F_1^{\vartheta\cup\{y\}})$ Since $\mathtt{r}_1 : r_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} r$ is a ϑ-external step, we can write $r_0 = F_3^{\vartheta}\langle\Sigma\rangle$ and $r = F_3^{\vartheta}\langle\Sigma'\rangle$ where Σ is the anchor of $\mathtt{r}_1$ and Σ' is its contractum, and moreover F_3^{ϑ} is an evaluation context $F_3^{\vartheta} \in \mathsf{E}_{\vartheta}$. Moreover, since structural variables are below evaluation contexts (Lem. A.54), there is an evaluation context $F_2^{\vartheta} \in \mathbb{X}^{\vartheta}$ such that $F_1^{\vartheta\cup\{y\}}\langle\Delta\rangle = F_2^{\vartheta}\langle\!\langle y\rangle\!\rangle$. Hence

the step $\mathtt{r}$ at the top of the diagram is of the form $\mathtt{r} : t_0 = F_2^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash F_3^\vartheta\langle\Sigma\rangle] \to_{\mathtt{sh}\backslash\mathtt{gc}} F_2^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash F_3^\vartheta\langle\Sigma'\rangle] = t_1$. If F_3^ϑ happens to be a inert evaluation context, *i.e.* $F_3^\vartheta \in \mathsf{E}_\vartheta^\circ$ then the composition $F_2^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash F_3^\vartheta]$ is a ϑ-evaluation context and $\mathtt{r}$ is a ϑ-external step, contradicting the hypothesis that it was internal.

So we may suppose that F_3^ϑ is *not* a inert evaluation context. By Lem. A.23, evaluation contexts which are not inert evaluation contexts have the shape of an answer. In particular $F_3^\vartheta\langle\Sigma'\rangle = (\lambda x'.t')\mathtt{L}''$ and we have a ϑ-external step: $\mathtt{r}_2 : t_1 = F_2^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash(\lambda x'.t')\mathtt{L}''] \overset{\vartheta}{\rightsquigarrow} F_2^\vartheta\langle\lambda x'.t'\rangle[y\backslash\lambda x'.t']\mathtt{L}''$. Hence t_1 has two distinct external steps, namely $\mathtt{r}'$ and $\mathtt{r}_2$. This is impossible as a consequence of the unique decomposition lemma (Lem. 4.17).

4.3.1.2 If $y \notin \mathsf{sv}(F_1^{\vartheta\cup\{y\}})$ Then by Lem. A.54 we have that $F_1^{\vartheta\cup\{y\}} \in \mathbb{X}^\vartheta$, so $F_1^{\vartheta\cup\{y\}}[y\backslash r_0] \in \mathbb{X}^\vartheta$, regardless of whether r_0 is a ϑ-structure or not. Then it is straightforward to close the diagram.

4.3.2 If $\mathtt{r}_1$ is ϑ-internal Since normal forms are backward preserved by internal steps (Lem. A.69), $\mathtt{r}_0$ is a structure; more precisely $\mathtt{r}_0 \in \mathsf{S}_\vartheta$. This allows us to conclude that $F_1^{\vartheta\cup\{y\}}[y\backslash r_0] \in \mathbb{X}^\vartheta$, and it is straightforward to close the diagram.

5. ESubsR, $F^\vartheta = F_1^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{I}^\vartheta]$, where $F_1^\vartheta \in \mathbb{X}^\vartheta$ and $\mathtt{I}^\vartheta \in \mathsf{E}_\vartheta^\circ$ The situation is $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_1^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{I}^\vartheta\langle\Delta\rangle] = t_1 \overset{\vartheta}{\rightsquigarrow} F_1^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{I}^\vartheta\langle\Delta'\rangle] = t_3$. There are three cases: (1) the step $\mathtt{r}$ is at the root of t_0; (2) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step takes place inside t_0'; (3) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step takes place inside r_0.

5.1 The internal step $\mathtt{r}$ is at the root of t_0 Note that $\mathtt{r}$ cannot be a $\mathtt{db}$ step since then it would be ϑ-external, so $\mathtt{r}$ must be a $\mathtt{lsv}$ step $t_0 = \mathtt{C}\langle\!\langle z\rangle\!\rangle[z\backslash\mathtt{vL}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle\!\langle z\rangle\!\rangle[z\backslash\mathtt{v}]\mathtt{L}' = F_1^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{I}^\vartheta\langle\Delta\rangle] = t_1$. Let $\mathtt{L}_1$ be a substitution context such that $[z\backslash\mathtt{v}]\mathtt{L}' = \mathtt{L}_1[y\backslash\mathtt{I}^\vartheta\langle\Delta\rangle]$, and using Lem. A.62 let us strip the substitution $\mathtt{L}_1$ from $F_1^\vartheta\langle\!\langle y\rangle\!\rangle$. This gives us two possibilities, case **A** and case **B** in the statement of Lem. A.62:

5.1.1 Case A Then $F_1^\vartheta = F_{11}^{\vartheta'}\mathtt{L}_1$ such that $F_{11}^{\vartheta'}\langle\!\langle y\rangle\!\rangle = \mathtt{C}\langle\mathtt{v}\rangle$ where $\vartheta' = \mathsf{fz}^\vartheta(\mathtt{L}_1)$ and the evaluation context $F_{11}^{\vartheta'}$ is in $\mathbb{X}^{\vartheta'}$. We consider three subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{11}^{\vartheta'}$.

5.1.1.1 The hole of $\mathtt{C}$ and the hole of $F_{11}^{\vartheta'}$ are disjoint Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box,\mathtt{v}\rangle = F_{11}^{\vartheta'}$ and $\widehat{\mathtt{C}}\langle y,\Box\rangle = \mathtt{C}$. So the starting term t_0 is of the form: $t_0 = \mathtt{C}\langle\!\langle z\rangle\!\rangle[z\backslash\mathtt{vL}'] = \widehat{\mathtt{C}}\langle y,z\rangle[z\backslash\mathtt{vL}'] = F_{11}^{\vartheta'}\langle\!\langle y\rangle\!\rangle[z\backslash\mathtt{vL}']$ and $t_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} t_1$. The variable y occurs bound in t_1, so it must also occur bound in t_0, which means that $y = z$. Since $\mathtt{L}_1[y\backslash\mathtt{I}^\vartheta\langle\Delta\rangle] = [z\backslash\mathtt{v}]\mathtt{L}'$ we have that $\mathtt{I}^\vartheta\langle\Delta\rangle = \mathtt{v}$. This is impossible since answers do not have redexes under inert evaluation contexts (Lem. A.21).

5.1.1.2 The context $\mathtt{C}$ is a prefix of $F_{11}^{\vartheta'}$ Then $F_{11}^{\vartheta'} = \mathtt{C}\langle\mathtt{C}'\rangle$, so by the decomposition lemma for evaluation contexts (Lem. A.20) $\mathtt{C} \in \mathbb{X}^{\vartheta'}$. By strengthening ϑ

(Tactic A.55), $\mathtt{C}[z\backslash \mathtt{vL}'] \in \mathbb{X}^{\vartheta}$. This contradicts the fact that $\mathtt{r}$ is an internal step.

5.1.1.3 The context $F_{11}^{\vartheta'}$ is a prefix of $\mathtt{C}$ Then $\mathtt{C} = F_{11}^{\vartheta'}\langle \mathtt{C}' \rangle$. Given that $\mathtt{C}\langle \mathtt{v} \rangle = F_{11}^{\vartheta}\langle\!\langle y \rangle\!\rangle$ we have that $\mathtt{v} = y$, which is impossible.

5.1.2 Case B Then $F_1^{\vartheta} = F_{11}^{\vartheta'}\langle\!\langle x' \rangle\!\rangle \mathscr{L}\{\Box\}$ such that $F_{11}^{\vartheta'}\langle\!\langle x' \rangle\!\rangle = \mathtt{C}\langle \mathtt{v} \rangle$ and $\mathtt{L}_1 = \Box \mathscr{L}\{y\}$ where $\vartheta' = \mathsf{fz}^{\vartheta}([z\backslash \mathtt{v}]\mathtt{L}') = \mathsf{fz}^{\vartheta}(\mathtt{L}_1[y\backslash \mathtt{I}^{\vartheta}\langle \Delta \rangle])$, the evaluation context $F_{11}^{\vartheta'}$ is in $\mathbb{X}^{\vartheta'}$, and $\mathscr{L}$ is a (ϑ, x')-chain context. The remainder of this case is similar to the previous item 5.1.1. For case 5.1.1.1, recall that answers do not have variables under inert evaluation contexts (Lem. A.21).

5.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'[y\backslash r_0]$ Then there is a step $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash \mathtt{gc}} F_1^{\vartheta}\langle\!\langle y \rangle\!\rangle$. The step $\mathtt{r}_1$ must be ϑ-internal, otherwise $\mathtt{r}$ would be ϑ-external. Since $\mathtt{r}_1$ is internal, by Lem. A.70 we have that t_0' is of the form $F_0^{\vartheta}\langle\!\langle y \rangle\!\rangle$, where F_0^{ϑ} is an evaluation context in $\mathbb{X}^{\vartheta}$. Then it is immediate to close the diagram.

5.3 The internal step $\mathtt{r}$ is to the right of $t_0 = t_0'[y\backslash r_0]$ Then there is a step $\mathtt{r}_1 : r_0 \to_{\mathtt{sh}\backslash \mathtt{gc}} \mathtt{I}^{\vartheta}\langle \Delta' \rangle$. We consider two subcases, depending on whether $\mathtt{r}_1$ is ϑ-external or ϑ-internal:

5.3.1 If $\mathtt{r}_1$ is ϑ-external Then its source r_0 is of the form $r_0 = \widetilde{F}^{\vartheta}\langle \Sigma \rangle$ where $\widetilde{F}^{\vartheta} \in \mathsf{E}_{\vartheta}$ is an evaluation context and Σ is the anchor of a redex. Moreover, $\widetilde{F}^{\vartheta} \notin \mathsf{E}_{\vartheta}^{\circ}$, *i.e.* it cannot be a inert evaluation context, since otherwise we would have that $\mathtt{r}$ is external. So given that $\widetilde{F}^{\vartheta} \in \mathsf{E}_{\vartheta} \backslash \mathsf{E}_{\vartheta}^{\circ}$ by Lem. A.23 we conclude that r_0 is of the form $r_0 = \mathtt{v}_0\mathtt{L}_0$, *i.e.* an answer. By the fact that answers are stable by reduction (Lem. A.22) this means that $\mathtt{I}^{\vartheta}\langle \Delta' \rangle$ is also an answer, which contradicts the fact that answers do not have redexes below inert evaluation contexts (Lem. A.21).

5.3.2 If $\mathtt{r}_1$ is ϑ-internal Immediate by *i.h.*.

6. EAppRStr, $F^{\vartheta} = r\, F_1^{\vartheta}$, where $r \in \mathsf{S}_{\vartheta}$ and $F_1^{\vartheta} \in \mathsf{E}_{\vartheta}$ The situation is $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} r\, F_1^{\vartheta}\langle \Delta \rangle = t_1 \overset{\vartheta}{\rightsquigarrow} r\, F_1^{\vartheta}\langle \Delta' \rangle = t_3$. There are three cases: (1) the step $\mathtt{r}$ is at the root of t_0; (2) t_0 is an application $r_0\, t_0'$ and the step takes place inside r_0; (3) t_0 is a substitution $r_0\, t_0'$ and the step takes place inside t_0'.

6.1 The internal step $\mathtt{r}$ is at the root of t_0 This case is impossible. Note that $\mathtt{r}$ cannot be a $\mathtt{db}$ step at the root, since it would be an external step. Moreover, $\mathtt{r}$ cannot be a $\mathtt{lsv}$ step at the root, since then the outermost constructor of $t_1 = r\, F_1^{\vartheta}\langle \Delta \rangle$ would be a substitution, but it is an application.

6.2 The internal step $\mathtt{r}$ is to the left of $t_0 = r_0\, t_0'$ Then there is a step $\mathtt{r}_1 : r_0 \to_{\mathtt{sh}\backslash \mathtt{gc}} r$. It cannot be ϑ-external, for this would imply that $\mathtt{r}$ is ϑ-external. Note that $r \in \mathsf{S}_{\vartheta}$, so by Lem. A.69 we have that $r_0 \in \mathsf{S}_{\vartheta}$. Then it is immediate to close the diagram.

6.3 The internal step $\mathtt{r}$ is to the right of $t_0 = r_0\, t_0'$ There is a step $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash \mathtt{gc}} F_1^{\vartheta}\langle \Delta \rangle$. It cannot be ϑ-external, as this would imply that $\mathtt{r}$ is also ϑ-external. Then it is straightforward to conclude by *i.h.*.

7. ELAM, $F^{\vartheta} = \lambda y.F^{\vartheta \cup \{y\}}$ Straightforward by *i.h.*.

□

Lemma A.72 (Postponement of internal steps after external `lsv` steps)**.** *Given any set of variables ϑ such that $\mathtt{fv}(t_0) \subseteq \vartheta$, if $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_1 \overset{\vartheta}{\rightsquigarrow} t_3$ where the second step is a `lsv` step, there exists a term t_2 such that $t_0 \overset{\vartheta}{\rightsquigarrow} t_2 \twoheadrightarrow_{\mathtt{sh}\backslash\mathtt{gc}} t_3$ where the first step is a `lsv` step. An explicit construction for the diagrams is given.*

Proof. Let $\mathbb{X}^{\vartheta}$ denote either the set E_{ϑ} or the set $\mathsf{E}^{\circ}_{\vartheta}$. Let us call $\mathtt{r}$ to the internal step $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_1$ and $\mathtt{r}'$ to the external `lsv` step $t_1 \overset{\vartheta}{\rightsquigarrow} t_3$. Then $t_1 = F_1^{\vartheta}\langle F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}]\rangle$ and $t_3 = F_1^{\vartheta}\langle F_2^{\vartheta'}\langle \mathtt{v}\rangle[x\backslash\mathtt{v}]\mathtt{L}\rangle$, where $F_1^{\vartheta}\langle F_2^{\vartheta'}[x\backslash\mathtt{vL}]\rangle \in \mathbb{X}^{\vartheta}$. We write Δ to stand for the `lsv` redex $F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash\mathtt{vL}]$ and Δ' for its contractum $F_2^{\vartheta'}\langle \mathtt{v}\rangle[x\backslash\mathtt{v}]\mathtt{L}$.

By induction on the derivation that $F_1^{\vartheta} \in \mathbb{X}^{\vartheta}$, the term t_0 will be shown to be of the form $F_{10}^{\vartheta}\langle F_{20}^{\vartheta''}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{v}_0\mathtt{L}_0]\rangle$, where $F_{10}^{\vartheta}\langle F_{20}^{\vartheta''}[x\backslash\mathtt{v}_0\mathtt{L}_0]\rangle \in \mathbb{X}^{\vartheta}$, and then $t_2 = F_{10}^{\vartheta}\langle F_{20}^{\vartheta''}\langle \mathtt{v}_0\rangle[x\backslash\mathtt{v}_0]\mathtt{L}_0\rangle$, in such a way that the diagram is closed as required by the statement. We write Δ_0 to stand for the `lsv` redex $F_{20}^{\vartheta''}\langle\!\langle x \rangle\!\rangle[x\backslash\mathtt{v}_0\mathtt{L}_0]$ and Δ_0' for its contractum $F_{20}^{\vartheta''}\langle \mathtt{v}\rangle[x\backslash\mathtt{v}_0]\mathtt{L}_0$. Furthermore, suppose that $F_2^{\vartheta'} \in \mathbb{Y}^{\vartheta'}$. Then the inductive construction will ensure that $F_{20}^{\vartheta''} \in \mathbb{Y}^{\vartheta''}$, where $\mathbb{Y}^{\vartheta}$ is either E_{ϑ} or $\mathsf{E}^{\circ}_{\vartheta}$.

1. EBOX, $F_1^{\vartheta} = \Box \in \mathbb{X}^{\vartheta}$ Note that in this case $\vartheta' = \mathsf{fz}^{\vartheta}(\Box) = \vartheta$. Then there is a `lsv` redex at the root of $t_1 = F_2^{\vartheta}\langle\!\langle x \rangle\!\rangle[x\backslash\mathtt{vL}]$. We consider three cases: (1) the step $\mathtt{r} : t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_1$ is at the root of t_0; (2) t_0 is a substitution $t_0'[x\backslash s_0]$ and $\mathtt{r}$ is internal to t_0'; (3) t_0 is a substitution $t_0'[x\backslash s_0]$ and $\mathtt{r}$ is internal to s_0.

   1.1 The internal step $\mathtt{r}$ is at the root of t_0 Note that $\mathtt{r}$ cannot be a `db` step, since it would be external. So $\mathtt{r}$ is a `lsv` step, *i.e.* $t_0 = \mathtt{C}\langle\!\langle y \rangle\!\rangle[y\backslash\mathtt{v}'\mathtt{L}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle \mathtt{v}'\rangle[y\backslash\mathtt{v}']\mathtt{L}' = t_1 = F_2^{\vartheta}\langle\!\langle x \rangle\!\rangle[x\backslash\mathtt{vL}]$. Let $\mathtt{L}_1$ be a substitution context such that $[y\backslash\mathtt{v}']\mathtt{L}' = \mathtt{L}_1[x\backslash\mathtt{vL}]$, and using Lem. A.62 let us strip $\mathtt{L}_1$ from $F_2^{\vartheta}\langle\!\langle x \rangle\!\rangle$. This gives us two possibilities, case **A** and case **B** in the statement of Lem. A.62:

      1.1.1 Case A Then $F_2^{\vartheta}\langle\!\langle x \rangle\!\rangle = F_{21}^{\vartheta''}\langle\!\langle x \rangle\!\rangle\mathtt{L}_1$ where $\vartheta'' = \mathsf{fz}^{\vartheta}(\mathtt{L}_1)$, the evaluation context $F_{21}^{\vartheta''}$ is in $\mathbb{X}^{\vartheta''}$ and we have $\mathtt{C}\langle \mathtt{v}'\rangle = F_{21}^{\vartheta''}\langle\!\langle x \rangle\!\rangle$. We consider three further subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{21}^{\vartheta''}$.

         1.1.1.1 The hole of $\mathtt{C}$ and the hole of $F_{21}^{\vartheta''}$ are disjoint Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}\rangle = F_{21}^{\vartheta''}$ and $\widehat{\mathtt{C}}\langle x, \Box\rangle = \mathtt{C}$. By Lem. A.58 there are two possibilities for $\widehat{\mathtt{C}}$: the left and the right branch of the disjunction. The right branch, $\mathtt{C} = \widehat{\mathtt{C}}\langle x, \Box\rangle \in \mathbb{X}^{\vartheta''}$, is impossible since by strengthening ϑ (Tactic A.55), $\mathtt{C}[y\backslash\mathtt{v}'\mathtt{L}'] \in \mathbb{X}^{\vartheta}$, which contradicts the fact that $\mathtt{r}$ is external.
         In the left branch case, $\widehat{\mathtt{C}}\langle\Box, y\rangle \in \mathbb{X}^{\vartheta''}$. By strengthening ϑ (Tactic A.55), $\widehat{\mathtt{C}}\langle\Box, y\rangle \in \mathbb{X}^{\vartheta}$. Note also that x is bound by $\mathtt{L}_1[x\backslash\mathtt{vL}] = [y\backslash\mathtt{v}']\mathtt{L}'$, and that it must occur bound in $t_0 = \widehat{\mathtt{C}}\langle x, y\rangle[y\backslash\mathtt{v}'\mathtt{L}']$, since free variables cannot become bound. So x it must be bound by $[y\backslash\mathtt{v}'\mathtt{L}']$, which means that $x = y$,

and in particular $\mathtt{v} = \mathtt{v}'$ and $\mathtt{L} = \mathtt{L}'$. Then it is immediate to close the diagram.

1.1.1.2 The context $\mathtt{C}$ is a prefix of $F_{21}^{\vartheta''}$ Then $F_{21}^{\vartheta''} = \mathtt{C}\langle\mathtt{C}'\rangle$, so by Lem. A.20 $\mathtt{C}$ must be an evaluation context in $\mathbb{X}^{\vartheta''}$. By the fact that non-structural variables are not required in "ϑ" (Lem. A.54) we obtain that $\mathtt{C} \in \mathbb{X}^{\vartheta}$. This contradicts the hypothesis that $\mathtt{r}$ is an internal step.

1.1.1.3 The context $F_{21}^{\vartheta''}$ is a prefix of $\mathtt{C}$ Then $\mathtt{C} = F_{21}^{\vartheta''}\langle\mathtt{C}'\rangle$. Since $\mathtt{C}\langle\mathtt{v}'\rangle = F_{21}^{\vartheta''}\langle\!\langle x\rangle\!\rangle$ this implies that $\mathtt{v}' = x$, which is impossible.

1.1.2 Case B Then $F_2^{\vartheta}\langle\!\langle x\rangle\!\rangle = F_{21}^{\vartheta''}\langle\!\langle z\rangle\!\rangle\mathscr{L}\{x\}$ such that $\mathtt{C}\langle\mathtt{v}'\rangle = F_{21}^{\vartheta''}\langle\!\langle z\rangle\!\rangle$ and $\Box\mathscr{L}\{x\} = \mathtt{L}_1$, where $\vartheta'' = \mathsf{fz}^{\vartheta}(\mathtt{L}_1)$, the evaluation context $F_{21}^{\vartheta''}$ is in $\mathbb{X}^{\vartheta''}$, and $\mathscr{L}$ is a (ϑ, z)-chain context. The remainder of this case is by case analysis on the relative positions of the hole of $\mathtt{C}$ and the hole of $F_{21}^{\vartheta''}$, similar to item 1.1.1.

1.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'[x\backslash s_0]$ Then there is a step $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} F_2^{\vartheta}\langle\!\langle x\rangle\!\rangle$. Note that $\mathtt{r}_1$ cannot be ϑ-external, for this would imply that $\mathtt{r}$ is ϑ-external. Hence $\mathtt{r}_1$ is ϑ-internal, so given that evaluation contexts are backward preserved by internal steps (Lem. A.70), there is an evaluation context $F_{20}^{\vartheta} \in \mathbb{X}^{\vartheta}$ such that $t_0' = F_{20}^{\vartheta}\langle\!\langle x\rangle\!\rangle$. Then it is immediate to close the diagram.

1.3 The internal step $\mathtt{r}$ is to the right of $t_0 = t_0'[x\backslash s_0]$ Then there is a step $\mathtt{r}_1 : s_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} \mathtt{vL}$. We consider two cases, depending on whether $\mathtt{r}_1$ is a ϑ-external or a ϑ-internal step:

1.3.1 If $\mathtt{r}_1$ is ϑ-external Then s_0 is of the form $\widetilde{F}^{\vartheta}\langle\Sigma\rangle$, where $\widetilde{F}^{\vartheta}$ is an evaluation context in E_{ϑ} and Σ is the anchor of a redex. Note that $\widetilde{F}^{\vartheta}$ is an evaluation context but it is not a inert evaluation context, *i.e.* $\widetilde{F}^{\vartheta} \notin \mathsf{E}_{\vartheta}^{\circ}$, since if we had $\widetilde{F}^{\vartheta} \in \mathsf{E}_{\vartheta}^{\circ}$ then the context $F_2^{\vartheta}\langle\!\langle x\rangle\!\rangle[x\backslash\widetilde{F}^{\vartheta}]$ would be an evaluation context, and the step $\mathtt{r}$ would be external, contradicting the hypothesis that it is internal. Then since $\widetilde{F}^{\vartheta} \in \mathsf{E}_{\vartheta}\backslash\mathsf{E}_{\vartheta}^{\circ}$ by the Lem. A.23 we may conclude that $\widetilde{F}^{\vartheta}\langle\Sigma\rangle = \mathtt{v}_0\mathtt{L}_0$, and it is immediate to close the diagram.

1.3.2 If $\mathtt{r}_1$ is ϑ-internal Then since answers are backward stable by internal steps (Lem. A.51), s_0 is of the form $s_0 = \mathtt{v}_0\mathtt{L}_0$, and the diagram can be closed just as in the previous case.

2. EAppL, $F_1^{\vartheta} = \mathtt{I}^{\vartheta}\, t$ The situation is $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{I}^{\vartheta}\langle\Delta\rangle t = t_1 \overset{\vartheta}{\rightsquigarrow} \mathtt{I}^{\vartheta}\langle\Delta'\rangle t = t_3$, where $\mathtt{I}^{\vartheta}\langle F_2^{\vartheta'}[x\backslash\mathtt{vL}]\rangle$ is a inert evaluation context in $\mathsf{E}_{\vartheta}^{\circ}$. This case is analogous to item 2 of the previous lemma (Lem. A.71), as the proof does not rely on Δ being a $\mathtt{db}$ redex.

3. ESubLNonStr, $F_1^{\vartheta} = F_{11}^{\vartheta}[y\backslash t]$, where $y \notin \vartheta$, $t \notin \mathsf{S}_{\vartheta}$, and $F_{11}^{\vartheta} \in \mathbb{X}^{\vartheta}$ The situation is $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_{11}^{\vartheta}\langle\Delta\rangle[y\backslash t] = t_1 \overset{\vartheta}{\rightsquigarrow} F_{11}^{\vartheta}\langle\Delta'\rangle[y\backslash t] = t_3$. We consider three cases, depending on whether (1) the internal step $\mathtt{r}$ is at the root of t_0, (2) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to t_0', (3) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to r_0.

3.1 The internal step $\mathtt{r}$ is at the root of t_0 Then $\mathtt{r}$ cannot be a $\mathtt{db}$ step, since it would be external. So it must be a $\mathtt{lsv}$ step. Then the step $\mathtt{r}$ is of the form: $t_0 =$

$\mathtt{C}\langle\!\langle z \rangle\!\rangle[z\backslash \mathtt{v}'\mathtt{L}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle \mathtt{v}'\rangle[z\backslash \mathtt{v}']\mathtt{L}' = F_{11}^{\vartheta}\langle\Delta\rangle[y\backslash t] = t_1$. Let $\mathtt{L}_1$ be a substitution context such that $\mathtt{L}_1[y\backslash t] = [z\backslash \mathtt{v}']\mathtt{L}'$. Recall that $\Delta = F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}]$. Using Lem. A.63 let us strip $\mathtt{L}_1$ from $F_{11}^{\vartheta}\langle F_2^{\vartheta'}[x\backslash \mathtt{vL}]\rangle$. This gives us four possibilities, **A**, **B**, **C**, and **D** in the statement of Lem. A.63.

3.1.1 Case A Then $F_{11}^{\vartheta} = F_{111}^{\hat{\vartheta}}\mathtt{L}_1$ and $\mathtt{C}\langle \mathtt{v}'\rangle = F_{111}^{\hat{\vartheta}}\langle\Delta\rangle$, where $\hat{\vartheta} = \mathsf{fz}^{\vartheta}(\mathtt{L}_1)$ and $F_{111}^{\hat{\vartheta}} \in \mathbb{X}^{\hat{\vartheta}}$. We consider three further subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{111}^{\hat{\vartheta}}$.

3.1.1.1 The hole of $\mathtt{C}$ and the hole of $F_{111}^{\hat{\vartheta}}$ are disjoint Then there is a two hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}'\rangle = F_{111}^{\hat{\vartheta}}$ and $\widehat{\mathtt{C}}\langle\Delta, \Box\rangle = \mathtt{C}$. By Lem. A.58 there are two possibilities for $\widehat{\mathtt{C}}$: the left and the right branch of the disjunction. The right branch, $\mathtt{C} \in \mathbb{X}^{\hat{\vartheta}}$, is impossible, since by strengthening ϑ (Tactic A.55) we have $\mathtt{C} \in \mathbb{X}^{\vartheta}$, which contradicts the fact that $\mathtt{r}$ is an internal step.

In the left branch case, $\widehat{\mathtt{C}}\langle\Box, z\rangle$ is an evaluation context in $\mathbb{X}^{\hat{\vartheta}}$. Since $\hat{\vartheta} = \mathsf{fz}^{\vartheta}([z\backslash \mathtt{v}']\mathtt{L}') \subseteq \vartheta \cup \mathsf{dom}\mathtt{L}'$, by applying the fact that non-structural variables are not required in "ϑ" (Lem. A.54) we obtain that $\widehat{\mathtt{C}}\langle\Box, z\rangle \in \mathbb{X}^{\vartheta}$. Then closing the diagram is straightforward.

3.1.1.2 The context $\mathtt{C}$ is a prefix of $F_{111}^{\hat{\vartheta}}$ By the decomposition of evaluation contexts lemma (Lem. A.20), we have that $\mathtt{C} \in \mathbb{X}^{\hat{\vartheta}}$. Since $\hat{\vartheta} \subseteq \vartheta \cup \mathsf{dom}\mathtt{L}'$, by applying the fact that non-structural variables are not required in "ϑ" (Lem. A.54) we obtain that $\mathtt{C} \in \mathbb{X}^{\vartheta}$. This contradicts the fact that $\mathtt{r}$ is an internal step.

3.1.1.3 The context $F_{111}^{\hat{\vartheta}}$ is a prefix of $\mathtt{C}$ Then $\mathtt{C} = F_{111}^{\hat{\vartheta}}\langle \mathtt{C}_1\rangle$, so $\mathtt{C}_1\langle \mathtt{v}'\rangle = F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}]$. We proceed by case analysis on the position of the hole of $\mathtt{C}_1$ in the term $F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}]$: it can be to the left of the substitution $[x\backslash \mathtt{vL}]$, or inside the substitution.

- Left of the substitution, $\mathtt{C}_1 = \mathtt{C}_{11}[x\backslash \mathtt{vL}]$ Now $\mathtt{C}_{11}\langle \mathtt{v}'\rangle = F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle$. Let us analyze the relative positions of the holes of the contexts $\mathtt{C}_{11}$ and $F_2^{\vartheta'}$. Observe that $F_2^{\vartheta'}$ cannot be a prefix of $\mathtt{C}_{11}$, as this would imply that $x = \mathtt{C}_2\langle \mathtt{v}\rangle$. So there are two possibilities, either the holes of $\mathtt{C}_{11}$ and $F_2^{\vartheta'}$ are disjoint, or $\mathtt{C}_{11}$ is a prefix of $F_2^{\vartheta'}$:
  - If the holes of $\mathtt{C}_{11}$ and $F_2^{\vartheta'}$ are disjoint Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}'\rangle = F_2^{\vartheta'}$ and $\widehat{\mathtt{C}}\langle x, \Box\rangle = \mathtt{C}_{11}$. By Lem. A.58 there are two possibilities for $\widehat{\mathtt{C}}$: the left and the right branch of the disjunction. The right branch case, $\mathtt{C}_{11} \in \mathbb{Y}^{\vartheta'}$, is impossible since then $\mathtt{C} = F_{111}^{\hat{\vartheta}}\langle \mathtt{C}_{11}[x\backslash \mathtt{vL}]\rangle[z\backslash \mathtt{v}'\mathtt{L}'] \in \mathbb{X}^{\vartheta}$, which contradicts the fact that $\mathtt{r}$ is an internal step. In the left branch case, $\widehat{\mathtt{C}}\langle\Box, z\rangle \in \mathbb{Y}^{\vartheta'}$, so closing the diagram is straightforward.
  - If $\mathtt{C}_{11}$ is a prefix of $F_2^{\vartheta'}$ Then $F_2^{\vartheta'} = \mathtt{C}_{11}\langle \mathtt{C}_2\rangle$. The situation is $t_0 = F_{111}^{\hat{\vartheta}}\langle \mathtt{C}_{11}\langle\!\langle z \rangle\!\rangle[x\backslash \mathtt{vL}]\rangle[z\backslash \mathtt{v}'\mathtt{L}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_{111}^{\hat{\vartheta}}\langle \mathtt{C}_{11}\langle \mathtt{v}'\rangle[x\backslash \mathtt{vL}]\rangle[z\backslash \mathtt{v}']\mathtt{L}' = t_1$ and we have that $F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle = \mathtt{C}_{11}\langle \mathtt{v}'\rangle$. Given that $\mathtt{C}_{11}$ is a prefix of $F_2^{\vartheta'}$, we have in particular that x occurs free in $\mathtt{v}'$. This is impossible

by Barendregt's variable convention, since $\mathtt{v}'$ is outside the scope of the substitution binding x in t_0.

- Inside the substitution, $\mathtt{C}_1 = F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{C}_{11}]$ So $\mathtt{C}_1\langle \mathtt{v}' \rangle = \mathtt{vL}$. We consider two further subcases, depending on whether the hole of $\mathtt{C}_1$ is inside $\mathtt{v}$ or inside one of the substitutions in $\mathtt{L}$.
  - If $\mathtt{C}_1 = \mathtt{C}_{111}\mathtt{L}$ and $\mathtt{v} = \mathtt{C}_{111}\langle \mathtt{v}' \rangle$ There are two possibilities, depending on whether the context $\mathtt{C}_{111}$ is empty. If $\mathtt{C}_{111}$ is empty, then the situation is $t_0 = F_{111}^{\hat{\vartheta}}\langle F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash z\mathtt{L}]\rangle[z\backslash \mathtt{v}'\mathtt{L}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_{111}^{\hat{\vartheta}}\langle F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{v}'\mathtt{L}]\rangle[z\backslash \mathtt{v}']\mathtt{L}' = t_1$. Note that the context $F_{111}^{\hat{\vartheta}}\langle F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash \Box\mathtt{L}]\rangle[z\backslash \mathtt{v}'\mathtt{L}']$ is a ϑ-evaluation context, so the step $\mathtt{r}$ is external, contradicting the hypothesis that it is internal.
    On the other hand, if $\mathtt{C}_{111}$ is non-empty, *i.e.* $\mathtt{C}_{111} = \lambda x'.\mathtt{C}_2$, then closing the diagram is straightforward.
  - If $\mathtt{C}_1 = \mathtt{vL}_1[y\backslash \mathtt{C}_{111}]\mathtt{L}_2$ and $\mathtt{L} = \mathtt{L}_1[y\backslash \mathtt{C}_{111}\langle \mathtt{v}' \rangle]\mathtt{L}_2$ Then closing the diagram is straightforward.

3.1.2 Case B Then $F_{11}^{\vartheta} = F_{111}^{\hat{\vartheta}}\langle\!\langle w \rangle\!\rangle\mathscr{L}\{\Box\}$, $\mathtt{C}\langle \mathtt{v}' \rangle = F_{111}^{\hat{\vartheta}}\langle\!\langle w \rangle\!\rangle$, and $\mathtt{L}_1 = \Box\mathscr{L}\{\Delta\}$, where $\hat{\vartheta} = \mathsf{fz}^{\vartheta}(\mathtt{L}_1)$, the evaluation context $F_{111}^{\hat{\vartheta}}$ is in $\mathbb{X}^{\hat{\vartheta}}$, and $\mathscr{L}$ is a (ϑ, w)-chain context. We consider three further subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{111}^{\hat{\vartheta}}$.

3.1.2.1 The hole of $\mathtt{C}$ and the hole of $F_{111}^{\hat{\vartheta}}$ are disjoint Then there is a two hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle \Box, \mathtt{v}' \rangle = F_{111}^{\hat{\vartheta}}$ and $\widehat{\mathtt{C}}\langle w, \Box \rangle = \mathtt{C}$. By Lem. A.58 there are two possibilities for $\widehat{\mathtt{C}}$: the left and the right branch of the disjunction. The right branch case, $\mathtt{C} \in \mathbb{X}^{\hat{\vartheta}}$, is impossible since by strengthening ϑ (Tactic A.55) we have $\mathtt{C}[z\backslash \mathtt{v}'\mathtt{L}'] \in \mathbb{X}^{\vartheta}$, which contradicts the fact that $\mathtt{r}$ is an internal step.
In the left branch case, $\widehat{\mathtt{C}}\langle \Box, z \rangle \in \mathbb{X}^{\hat{\vartheta}}$. Note that in the term t_1, the variable w is bound by $\Box\mathscr{L}\{\Delta\}[y\backslash t] = [z\backslash \mathtt{v}']\mathtt{L}'$ since $\mathscr{L}$ is a (ϑ, w)-chain context. Then w must also occur bound in the term $t_0 = \widehat{\mathtt{C}}\langle w, \mathtt{v}' \rangle[z\backslash \mathtt{v}'\mathtt{L}']$, since reduction cannot make a free variable become bound. Hence $w = z$. Consider the binding of w in the substitution context $\Box\mathscr{L}\{\Delta\}$. We know that it is of the form $\mathtt{I}^{\vartheta_1}\langle \Sigma \rangle$ where $\mathtt{I}^{\vartheta_1}$ is a inert evaluation context for some value of ϑ_1, and Σ is either Δ (if $\mathscr{L}$ has exactly one jump) or a variable (if $\mathscr{L}$ has more than one jump). So we have that $\mathtt{v}'\mathtt{L}' = \mathtt{I}^{\vartheta_1}\langle \Sigma \rangle$. This is impossible since answers do not have redexes or variables below inert evaluation contexts (Lem. A.21).

3.1.2.2 The context $\mathtt{C}$ is a prefix of $F_{111}^{\hat{\vartheta}}$ Then by Lem. A.20, $\mathtt{C} \in \mathbb{X}^{\hat{\vartheta}}$. By strengthening ϑ (Tactic A.55), $\mathtt{C}[z\backslash \mathtt{v}'\mathtt{L}'] \in \mathbb{X}^{\vartheta}$. This contradicts the fact that $\mathtt{r}$ is an internal step.

3.1.2.3 The context $F_{111}^{\hat{\vartheta}}$ is a prefix of $\mathtt{C}$ Then $\mathtt{C} = F_{111}^{\hat{\vartheta}}\langle \mathtt{C}_1 \rangle$, so $w = \mathtt{C}_1\langle\!\langle \mathtt{v}' \rangle\!\rangle$, which is impossible.

3.1.3 Case C Then F_{11}^{ϑ} is a substitution context, and: $F_2^{\vartheta'} = F_{21}^{\hat{\vartheta}'}\mathtt{L}_2$, $\mathtt{L}_1 = F_{11}^{\vartheta}\langle \mathtt{L}_2[x\backslash \mathtt{vL}] \rangle$, and $\mathtt{C}\langle \mathtt{v}' \rangle = F_{21}^{\hat{\vartheta}'}\langle\!\langle x \rangle\!\rangle$, where $\hat{\vartheta}' = \mathsf{fz}^{\vartheta'}(\mathtt{L}_2)$ and the evaluation context $F_{21}^{\hat{\vartheta}'}$ is

in $\mathbb{Y}^{\hat{\vartheta}'}$. The remainder of this case is similar to case 3.1.2, by case analysis on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{21}^{\hat{\vartheta}'}$.

3.1.4 Case D Then F_{11}^{ϑ} is a substitution context, $F_2^{\vartheta'} = F_{21}^{\hat{\vartheta}'}\langle\!\langle w \rangle\!\rangle \mathscr{L}\{\Box\}$, $\mathtt{L}_1 = F_{11}^{\vartheta}\langle \Box \mathscr{L}\{x\}[x\backslash \mathtt{vL}]\rangle$, and $\mathtt{C}\langle \mathtt{v}'\rangle = F_{21}^{\hat{\vartheta}'}\langle\!\langle w \rangle\!\rangle$, where $\hat{\vartheta}' = \mathtt{fz}^{\vartheta'}(\Box \mathscr{L}\{x\})$, the evaluation context $F_{21}^{\hat{\vartheta}'}$ is in $\mathbb{Y}^{\hat{\vartheta}'}$, and $\mathscr{L}$ is a (ϑ', w)-chain context. The remainder of this case is similar to case 3.1.2, by case analysis on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{21}^{\hat{\vartheta}'}$.

3.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'[y\backslash r_0]$ Then there is a step $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} F_{11}^{\vartheta}\langle\Delta\rangle$. Note that $\mathtt{r}_1$ must be ϑ-internal, for otherwise $\mathtt{r}$ would be ϑ-external. Then it is straightforward to conclude by *i.h.*.

3.3 The internal step $\mathtt{r}$ is to the right of $t_0 = t_0'[y\backslash r_0]$ Then $\mathtt{r} : r_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} r$ and closing the diagram is immediate.

4. ESubLStr, $F_1^{\vartheta} = F_{11}^{\vartheta\cup\{y\}}[y\backslash t]$ with $F_{11}^{\vartheta\cup\{y\}} \in \mathbb{X}^{\vartheta\cup\{y\}}$ and $t \in \mathsf{S}_{\vartheta}$ The situation is $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_{11}^{\vartheta\cup\{y\}}\langle\Delta\rangle[y\backslash t] = t_1 \overset{\vartheta}{\rightsquigarrow} F_{11}^{\vartheta\cup\{y\}}\langle\Delta'\rangle[y\backslash t] = t_3$. We consider three cases, depending on whether (1) the internal step $\mathtt{r}$ is at the root of t_0, (2) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to t_0', (3) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to r_0.

4.1 The internal step $\mathtt{r}$ is at the root of t_0 Note that $\mathtt{r}$ cannot be a $\mathtt{db}$ step, since it would be external. So it must be a $\mathtt{lsv}$ step of the form: $\mathtt{r} : t_0 = \mathtt{C}\langle\!\langle z \rangle\!\rangle[z\backslash \mathtt{v}'\mathtt{L}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle\!\langle \mathtt{v}' \rangle\!\rangle[z\backslash\mathtt{v}']\mathtt{L}' = t_1$. Let $\mathtt{L}_1$ be a substitution context such that $\mathtt{L}_1[y\backslash t] = [z\backslash\mathtt{v}']\mathtt{L}'$. Recall that $\Delta = F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash\mathtt{vL}]$. Using Lem. A.63 let us strip $\mathtt{L}_1$ from $F_{11}^{\vartheta\cup\{y\}}\langle F_2^{\vartheta'}[x\backslash\mathtt{vL}]\rangle$. This gives us four possibilities, **A**, **B**, **C**, and **D** in the statement of Lem. A.63.

4.1.1 Case A Then: $F_{11}^{\vartheta\cup\{y\}} = F_{111}^{\hat{\vartheta}\cup\{y\}}\mathtt{L}_1$ and $\mathtt{C}\langle\mathtt{v}'\rangle = F_{111}^{\hat{\vartheta}\cup\{y\}}\langle\Delta\rangle$, where $\hat{\vartheta} = \mathtt{fz}^{\vartheta\cup\{y\}}(\mathtt{L}_1)\backslash\{y\}$ and $F_{111}^{\hat{\vartheta}\cup\{y\}} \in \mathbb{X}^{\vartheta\cup\{y\}}$. We consider three cases, depending on whether the holes of $\mathtt{C}$ and $F_{111}^{\hat{\vartheta}\cup\{y\}}$ are disjoint, $\mathtt{C}$ is a prefix of $F_{111}^{\hat{\vartheta}\cup\{y\}}$, or $F_{111}^{\hat{\vartheta}\cup\{y\}}$ is a prefix of $\mathtt{C}$.

4.1.1.1 The hole of $\mathtt{C}$ and the hole of $F_{111}^{\hat{\vartheta}\cup\{y\}}$ are disjoint Then there is a two-hole context such that $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}'\rangle = F_{111}^{\hat{\vartheta}\cup\{y\}}$ and $\widehat{\mathtt{C}}\langle\Delta, \Box\rangle = \mathtt{C}$. By Lem. A.58 there are two possibilities for $\widehat{\mathtt{C}}$: the left and the right branch of the disjunction. The right branch case, $\mathtt{C} \in \mathbb{X}^{\hat{\vartheta}\cup\{y\}}$, is impossible, since by strengthening ϑ (Tactic A.55) we have $\mathtt{C}[z\backslash\mathtt{v}'\mathtt{L}'] \in \mathbb{X}^{\vartheta}$ which contradicts the hypothesis that $\mathtt{r}$ is an internal step.
In the left branch case, $\widehat{\mathtt{C}}\langle\Box, z\rangle \in \mathbb{X}^{\hat{\vartheta}\cup\{y\}}$. By strengthening ϑ (Tactic A.55), $\widehat{\mathtt{C}}\langle\Box, z\rangle[z\backslash\mathtt{v}'\mathtt{L}'] \in \mathbb{X}^{\vartheta}$ and closing the diagram is straightforward.

4.1.1.2 The context $\mathtt{C}$ is a prefix of $F_{111}^{\hat{\vartheta}\cup\{y\}}$ Then by the decomposition of evaluation contexts lemma (Lem. A.20) we know that $\mathtt{C} \in \mathbb{X}^{\hat{\vartheta}\cup\{y\}}$. By strengthening ϑ (Tactic A.55), $\mathtt{C}[z\backslash\mathtt{v}'\mathtt{L}'] \in \mathbb{X}^{\vartheta}$. This contradicts the hypothesis that $\mathtt{r}$ is an internal step.

4.1.1.3 The context $F_{111}^{\hat{\vartheta}\cup\{y\}}$ is a prefix of $\mathtt{C}$ Then $\mathtt{C} = F_{111}^{\hat{\vartheta}\cup\{y\}}\langle\mathtt{C}_1\rangle$. So $\Delta = \mathtt{C}_1\langle\mathtt{v}'\rangle$. Recall that $\Delta = F_2^{\vartheta'}\langle\!\langle x \rangle\!\rangle[x\backslash\mathtt{vL}]$. The remainder of this case is analogous

to case 3.1.1.3, by case analysis on whether the hole of $\mathtt{C}_1$ lies to the left or inside the substitution $[x\backslash \mathtt{vL}]$.

4.1.2 Case B Then $F_{11}^{\vartheta\cup\{y\}} = F_{111}^{\hat{\vartheta}\cup\{y\}}\langle\!\langle w\rangle\!\rangle\mathscr{L}\{\Box\}$, $\mathtt{C}\langle \mathtt{v}'\rangle = F_{111}^{\hat{\vartheta}\cup\{y\}}\langle\!\langle w\rangle\!\rangle$, and $\mathtt{L}_1 = \Box\mathscr{L}\{\Delta\}$, where $\hat{\vartheta} = \mathsf{fz}^{\vartheta\cup\{y\}}(\mathtt{L}_1)\backslash\{y\}$, the evaluation context $F_{111}^{\hat{\vartheta}\cup\{y\}}$ is in $\mathbb{X}^{\hat{\vartheta}\cup\{y\}}$, and $\mathscr{L}$ is a $(\vartheta\cup\{y\}, w)$-chain context.
We consider three further subcases, depending on the position of the hole of $\mathtt{C}$ relative to the position of the hole of $F_{111}^{\hat{\vartheta}\cup\{y\}}$.

4.1.2.1 The hole of $\mathtt{C}$ and the hole of $F_{111}^{\hat{\vartheta}\cup\{y\}}$ are disjoint Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}'\rangle = F_{111}^{\hat{\vartheta}\cup\{y\}}$ and $\widehat{\mathtt{C}}\langle w, \Box\rangle = \mathtt{C}$. By Lem. A.58 there are two possibilities for $\widehat{\mathtt{C}}$: the left and the right branch of the disjunction. The right branch case, $\mathtt{C} \in \mathbb{X}^{\hat{\vartheta}\cup\{y\}}$, is impossible, since by strengthening ϑ (Tactic A.55), we have that $\widehat{\mathtt{C}}\langle\Box, z\rangle[z\backslash \mathtt{v}'\mathtt{L}'] \in \mathbb{X}^{\vartheta}$, which contradicts the fact that the step $\mathtt{r}$ is internal.
In the left branch case, $\widehat{\mathtt{C}}\langle\Box, z\rangle \in \mathbb{X}^{\hat{\vartheta}\cup\{y\}}$. By strengthening ϑ (Tactic A.55), $\widehat{\mathtt{C}}\langle\Box, z\rangle[z\backslash \mathtt{v}'\mathtt{L}'] \in \mathbb{X}^{\vartheta}$. Then closing the diagram is immediate.

4.1.2.2 The context $\mathtt{C}$ is a prefix of $F_{111}^{\hat{\vartheta}\cup\{y\}}$ By the decomposition lemma for evaluation contexts (Lem. A.20) we know that $\mathtt{C} \in \mathbb{X}^{\hat{\vartheta}\cup\{y\}}$. By strengthening ϑ (Tactic A.55), $\widehat{\mathtt{C}}\langle\Box, z\rangle[z\backslash \mathtt{v}'\mathtt{L}'] \in \mathbb{X}^{\vartheta}$. This contradicts the fact that the step $\mathtt{r}$ is internal.

4.1.2.3 The context $F_{111}^{\hat{\vartheta}\cup\{y\}}$ is a prefix of $\mathtt{C}$ Then $\mathtt{C} = F_{111}^{\hat{\vartheta}\cup\{y\}}\langle \mathtt{C}_1\rangle$. Hence $w = \mathtt{C}_1\langle \mathtt{v}\rangle$, which is impossible.

4.1.3 Case C Then F_{11}^{ϑ} is a substitution context, and: $F_2^{\vartheta'} = F_{21}^{\hat{\vartheta}'}\mathtt{L}_2, \mathtt{L}_1 = F_{11}^{\vartheta}\langle \mathtt{L}_2[x\backslash \mathtt{vL}]\rangle$, and $\mathtt{C}\langle \mathtt{v}'\rangle = F_{21}^{\vartheta'}\langle\!\langle x\rangle\!\rangle$, where $\hat{\vartheta}' = \mathsf{fz}^{\vartheta'}(\mathtt{L}_2)$ and the evaluation context $F_{21}^{\hat{\vartheta}'}$ is in $\mathbb{Y}^{\hat{\vartheta}'}$. The remainder of this case is analogous to case 3.1.3, by case analysis on the relative positions of the holes of $\mathtt{C}$ and $F_{21}^{\hat{\vartheta}'}$.

4.1.4 Case D Then F_{11}^{ϑ} is a substitution context, and: $F_2^{\vartheta'} = F_{21}^{\hat{\vartheta}'}\langle\!\langle w\rangle\!\rangle\mathscr{L}\{\Box\}$, $\mathtt{L}_1 = F_{11}^{\vartheta\cup\{y\}}\langle\Box\mathscr{L}\{x\}[x\backslash \mathtt{vL}]\rangle$, and $\mathtt{C}\langle \mathtt{v}'\rangle = F_{21}^{\hat{\vartheta}'}\langle\!\langle w\rangle\!\rangle$, where $\hat{\vartheta}' = \mathsf{fz}^{\vartheta\cup\{y\}}(\mathtt{L}_1)$, the evaluation context $F_{21}^{\hat{\vartheta}'}$ is in $\mathbb{Y}^{\hat{\vartheta}'}$, and $\mathscr{L}$ is a (ϑ', w)-chain context. The remainder of this case is analogous to case 3.1.4, by case analysis on the relative positions of the holes of $\mathtt{C}$ and $F_{21}^{\hat{\vartheta}'}$.

4.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'[y\backslash r_0]$ Then there is a step $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} F_{11}^{\vartheta\cup\{y\}}\langle\Delta\rangle[y\backslash t]$. It must be a $(\vartheta\cup\{y\})$-internal step, for otherwise $\mathtt{r}$ would be ϑ-external. Then it is straightforward to conclude by *i.h.*.

4.3 The internal step $\mathtt{r}$ is to the right of $t_0 = t_0'[y\backslash r_0]$ Then the internal step $\mathtt{r}$ is of the form: $F_1^{\vartheta\cup\{y\}}\langle\Delta\rangle[y\backslash r_0] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_1^{\vartheta\cup\{y\}}\langle\Delta\rangle[y\backslash t]$ and there is a step $\mathtt{r}_1 : r_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} t$. We consider two cases, depending on whether y is a structural variable in $F_1^{\vartheta\cup\{y\}}$.

4.3.1 If $y \in \mathsf{sv}(F_1^{\vartheta\cup\{y\}})$ Then since structural variables are below evaluation contexts (Lem. A.54) there is a context $\widetilde{F}_1^{\vartheta} \in \mathbb{X}^{\vartheta}$ such that $F_1^{\vartheta\cup\{y\}}\langle\Delta\rangle = \widetilde{F}_1^{\vartheta}\langle\!\langle y\rangle\!\rangle$. Consider two further subcases, depending on whether $\mathtt{r}_1$ is ϑ-external or ϑ-internal:

4.3.1.1 If $\mathtt{r}_1$ is a ϑ-external step Then $\mathtt{r}$ is ϑ-external, contradicting the hypothesis that it is ϑ-internal.

4.3.1.2 If $\mathtt{r}_1$ is a ϑ-internal step Then since normal forms are backward preserved by internal steps (Lem. A.69), r_0 is a structure in S_ϑ, so $F_1^{\vartheta\cup\{y\}}[y\backslash r_0]$ is an evaluation context in $\mathbb{X}^\vartheta$ and closing the diagram is straightforward.

4.3.2 If $y \notin \mathsf{sv}(F_1^{\vartheta\cup\{y\}})$ Then since non-structural variables are not required in "ϑ" (Lem. A.54), $F^{\vartheta\cup\{y\}}$ is an evaluation context in $\mathbb{X}^\vartheta$. Regardless of whether r_0 is a structure or not a structure, the context $F_1^{\vartheta\cup\{y\}}[x\backslash r_0]$ is an evaluation context in $\mathbb{X}^\vartheta$. Then closing the diagram is straightforward.

5. ESUBSR, $F_1^\vartheta = F_{11}^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{I}^\vartheta]$ with $F_{11}^\vartheta \in \mathbb{X}^\vartheta$ and $\mathtt{I}^\vartheta \in \mathsf{E}_\vartheta^\circ$ The situation is $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} F_{11}^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{I}^\vartheta\langle\Delta\rangle] = t_1 \overset{\vartheta}{\rightsquigarrow} F_{11}^\vartheta\langle\!\langle y\rangle\!\rangle[y\backslash \mathtt{I}^\vartheta\langle\Delta'\rangle] = t_3$. We consider three cases, depending on whether (1) the internal step $\mathtt{r}$ is at the root of t_0, (2) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to t_0', (3) t_0 is a substitution $t_0'[y\backslash r_0]$ and the step $\mathtt{r}$ is internal to r_0.

5.1 The internal step $\mathtt{r}$ is at the root of t_0 Note that $\mathtt{r}$ cannot be a $\mathtt{db}$ step, since it would be external, so it must be a $\mathtt{lsv}$ step of the form $\mathtt{r} : t_0 = \mathtt{C}\langle\!\langle z\rangle\!\rangle[z\backslash \mathtt{v}'\mathtt{L}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{C}\langle \mathtt{v}'\rangle[z\backslash \mathtt{v}']\mathtt{L}' = t_1$. Let $\mathtt{L}_1$ be a substitution context such that $\mathtt{L}_1[y\backslash \mathtt{I}^\vartheta\langle\Delta\rangle] = [z\backslash \mathtt{v}']\mathtt{L}'$. Using Lem. A.62 let us strip $\mathtt{L}_1$ from $F_{11}^\vartheta[y\backslash \mathtt{I}^\vartheta\langle\Delta\rangle]$. This gives us two possibilities, **A** and **B** in the statement of Lem. A.62.

5.1.1 Case A Then $F_{11}^\vartheta = F_{111}^{\hat\vartheta}\mathtt{L}_1$ and $\mathtt{C}\langle \mathtt{v}'\rangle = F_{111}^{\hat\vartheta}\langle\!\langle y\rangle\!\rangle$, where $\hat\vartheta = \mathsf{fz}^\vartheta(\mathtt{L}_1)$ and $F_{111}^{\hat\vartheta} \in \mathbb{X}^{\hat\vartheta}$. We consider three cases, depending on whether the holes of $\mathtt{C}$ and $F_{111}^{\hat\vartheta}$ are disjoint, $\mathtt{C}$ is a prefix of $F_{111}^{\hat\vartheta}$, or $F_{111}^{\hat\vartheta}$ is a prefix of $\mathtt{C}$.

5.1.1.1 The hole of $\mathtt{C}$ and the hole of $F_{111}^{\hat\vartheta}$ are disjoint Then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box, \mathtt{v}'\rangle = F_{111}^{\hat\vartheta}$ and $\widehat{\mathtt{C}}\langle y, \Box\rangle = \mathtt{C}$. Note that the internal step $\mathtt{r}$ is of the form: $\mathtt{r} : \widehat{\mathtt{C}}\langle y, z\rangle[z\backslash \mathtt{v}'\mathtt{L}'] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \widehat{\mathtt{C}}\langle y, \mathtt{v}'\rangle[z\backslash \mathtt{v}']\mathtt{L}'$ and y is bound by $\mathtt{L}_1[y\backslash \mathtt{I}^\vartheta\langle\Delta\rangle] = [z\backslash \mathtt{v}']\mathtt{L}'$ on the right-hand side, so it must be the case that $y = z$, for otherwise y would be free on the left-hand side, and free variables cannot become bound. Therefore, since $y = z$, we have that $\mathtt{v}' = \mathtt{I}^\vartheta\langle\Delta\rangle$. This is impossible, since answers do not have redexes below inert evaluation contexts (Lem. A.21).

5.1.1.2 The context $\mathtt{C}$ is a prefix of $F_{111}^{\hat\vartheta}$ By the decomposition of evaluation contexts lemma (Lem. A.20) we know that $\mathtt{C} \in \mathbb{X}^{\hat\vartheta}$. By strengthening ϑ (Tactic A.55), $\mathtt{C}[z\backslash \mathtt{v}'\mathtt{L}'] \in \mathbb{X}^\vartheta$. This contradicts the fact that $\mathtt{r}$ is a ϑ-internal step.

5.1.1.3 The context of $F_{111}^{\hat\vartheta}$ is a prefix of $\mathtt{C}$ Then $\mathtt{C} = F_{111}^{\hat\vartheta}\langle \mathtt{C}_1\rangle$, so $y = \mathtt{C}_1\langle \mathtt{v}'\rangle$ which is impossible.

5.1.2 Case B Then $F_{11}^\vartheta = F_{111}^{\hat\vartheta}\langle\!\langle w\rangle\!\rangle\mathscr{L}\{\Box\}$, $\mathtt{C}\langle \mathtt{v}'\rangle = F_{111}^{\hat\vartheta}\langle\!\langle w\rangle\!\rangle$, and $\mathtt{L}_1 = \Box\mathscr{L}\{y\}$, where $\hat\vartheta = \mathsf{fz}^\vartheta(\mathtt{L}_1)$, the evaluation context $F_{111}^{\hat\vartheta}$ is in $\mathbb{X}^{\hat\vartheta}$, and $\mathscr{L}$ is a (ϑ, w)-chain context. The remainder of this case is similar to case 5.1.1, by case analysis on the relative positions of the holes of $\mathtt{C}$ and $F_{111}^{\hat\vartheta}$.

5.2 The internal step $\mathtt{r}$ is to the left of $t_0 = t_0'[y\backslash r_0]$ Let $\mathtt{r}_1 : t_0' \to_{\mathtt{sh}\backslash\mathtt{gc}} F_{11}^{\vartheta}\langle\!\langle y \rangle\!\rangle$ be the step isomorphic to $\mathtt{r}$ but going under the substitution $[y\backslash \mathtt{I}^{\vartheta}\langle\Delta\rangle]$. Note that $\mathtt{r}_1$ cannot be ϑ-external since, by Lem. A.26, this would imply that $\mathtt{r}$ is also ϑ-external. So $\mathtt{r}_1$ is ϑ-internal and we may apply the fact that evaluation contexts are backward preserved by internal steps (Lem. A.70) to conclude that t_0' has to be of the form $F_{110}^{\vartheta}\langle\!\langle y \rangle\!\rangle$. Then closing the diagram is straightforward.

5.3 The internal step $\mathtt{r}$ is to the right of $t_0 = t_0'[y\backslash r_0]$ Let $\mathtt{r}_1 : r_0 \to_{\mathtt{sh}\backslash\mathtt{gc}} \mathtt{I}^{\vartheta}\langle\Delta\rangle$ be the step isomorphic to $\mathtt{r}$ but going inside the substitution $F_{11}^{\vartheta}\langle\!\langle y \rangle\!\rangle[y\backslash\Box]$. Note that $\mathtt{r}_1$ cannot be ϑ-external since this would imply that $\mathtt{r}$ is ϑ-external. Then it is immediate to conclude by *i.h.*.

6. EAppRStr, $F_1^{\vartheta} = M^{\vartheta} F_{11}^{\vartheta}$, where $M^{\vartheta} \in \mathsf{S}_{\vartheta}$ and $F_{11}^{\vartheta} \in \mathsf{E}_{\vartheta}$ The situation is $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} M^{\vartheta} F_{11}^{\vartheta}\langle\Delta\rangle = t_1 \overset{\vartheta}{\rightsquigarrow} M^{\vartheta} F_{11}^{\vartheta}\langle\Delta'\rangle = t_3$. Note that the internal step $\mathtt{r}$ cannot be at the root: it cannot be a $\mathtt{db}$ step, since it would be external, and it cannot be a $\mathtt{lsv}$ step, since then there would be a substitution node at the root of t_1. So t_0 must be an application node $r_1\, r_2$ and there are two remaining cases: (1) the step $\mathtt{r}$ is internal to r_1, (2) the step $\mathtt{r}$ is internal to r_2.

6.1 The internal step $\mathtt{r}$ is internal to the left of $t_0 = r_1\, r_2$ Then $t_0 = r_1\, F_{11}^{\vartheta}\langle\Delta\rangle$. Let $\mathtt{r}_1 : r_1 \to_{\mathtt{sh}\backslash\mathtt{gc}} M^{\vartheta}$ be the step isomorphic to $\mathtt{r}$ below the context $\Box\, F_{11}^{\vartheta}\langle\Delta\rangle$. Note that $\mathtt{r}_1$ cannot be ϑ-external as this would imply that $\mathtt{r}$ is also ϑ-external. By the fact that strong normal forms are backward stable by internal steps (Lem. A.69), r_1 must be a strong ϑ-structure. Then closing the diagram is straightforward.

6.2 The internal step $\mathtt{r}$ is internal to the right of $t_0 = r_1\, r_2$ Then $t_0 = M^{\vartheta}\, r_2$. Let $\mathtt{r}_1 : r_2 \to_{\mathtt{sh}\backslash\mathtt{gc}} F_{11}^{\vartheta}\langle\Delta\rangle$ be the step isomorphic to $\mathtt{r}$ below the context $M^{\vartheta}\,\Box$. Note that $\mathtt{r}_1$ cannot ϑ-external since this would imply that $\mathtt{r}$ is also ϑ-external. Then it is immediate to conclude by *i.h.*.

7. ELam, $F_1^{\vartheta} = \lambda y.F^{\vartheta\cup\{y\}}$, where $F^{\vartheta} \in \mathsf{E}_{\vartheta}$ Straightforward by *i.h.*.

□

Lemma A.73 (Full proof of Lem. 4.50—Postponement of internal steps)**.** *Let ϑ be such that $\mathtt{fv}(t_0) \subseteq \vartheta$. If $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_1 \overset{\vartheta}{\rightsquigarrow} t_3$ there exists a term t_2 such that $t_0 \overset{\vartheta}{\rightsquigarrow} t_2 \twoheadrightarrow_{\mathtt{sh}\backslash\mathtt{gc}} t_3$. Furthermore, $(\xrightarrow{\neg\vartheta}_{\mathtt{shdb}}, \overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}}, \xrightarrow{\neg\vartheta}_{\mathtt{shlsv}}, \overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}})$ forms a* square factorization system *according to the terminology of [3], taking $\overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}}$ (resp. $\overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}}$) to be the external* $\mathtt{db}$ *(resp.* $\mathtt{lsv}$*) reduction, and $\xrightarrow{\neg\vartheta}_{\mathtt{shdb}}$ (resp. $\xrightarrow{\neg\vartheta}_{\mathtt{shlsv}}$) to be the internal* $\mathtt{db}$ *(resp.* $\mathtt{lsv}$*) reduction. More precisely, only the following swaps are allowed:*

$$
\begin{array}{rcl}
\xrightarrow{\neg\vartheta}_{\mathtt{shdb}} \overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}} & \subseteq & (\overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}})^{+} (\xrightarrow{\neg\vartheta}_{\mathtt{shdb}})^{*} \\
\xrightarrow{\neg\vartheta}_{\mathtt{shlsv}} \overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}} & \subseteq & (\overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}})^{+} (\xrightarrow{\neg\vartheta}_{\mathtt{shlsv}})^{*} \\
\xrightarrow{\neg\vartheta}_{\mathtt{shdb}} \overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}} & \subseteq & \overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}} (\xrightarrow{\neg\vartheta}_{\mathtt{sh}})^{*} \\
\xrightarrow{\neg\vartheta}_{\mathtt{shlsv}} \overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}} & \subseteq & \overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}} (\xrightarrow{\neg\vartheta}_{\mathtt{sh}})^{*}
\end{array}
$$

Proof. Let $\mathtt{r}$ be the internal step $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_1$ and $\mathtt{r}'$ the external step $t_1 \overset{\vartheta}{\rightsquigarrow} t_3$. The proof goes by case analysis on the kind of step $\mathtt{r}'$. If $\mathtt{r}'$ is a $\mathtt{db}$ step, this is a consequence of Lem. A.71. If $\mathtt{r}'$ is a $\mathtt{lsv}$ step, this is a consequence of Lem. A.72. Note that in both cases the construction is given inductively. In all the base cases, the diagram is closed according to the allowed swaps. In all the inductive cases, the diagram is closed using the same kind of swaps as in the inductive hypothesis. □

A.3 Proofs of Chapter 6 – A Labeled Linear Substitution Calculus

A.3.1 Redex creation — proof of Prop. 6.4

Definition A.74 (Ancestor of a variable)**.** Let $\mathtt{C}_1\langle\!\langle x \rangle\!\rangle \xrightarrow{R} \mathtt{C}_2\langle\!\langle x \rangle\!\rangle$ be a step in the LSC. Consider the term that results from marking the occurrence of x under the context $\mathtt{C}_1$, *i.e.* $\mathtt{C}_1\langle\!\langle x^{\mathfrak{a}} \rangle\!\rangle$, and let R' be the step in the LSC with marks corresponding to R via the obvious bijection, *i.e.* $R' : \mathtt{C}_1\langle\!\langle x^{\mathfrak{a}} \rangle\!\rangle \xrightarrow{\mathfrak{a}} t$. Then the occurrence of x under $\mathtt{C}_1$ is an *ancestor* of the occurrence of x under $\mathtt{C}_2$ before the step R if $t = \mathtt{C}_2'\langle\!\langle x^{\mathfrak{a}} \rangle\!\rangle$ where the context $\mathtt{C}_2'$ is a variant of the context $\mathtt{C}_2$ (with possibly some other marks).

It is a well-known fact that in the λ-calculus free variables cannot be created. That is, if $t \to s$ is a step, then $\mathtt{fv}(t) \supseteq \mathtt{fv}(s)$. The same property also holds in the LSC. Actually, both in the λ-calculus and in the LSC, a stronger property holds: for any step $t \to s$, every occurrence of a free variable x in s has an ancestor in t.

Lemma A.75 (Every variable occurrence has an ancestor)**.** *Let $R : t_1 \to t_2 = \mathtt{C}\langle\!\langle x \rangle\!\rangle$ be a step in the LSC. Then x has an ancestor,* i.e. *there exists a context $\mathtt{C}_0$ such that $t_1 = \mathtt{C}_0\langle\!\langle x \rangle\!\rangle$ and such that the occurrence of x under $\mathtt{C}_0$ is an ancestor of the occurrence of x under $\mathtt{C}$ before the step R.*

Proof. This property can be checked by a straightforward case analysis on the kind of redex R ($\mathtt{db}$, $\mathtt{ls}$, or $\mathtt{gc}$). If R is a $\mathtt{db}$ redex, the step is of the form:

$$t_1 = (\lambda y.t)\mathtt{L}s \xrightarrow{R} t[y\backslash s]\mathtt{L} = t_2$$

Consider an occurrence of a variable x on the term t_2, under a context $\mathtt{C}$, *i.e.* $t = \mathtt{C}\langle\!\langle x \rangle\!\rangle$. Then there are three possibilities:

1. **The variable occurrence is inside t.** That is, $\mathtt{C} = \mathtt{C}'[x\backslash s]\mathtt{L}$. Then taking $\mathtt{C}_0 := (\lambda x.\mathtt{C}')\mathtt{L}\, s$, the occurrence of x in t_1 under $\mathtt{C}_0$ is an ancestor of the occurrence of x in t_2 under $\mathtt{C}$.

2. **The variable occurrence is inside s.** Similar to the previous case. More precisely, we have that $\mathtt{C} = t[x\backslash \mathtt{C}']\mathtt{L}$, and we take $\mathtt{C}_0 := (\lambda x.t)\mathtt{L}\, \mathtt{C}'$.

3. **The variable occurrence is inside one of the substitutions in $\mathtt{L}$.** Similar to the previous cases. More precisely, we have that there exist substitution contexts $\mathtt{L}_1$ and

$\mathtt{L}_2$ such that $\mathtt{L} = \mathtt{L}_1[z\backslash \mathtt{C}'\langle\langle x \rangle\rangle]\mathtt{L}_2$ and $\mathtt{C} = t[x\backslash s]\mathtt{L}_1[z\backslash \mathtt{C}']\mathtt{L}_2$. Then we may conclude by taking $\mathtt{C}_0 := (\lambda x.t)\mathtt{L}_1[z\backslash \mathtt{C}']\mathtt{L}_2\, s$.

The proofs for the `ls` and `gc` cases are similar. □

Lemma A.76 (Creation of an answer)**.** *Recall that an answer is a term of the form* $(\lambda x.t)\mathtt{L}$*. Suppose that* $\mathtt{C}\langle\langle x \rangle\rangle$ *is not an answer and* $\mathtt{C}\langle t \rangle$ *is an answer. Then* $\mathtt{C}$ *is a substitution context and* t *is an answer.*

Proof. Straightforward by induction on $\mathtt{C}$. □

Proposition A.77 (Full proof of Prop. 6.4—Redex creation in the LSC)**.** *Let* $t_1 \xrightarrow{R} t_2 \xrightarrow{S} t_3$ *be a sequence of two redexes in the LSC such that* R *creates* S*. Then* S *is created in exactly one of the following possible ways. The anchors of the redexes* R *and* S *are underlined in each case for clarity.*

1. **Creation case 1:** `db` **creates** `db`**.**

$$t_1 = \mathtt{C}\langle((\lambda \underline{x}.(\lambda y.t)\mathtt{L}_1)\mathtt{L}_2\, s)\mathtt{L}_3\, u\rangle \xrightarrow{R} \mathtt{C}\langle(\lambda \underline{y}.t)\mathtt{L}_1[x\backslash s]\mathtt{L}_2\mathtt{L}_3\, u\rangle = t_2$$

2. **Creation case 2:** `db` **creates** `ls`**.**

$$t_1 = \mathtt{C}_1\langle(\lambda \underline{x}.\mathtt{C}_2\langle\langle x \rangle\rangle)\mathtt{L}\, t\rangle \xrightarrow{R} \mathtt{C}_1\langle\mathtt{C}_2\langle\langle \underline{x} \rangle\rangle[x\backslash t]\mathtt{L}\rangle = t_2$$

3. **Creation case 3:** `db` **creates** `gc`**.** *For* $x \notin \mathtt{fv}(t)$*:*

$$t_1 = \mathtt{C}_1\langle(\lambda \underline{x}.t)\mathtt{L}\, s\rangle \xrightarrow{R} \mathtt{C}_1\langle t[\underline{x}\backslash s]\mathtt{L}\rangle = t_2$$

4. **Creation case 4:** `ls` **creates** `db` **upwards.**

$$t_1 = \mathtt{C}\langle \underline{x}\mathtt{L}_1[x\backslash(\lambda y.t)\mathtt{L}_2]\mathtt{L}_3\, s\rangle \xrightarrow{R} \mathtt{C}\langle(\lambda \underline{y}.t)\mathtt{L}_2\mathtt{L}_1[x\backslash(\lambda y.t)\mathtt{L}_2]\mathtt{L}_3\, s\rangle = t_2$$

5. **Creation case 5:** `ls` **creates** `db` **downwards.**

$$t_1 = \mathtt{C}_1\langle\mathtt{C}_2\langle \underline{x}\mathtt{L}_1\, t\rangle[x\backslash(\lambda y.s)\mathtt{L}_2]\rangle \xrightarrow{R} \mathtt{C}_1\langle\mathtt{C}_2\langle(\lambda \underline{y}.s)\mathtt{L}_2\mathtt{L}_1\, t\rangle[x\backslash(\lambda y.s)\mathtt{L}_2]\rangle = t_2$$

6. **Creation case 6:** `ls` **creates** `gc`**.** *For* $x \notin \mathtt{C}_2\langle t \rangle$*:*

$$t_1 = \mathtt{C}_1\langle\mathtt{C}_2\langle\langle \underline{x} \rangle\rangle[x\backslash t]\rangle \xrightarrow{R} \mathtt{C}_1\langle\mathtt{C}_2\langle t\rangle[\underline{x}\backslash t]\rangle = t_2$$

7. **Creation case 7:** `gc` **creates** `gc`**.** *For* $y \in \mathtt{fv}(s)$ *and* $y \notin \mathtt{fv}(\mathtt{C}_2\langle t \rangle)$*:*

$$t_1 = \mathtt{C}_1\langle\mathtt{C}_2\langle t[\underline{x}\backslash s]\rangle[y\backslash u]\rangle \xrightarrow{R} \mathtt{C}_1\langle\mathtt{C}_2\langle t\rangle[\underline{y}\backslash u]\rangle = t_2$$

Proof. Throughout the proof, we let Δ stand for the pattern of S and Δ' for its contractum. Similarly, Σ stands for the pattern of S and Σ' for its contractum. By case analysis on the kind of redex R*:*

1. **If R is a $\mathtt{db}$ redex.** Then $t_1 = \mathtt{C}\langle(\lambda x.t)\mathtt{L}\, s\rangle \xrightarrow{R} \mathtt{C}\langle t[x\backslash s]\mathtt{L}\rangle = t_2$. Consider the position of the hole of $\mathtt{C}$, relative to the position the pattern Σ of S.

   First, if the position of the hole of $\mathtt{C}$ is a prefix of the position of Σ, then there are two subcases, depending on whether Σ overlaps the spine of $t[x\backslash s]\mathtt{L}$ or it does not overlap the spine:

   1.1 **If Σ overlaps the spine of $t[x\backslash s]\mathtt{L}$.** Then the redex S must be either a $\mathtt{ls}$ redex contracting one of the substitutions among $[x\backslash s]\mathtt{L}$, or a $\mathtt{gc}$ redex collecting one of the substitutions among $[x\backslash s]\mathtt{L}$. Let us call the *affected substitution* the one that is either contracted by a $\mathtt{ls}$ step or collected by a $\mathtt{gc}$ step.

   If the substitution that is being affected is one of the substitutions in $\mathtt{L}$ then it is immediate to observe that this case is impossible, as S has an ancestor before R. For example, if S is a $\mathtt{gc}$ step, then $\mathtt{L} = \mathtt{L}_1[y\backslash r]\mathtt{L}_2$ and the situation is:

$$\begin{array}{ccc} \mathtt{C}\langle(\lambda x.t)\mathtt{L}_1[y\backslash r]\mathtt{L}_2\, s\rangle & \xrightarrow{R} & \mathtt{C}\langle t[x\backslash s]\mathtt{L}_1[y\backslash r]\mathtt{L}_2\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}\langle(\lambda x.t)\mathtt{L}_1\mathtt{L}_2\, s\rangle & & \mathtt{C}\langle t[x\backslash s]\mathtt{L}_1\mathtt{L}_2\rangle \end{array}$$

   observe that $y \notin \mathtt{fv}(t[x\backslash s]\mathtt{L}_1)$ implies $y \notin \mathtt{fv}((\lambda x.t)\mathtt{L}_1)$.

   If the substitution that is being affected is $[x\backslash s]$ then the step S is created by R, and we are either in **Creation case 2: $\mathtt{db}$ creates $\mathtt{ls}$** or in **Creation case 3: $\mathtt{db}$ creates $\mathtt{gc}$**. For example, if S is a $\mathtt{gc}$ step:

$$t_1 = \mathtt{C}\langle(\lambda x.t)\mathtt{L}\, s\rangle \xrightarrow{R} \mathtt{C}\langle t[x\backslash s]\mathtt{L}\rangle = t_2$$

   1.2 **If Σ does not overlap the spine of $t[x\backslash s]\mathtt{L}$.** Then it may be the case that Σ lies inside t, or inside s, or inside one of the subsitutions of $\mathtt{L}$. In all of these cases, S has an ancestor and the situation is impossible. For example, if Σ lies inside t then t is of the form $t = \mathtt{C}'\langle\Sigma\rangle$, and the situation is:

$$\begin{array}{ccc} \mathtt{C}\langle(\lambda x.\mathtt{C}'\langle\Sigma\rangle)\mathtt{L}\, s\rangle & \xrightarrow{R} & \mathtt{C}\langle\mathtt{C}'\langle\Sigma\rangle[x\backslash s]\mathtt{L}\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}\langle(\lambda x.\mathtt{C}'\langle\Sigma'\rangle)\mathtt{L}\, s\rangle & & \mathtt{C}\langle\mathtt{C}'\langle\Sigma'\rangle[x\backslash s]\mathtt{L}\rangle \end{array}$$

   Second, if the positions of the hole of $\mathtt{C}$ and Σ are disjoint, then S has an ancestor and this case is impossible. More precisely, there must exist a two-hole context $\widehat{\mathtt{C}}$ such that $\mathtt{C} = \widehat{\mathtt{C}}\langle\Box, \Sigma\rangle$, and the situation is:

$$\begin{array}{ccc} \widehat{\mathtt{C}}\langle(\lambda x.t)\mathtt{L}s, \Sigma\rangle & \xrightarrow{R} & \widehat{\mathtt{C}}\langle t[x\backslash s], \Sigma\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \widehat{\mathtt{C}}\langle(\lambda x.t)\mathtt{L}s, \Sigma'\rangle & & \widehat{\mathtt{C}}\langle t[x\backslash s], \Sigma'\rangle \end{array}$$

   Finally we arrive to the more complex case, when the position of the pattern of Σ is a prefix of the position of the hole of $\mathtt{C}$, that is, there exist contexts $\mathtt{C}_1$ and $\mathtt{C}_2$ such that $t_2 = \mathtt{C}_1\langle\Sigma\rangle$ and $\mathtt{C} = \mathtt{C}_1\langle\mathtt{C}_2\rangle$. Note that $\Sigma = \mathtt{C}_2\langle t[x\backslash s]\mathtt{L}\rangle$. We proceed by case analysis on the kind of redex S:

1.1 **If S is a $\mathtt{db}$ redex.** Then $\Sigma = (\lambda y.u)\mathtt{L}'\,r = \mathtt{C}_2\langle t[x\backslash s]\mathtt{L}\rangle$. If the hole of $\mathtt{C}_2$ is inside u, inside r, or inside one of the substitutions of $\mathtt{L}'$, then S has an ancestor S_0. For example if the hole of $\mathtt{C}_2$ is inside u then $\mathtt{C}_2 = (\lambda y.\mathtt{C}_3)\mathtt{L}'\,r$, and the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle(\lambda y.\mathtt{C}_3\langle\Delta\rangle)\mathtt{L}'\,r\rangle & \xrightarrow{R} & \mathtt{C}_1\langle(\lambda y.\mathtt{C}_3\langle\Delta'\rangle)\mathtt{L}'\,r\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\mathtt{C}_3\langle\Delta\rangle[y\backslash r]\mathtt{L}'\rangle & & \mathtt{C}_1\langle\mathtt{C}_3\langle\Delta'\rangle[y\backslash r]\mathtt{L}'\rangle \end{array}$$

The only remaining possibility is that there exist substitution contexts $\mathtt{L}_1$ and $\mathtt{L}_2$ such that $\mathtt{L}' = \mathtt{L}_1\mathtt{L}_2$ and $\mathtt{C}_2 = \mathtt{L}_2\,r$. The situation is:

$$t_1 = \mathtt{C}_1\langle((\lambda x.t)\mathtt{L}\,s)\,\mathtt{L}_2\,r\rangle \xrightarrow{R} \mathtt{C}_1\langle t[x\backslash s]\mathtt{L}\,\mathtt{L}_2\,r\rangle = t_2$$

so t is of the form $t = (\lambda y.r)\mathtt{L}_3$. Hence we are in **Creation case 1:** $\mathtt{db}$ **creates** $\mathtt{db}$.

1.2 **If S is a $\mathtt{ls}$ redex.** Then $\Sigma = \mathtt{C}_3\langle\!\langle y\rangle\!\rangle[y\backslash r] = \mathtt{C}_2\langle t[x\backslash s]\mathtt{L}\rangle$. Let us consider three subcases, depending on the position of the hole of $\mathtt{C}_2$ inside Σ.

First, if the hole of $\mathtt{C}_2$ lies inside $\mathtt{C}_3\langle\!\langle y\rangle\!\rangle$, then $\mathtt{C}_2 = \mathtt{C}_4[y\backslash r]$ and $\mathtt{C}_3\langle\!\langle y\rangle\!\rangle = \mathtt{C}_4\langle\Delta'\rangle$. By Lem. A.75 there is a context $\mathtt{C}'_3$ such that $\mathtt{C}_4\langle\Delta\rangle = \mathtt{C}'_3\langle\!\langle y\rangle\!\rangle$ and, moreover, the occurrence of y under $\mathtt{C}'_3$ is an ancestor of the occurrence of y under $\mathtt{C}_3$ before the step R. This means that S has an ancestor S_0, so this case is impossible. More precisely, the situation is:

$$\begin{array}{ccccccc} \mathtt{C}_1\langle\mathtt{C}'_3\langle\!\langle y\rangle\!\rangle[y\backslash r]\rangle & = & \mathtt{C}_1\langle\mathtt{C}_4\langle\Delta\rangle[y\backslash r]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\mathtt{C}_4\langle\Delta'\rangle[y\backslash r]\rangle & = & \mathtt{C}_1\langle\mathtt{C}'_3\langle\!\langle y\rangle\!\rangle[y\backslash r]\rangle \\ {\scriptstyle S_0}\downarrow & & & & & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\mathtt{C}_3\langle r\rangle[y\backslash r]\rangle & & & & & & \mathtt{C}_1\langle\mathtt{C}_3\langle r\rangle[y\backslash r]\rangle \end{array}$$

Second, if the hole of $\mathtt{C}_2$ lies inside r, then $\mathtt{C}_2 = \mathtt{C}_3\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{C}_4]$. Then S has an ancestor S_0, so this case is impossible. More precisely, the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle\mathtt{C}_3\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{C}_4\langle\Delta\rangle]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\mathtt{C}_3\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{C}_4\langle\Delta'\rangle]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\mathtt{C}_3\langle\mathtt{C}_4\langle\Delta\rangle\rangle[y\backslash\mathtt{C}_4\langle\Delta\rangle]\rangle & & \mathtt{C}_1\langle\mathtt{C}_3\langle\mathtt{C}_4\langle\Delta'\rangle\rangle[y\backslash\mathtt{C}_4\langle\Delta'\rangle]\rangle \end{array}$$

The interesting case is the last, when $\mathtt{C}_2$ is an empty context. Then $\mathtt{C}_3\langle\!\langle y\rangle\!\rangle[y\backslash u] = t[x\backslash s]\mathtt{L}$. Again we consider two subcases, depending on whether $\mathtt{L}$ is empty or non-empty:

1.2.1 **If $\mathtt{L}$ is empty.** Then $x = y$ and $s = u$, so the situation is:

$$t_1 = \mathtt{C}_1\langle(\lambda x.\mathtt{C}_5\langle\!\langle x\rangle\!\rangle)s\rangle \xrightarrow{R} \mathtt{C}_1\langle\mathtt{C}_5\langle\!\langle x\rangle\!\rangle[x\backslash s]\rangle = t_2$$

and we are in **Creation case 2:** $\mathtt{db}$ **creates** $\mathtt{ls}$.

1.2.2 **If $\mathtt{L}$ is non-empty.** Then $\mathtt{L} = \mathtt{L}'[y\backslash r]$ and $\mathtt{C}_3\langle\!\langle y\rangle\!\rangle = t[x\backslash s]\mathtt{L}'$. The situation is:

$$\mathtt{C}_1\langle(\lambda x.t)\mathtt{L}'[y\backslash u]s\rangle \xrightarrow{R} \mathtt{C}_1\langle t[x\backslash s]\mathtt{L}'[y\backslash u]\rangle$$

Note that y cannot occur in s by Barendregt's convention. So y occurs either in t or in $\mathtt{L}'$, which means that the redex S has an ancestor S_0 before R. For example, if y occurs in t, then $t = \mathtt{C}_5\langle\!\langle y \rangle\!\rangle$ and:

$$\begin{array}{ccc} \mathtt{C}_1\langle(\lambda x.\mathtt{C}_5\langle\!\langle y\rangle\!\rangle)\mathtt{L}'[y\backslash u]s\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\mathtt{C}_5\langle\!\langle y\rangle\!\rangle[x\backslash s]\mathtt{L}'[y\backslash u]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle(\lambda x.\mathtt{C}_5\langle u\rangle)\mathtt{L}'[y\backslash u]s\rangle & & \mathtt{C}_1\langle\mathtt{C}_5\langle u\rangle[x\backslash s]\mathtt{L}'[y\backslash u]\rangle \end{array}$$

1.3 **If S is a `gc` redex.** Then $\Sigma = u[y\backslash r] = \mathtt{C}_2\langle t[x\backslash s]\mathtt{L}\rangle$ with $y \notin \mathtt{fv}(u)$. Let us consider three subcases, depending on the position of the hole of $\mathtt{C}_2$ inside Σ.

First, if the hole of $\mathtt{C}_2$ lies inside u, then $u = \mathtt{C}_2\langle\Delta'\rangle$. Note that $\mathtt{fv}(\Delta) = \mathtt{fv}((\lambda x.t)\mathtt{L}\, s) = \mathtt{fv}(t[x\backslash s]\mathtt{L}) = \mathtt{fv}(\Delta')$, so $\mathtt{fv}(\mathtt{C}_2\langle\Delta\rangle) = \mathtt{fv}(\mathtt{C}_2\langle\Delta'\rangle)$. In particular, $y \notin \mathtt{fv}(\mathtt{C}_2\langle\Delta\rangle)$. Then this case is impossible, since S has an ancestor S_0. Graphically:

$$\begin{array}{ccc} \mathtt{C}_1\langle\mathtt{C}_2\langle\Delta\rangle[y\backslash r]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\mathtt{C}_2\langle\Delta'\rangle[y\backslash r]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\mathtt{C}_2\langle\Delta\rangle\rangle & & \mathtt{C}_1\langle\mathtt{C}_2\langle\Delta'\rangle\rangle \end{array}$$

Second, if the hole of $\mathtt{C}_2$ lies inside r, then $r = \mathtt{C}_2\langle\Delta\rangle$. This case is impossible since S has an ancestor S_0. Graphically:

$$\begin{array}{ccc} \mathtt{C}_1\langle u[y\backslash\mathtt{C}_2\langle\Delta\rangle]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle u[y\backslash\mathtt{C}_2\langle\Delta'\rangle]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle u\rangle & & \mathtt{C}_1\langle u\rangle \end{array}$$

Finally, if $\mathtt{C}_2$ is the empty context, then $u[y\backslash r] = t[x\backslash s]\mathtt{L}$. Again we consider two subcases, depending on whether `L` is empty or non-empty:

1.3.1 **If `L` is empty.** Then $x = y$, $t = u$, and $s = r$. Note that $x \notin \mathtt{fv}(t)$, so the situation is:

$$t_1 = (\lambda x.t)\mathtt{L}\, s \xrightarrow{R} t[x\backslash s]\mathtt{L} = t_2$$

and we are in **Creation case 3:** `db` **creates** `gc`.

1.3.2 **If `L` is non-empty.** Then $\mathtt{L} = \mathtt{L}'[y\backslash r]$, so $u = t[x\backslash s]\mathtt{L}'$. Note that $\mathtt{fv}((\lambda x.t)\mathtt{L}') \supseteq \mathtt{fv}(t[x\backslash s]\mathtt{L}')$, so $y \notin \mathtt{fv}((\lambda x.t)\mathtt{L}')$ implies $y \notin \mathtt{fv}(t[x\backslash s]\mathtt{L}')$. This means that this case is impossible since S has an ancestor S_0. More precisely the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle(\lambda x.t)\mathtt{L}'[y\backslash r]\, s\rangle & \xrightarrow{R} & \mathtt{C}_1\langle t[x\backslash s]\mathtt{L}'[y\backslash r]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle(\lambda x.t)\mathtt{L}'\, s\rangle & & \mathtt{C}_1\langle t[x\backslash s]\mathtt{L}'\rangle \end{array}$$

2. **If R is a `ls` redex.** Then $t_1 = \mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash t]\rangle \xrightarrow{R} \mathtt{C}_1\langle\mathtt{C}_2\langle t\rangle[x\backslash t]\rangle = t_2$. Consider the position of the hole of $\mathtt{C}_1$, relative to the position of the pattern Σ of S.

First, if the position of the hole of $\mathtt{C}_1$ is a prefix of the position of Σ, then there is a context $\mathtt{C}_3$ such that $\mathtt{C}_2\langle t\rangle[x\backslash t] = \mathtt{C}_3\langle\Sigma\rangle$. We consider three subcases for the position of the hole of $\mathtt{C}_3$: to the left of the substitution, inside the substitution, or at the root (*i.e.* $\mathtt{C}_3$ empty):

2.1 **If** $\mathtt{C}_3 = \mathtt{C}_4[x\backslash t]$**.** Then $\mathtt{C}_2\langle t\rangle = \mathtt{C}_4\langle\Sigma\rangle$. Now again we consider the position of the hole of $\mathtt{C}_2$ relative to the position of Σ.

First, if the position of the hole of $\mathtt{C}_2$ is a prefix of the position of Σ, then there is a context $\mathtt{C}'_4$ such that $\mathtt{C}_4 = \mathtt{C}_2\langle\mathtt{C}'_4\rangle$ and $t = \mathtt{C}'_4\langle\Sigma\rangle$. Then this case is impossible since S has an ancestor S_0. More precisely, the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{C}'_4\langle\Sigma\rangle]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\mathtt{C}_2\langle\mathtt{C}'_4\langle\Sigma\rangle\rangle[x\backslash\mathtt{C}'_4\langle\Sigma\rangle]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{C}'_4\langle\Sigma'\rangle]\rangle & & \mathtt{C}_1\langle\mathtt{C}_2\langle\mathtt{C}'_4\langle\Sigma'\rangle\rangle[x\backslash\mathtt{C}'_4\langle\Sigma\rangle]\rangle \end{array}$$

Second, if the positions of the hole of $\mathtt{C}_2$ and the hole of $\mathtt{C}_4$ are disjoint, then there exists a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box,\Sigma\rangle = \mathtt{C}_2$ and $\widehat{\mathtt{C}}\langle t,\Box\rangle = \mathtt{C}_4$. Then again this case is impossible since S has an ancestor S_0:

$$\begin{array}{ccc} \mathtt{C}_1\langle\widehat{\mathtt{C}}\langle x,\Sigma\rangle[x\backslash t]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\widehat{\mathtt{C}}\langle t,\Sigma\rangle[x\backslash t]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\widehat{\mathtt{C}}\langle x,\Sigma'\rangle[x\backslash t]\rangle & & \mathtt{C}_1\langle\widehat{\mathtt{C}}\langle t,\Sigma'\rangle[x\backslash t]\rangle \end{array}$$

Third, if the position of the hole of $\mathtt{C}_4$ is a prefix of the position of the hole of $\mathtt{C}_2$, then there is a context $\mathtt{C}'_2$ such that $\mathtt{C}_2 = \mathtt{C}_4\langle\mathtt{C}'_2\rangle$ and $\Sigma = \mathtt{C}'_2\langle t\rangle$. We consider three subcases depending on the kind of redex S:

2.1.1 **If** S **is a** `db` **redex.** Then $\Sigma = (\lambda y.s)\mathtt{L}\,u = \mathtt{C}'_2\langle t\rangle$. The proof proceeds by analyzing the position of the hole of $\mathtt{C}'_2$ inside Σ.
First note that, if $\mathtt{C}'_2$ is empty, we have already considered this situation since $\mathtt{C}_2$ is a prefix of $\mathtt{C}_4$.
Second, if the hole of $\mathtt{C}'_2$ lies inside s, inside u, or inside one of the substitutions in $\mathtt{L}$, then this case is impossible since S has an ancestor S_0. For example, if the hole of $\mathtt{C}'_2$ lies inside s, we have that $\mathtt{C}'_2 = (\lambda y.\mathtt{C}''_2)\mathtt{L}\,u$, $s = \mathtt{C}''_2\langle t\rangle$, and the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle\mathtt{C}_4\langle(\lambda y.\mathtt{C}''_2\langle\!\langle x\rangle\!\rangle)\mathtt{L}\,u\rangle[x\backslash t]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\mathtt{C}_4\langle(\lambda y.\mathtt{C}''_2\langle t\rangle)\mathtt{L}\,u\rangle[x\backslash t]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}''_2\langle\!\langle x\rangle\!\rangle\mathtt{L}[y\backslash u]\rangle[x\backslash t]\rangle & & \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}''_2\langle t\rangle\mathtt{L}[y\backslash u]\rangle[x\backslash t]\rangle \end{array}$$

The only remaining possibility is the interesting one, when the hole of $\mathtt{C}'_2$ lies somewhere along the spine of $(\lambda y.s)\mathtt{L}$, more precisely, there exist substitution contexts $\mathtt{L}_1$ and $\mathtt{L}_2$ such that $\mathtt{L} = \mathtt{L}_1\mathtt{L}_2$ and $\mathtt{C}'_2 = \mathtt{C}''_2\mathtt{L}_2\,u$. Then $t = (\lambda y.s)\mathtt{L}_1$ so the situation is:

$$t_1 = \mathtt{C}_1\langle\mathtt{C}_4\langle x\mathtt{L}_2\,u\rangle[x\backslash(\lambda y.s)\mathtt{L}_1]\rangle \xrightarrow{R} \mathtt{C}_1\langle\mathtt{C}_4\langle(\lambda y.s)\mathtt{L}_1\mathtt{L}_2\,u\rangle[x\backslash(\lambda y.s)\mathtt{L}_1]\rangle = t_2$$

and we are in **Creation case 5:** `ls` **creates** `db` **downwards**.

2.1.2 **If** S **is a** `ls` **redex.** Then $\Sigma = \mathtt{C}_5\langle\!\langle y\rangle\!\rangle[y\backslash s]$. The proof proceeds by analyzing the position of the hole of $\mathtt{C}'_2$ inside Σ.

First note that, if $\mathtt{C}_2'$ is empty, we have already considered this situation since $\mathtt{C}_2$ is a prefix of $\mathtt{C}_4$.
Second, if the hole of $\mathtt{C}_2'$ is to the left of the substitution and it is disjoint from the hole of $\mathtt{C}_5$, that is, there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box, y\rangle[y\backslash s] = \mathtt{C}_2'$ and $\widehat{\mathtt{C}}\langle t, \Box\rangle = \mathtt{C}_5$. Then this case is impossible as S has an ancestor S_0. More precisely, the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle\mathtt{C}_4\langle\widehat{\mathtt{C}}\langle x, y\rangle[y\backslash s]\rangle[x\backslash t]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\mathtt{C}_4\langle\widehat{\mathtt{C}}\langle t, y\rangle[y\backslash s]\rangle[x\backslash t]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\mathtt{C}_4\langle\widehat{\mathtt{C}}\langle x, s\rangle[y\backslash s]\rangle[x\backslash t]\rangle & & \mathtt{C}_1\langle\mathtt{C}_4\langle\widehat{\mathtt{C}}\langle t, s\rangle[y\backslash s]\rangle[x\backslash t]\rangle \end{array}$$

Third, if the hole of $\mathtt{C}_2'$ is to the left of the substitution and it is a prefix of the position of the hole of $\mathtt{C}_5$, that is, $\mathtt{C}_5 = \mathtt{C}_2'\langle\mathtt{C}_5'\rangle$. Then $t = \mathtt{C}_5'\langle\!\langle y\rangle\!\rangle$. Note that the steps R and S would need to be of the form:

$$\begin{array}{ccc} \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_2'\langle\!\langle x\rangle\!\rangle[y\backslash s]\rangle[x\backslash\mathtt{C}_5'\langle\!\langle y\rangle\!\rangle]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_2'\langle\mathtt{C}_5'\langle\!\langle y\rangle\!\rangle\rangle[y\backslash s]\rangle[x\backslash\mathtt{C}_5'\langle\!\langle y\rangle\!\rangle]\rangle \\ & & \downarrow{\scriptstyle S} \\ & & \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_2'\langle\mathtt{C}_5'\langle s\rangle\rangle[y\backslash s]\rangle[x\backslash\mathtt{C}_5'\langle\!\langle y\rangle\!\rangle]\rangle \end{array}$$

However, this case is impossible, since the variable y is outside the scope of the substitution binding y on the left-hand side of R, so by Barendregt's convention the step S cannot exist.
The only remaining possibility is that the hole of $\mathtt{C}_2'$ is inside the substitution, that is, $\mathtt{C}_2' = \mathtt{C}_5\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{C}_2'']$ with $s = \mathtt{C}_2''\langle t\rangle$. Then this case is impossible, as S has an ancestor S_0. In fact the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_5\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{C}_2''\langle\!\langle x\rangle\!\rangle]\rangle[x\backslash t]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_5\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{C}_2''\langle t\rangle]\rangle[x\backslash t]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_5\langle\mathtt{C}_2''\langle\!\langle x\rangle\!\rangle\rangle[y\backslash\mathtt{C}_2''\langle\!\langle x\rangle\!\rangle]\rangle[x\backslash t]\rangle & & \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_5\langle\mathtt{C}_2''\langle t\rangle\rangle[y\backslash\mathtt{C}_2''\langle t\rangle]\rangle[x\backslash t]\rangle \end{array}$$

2.1.3 **If S is a $\mathtt{gc}$ redex.** Then $\Sigma = s[y\backslash u] = \mathtt{C}_2'\langle t\rangle$, with $y \notin \mathtt{fv}(s)$. The proof proceeds by analyzing the position of the hole of $\mathtt{C}_2'$ inside Σ.
First note that, if $\mathtt{C}_2'$ is empty, we have already considered this situation since $\mathtt{C}_2$ is a prefix of $\mathtt{C}_4$.
Second, if the hole of $\mathtt{C}_2'$ is to the left of the substitution, that is, $\mathtt{C}_2' = \mathtt{C}_2''[y\backslash u]$ and $s = \mathtt{C}_2''\langle t\rangle$. Note that $y \notin s = \mathtt{C}_2''\langle t\rangle$ implies that $y \notin \mathtt{C}_2''\langle\!\langle x\rangle\!\rangle$, since $x \neq y$. Then this case is impossible since S has an ancestor S_0. More precisely, the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_2''\langle\!\langle x\rangle\!\rangle[y\backslash u]\rangle[x\backslash t]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_2''\langle t\rangle[y\backslash u]\rangle[x\backslash t]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_2''\langle\!\langle x\rangle\!\rangle\rangle[x\backslash t]\rangle & & \mathtt{C}_1\langle\mathtt{C}_4\langle\mathtt{C}_2''\langle t\rangle\rangle[x\backslash t]\rangle \end{array}$$

The only remaining possibility is that the hole of $\mathtt{C}_2'$ is inside the substitution, *i.e.* that $\mathtt{C}_2' = s[y\backslash\mathtt{C}_2'']$ with $u = \mathtt{C}_2''\langle t\rangle$. Then this case is impossible since S has

an ancestor S_0. More precisely, the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle \mathtt{C}_4\langle s[y\backslash \mathtt{C}_2''\langle\!\langle x\rangle\!\rangle]\rangle[x\backslash t]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle \mathtt{C}_4\langle s[y\backslash \mathtt{C}_2''\langle t\rangle]\rangle[x\backslash t]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle \mathtt{C}_4\langle s\rangle[x\backslash t]\rangle & & \mathtt{C}_1\langle \mathtt{C}_4\langle s\rangle[x\backslash t]\rangle \end{array}$$

2.2 **If** $\mathtt{C}_3 = \mathtt{C}_2\langle t\rangle[x\backslash \mathtt{C}_4]$**.** Then $t = \mathtt{C}_4\langle\Sigma\rangle$. This case is impossible, since S has an ancestor S_0. More precisely, the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{C}_4\langle\Sigma\rangle]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle \mathtt{C}_2\langle \mathtt{C}_4\langle\Sigma\rangle\rangle[x\backslash \mathtt{C}_4\langle\Sigma\rangle]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{C}_4\langle\Sigma'\rangle]\rangle & & \mathtt{C}_1\langle \mathtt{C}_2\langle \mathtt{C}_4\langle\Sigma\rangle\rangle[x\backslash \mathtt{C}_4\langle\Sigma'\rangle]\rangle \end{array}$$

2.3 **If** $\mathtt{C}_3 = \Box$**.** Then $\mathtt{C}_2\langle t\rangle[x\backslash t] = \Sigma$. This means that Σ must be either a `ls` redex or a `gc` redex, because its pattern is a substitution. Let us check each of these cases:

2.3.1 **If** S **is a** `ls` **redex.** Then $\mathtt{C}_2\langle t\rangle = \mathtt{C}_4\langle\!\langle x\rangle\!\rangle$. Note that it cannot be the case that $\mathtt{C}_2$ is a prefix of $\mathtt{C}_4$, since this would mean that $\mathtt{C}_4 = \mathtt{C}_2\langle \mathtt{C}_4'\rangle$ and $t = \mathtt{C}_4'\langle\!\langle x\rangle\!\rangle$. This would mean that the step R should be of the form

$$\mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x\rangle\!\rangle\rangle[x\backslash \mathtt{C}_4'\langle\!\langle x\rangle\!\rangle] \xrightarrow{R} \mathtt{C}_1\langle \mathtt{C}_2\langle \mathtt{C}_4'\langle\!\langle x\rangle\!\rangle\rangle\rangle[x\backslash \mathtt{C}_4'\langle\!\langle x\rangle\!\rangle]$$

but this is impossible by Barendregt's convention, since the free occurrence of the variable x in $\mathtt{C}_4'\langle\!\langle x\rangle\!\rangle$ becomes bound when performing the substitution. So the holes of $\mathtt{C}_2$ and $\mathtt{C}_4$ must be disjoint. More precisely, there exists a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle\Box, x\rangle = \mathtt{C}_2$ and $\widehat{\mathtt{C}}\langle t, \Box\rangle = \mathtt{C}_4$. Then this whole case is impossible, as S would have an ancestor S_0, as shown in the following diagram:

$$\begin{array}{ccc} \mathtt{C}_1\langle \widehat{\mathtt{C}}\langle x, x\rangle[x\backslash t]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle \widehat{\mathtt{C}}\langle t, x\rangle[x\backslash t]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle \widehat{\mathtt{C}}\langle x, t\rangle[x\backslash t]\rangle & & \mathtt{C}_1\langle \widehat{\mathtt{C}}\langle t, t\rangle[x\backslash t]\rangle \end{array}$$

2.3.2 **If** S **is a** `gc` **redex.** Then $x \notin \mathtt{fv}(\mathtt{C}_2\langle t\rangle)$. So the situation is:

$$t_1 = \mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash t]\rangle \xrightarrow{R} \mathtt{C}_1\langle \mathtt{C}_2\langle t\rangle[x\backslash t]\rangle = t_2$$

and we are in **Creation case 6:** `ls` **creates** `gc`.

Second, if the positions of the hole of $\mathtt{C}_1$ and the position of Σ are disjoint, then there must be a two-hole context $\widehat{\mathtt{C}}$ such that $\mathtt{C}_1 = \widehat{\mathtt{C}}\langle\Box, \Sigma\rangle$. Then this case is impossible, since S has an ancestor S_0. Graphically, the situation must be:

$$\begin{array}{ccc} \widehat{\mathtt{C}}\langle \mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash t], \Sigma\rangle & \xrightarrow{R} & \widehat{\mathtt{C}}\langle \mathtt{C}_2\langle t\rangle[x\backslash t], \Sigma\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \widehat{\mathtt{C}}\langle \mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash t], \Sigma'\rangle & & \widehat{\mathtt{C}}\langle \mathtt{C}_2\langle t\rangle[x\backslash t], \Sigma'\rangle \end{array}$$

Finally, if the position of Σ is a prefix of the position of the hole of $\mathtt{C}_1$, then $\mathtt{C}_1 = \mathtt{C}_{11}\langle \mathtt{C}_{12}\rangle$ such that $\Sigma = \mathtt{C}_{12}\langle \mathtt{C}_2\langle t\rangle[x\backslash t]\rangle = \mathtt{C}_{12}\langle\Delta'\rangle$. We proceed by case analysis on the kind of redex S:

2.1 **If S is a $\mathtt{db}$ redex.** Then $\Sigma = (\lambda y.s)\mathtt{L}\,u = \mathtt{C}_{12}\langle \mathtt{C}_2\langle t\rangle[x\backslash t]\rangle$. We proceed by case analysis on the position of the hole of $\mathtt{C}_{12}$ inside Σ.

First, if the hole of $\mathtt{C}_{12}$ lies inside s, or inside u, or inside one of the substitutions in $\mathtt{L}$, then this case is impossible since S has an ancestor S_0. For example, if the hole of $\mathtt{C}_{12}$ lies inside s, that is $\mathtt{C}_{12} = (\lambda y.\mathtt{C}'_{12})\mathtt{L}\,u$ with $s = \mathtt{C}'_{12}\langle\Delta'\rangle$, then the situation is:

$$\begin{array}{ccc} (\lambda y.\mathtt{C}'_{12}\langle\Delta\rangle)\mathtt{L}\,u & \xrightarrow{R} & (\lambda y.\mathtt{C}'_{12}\langle\Delta'\rangle)\mathtt{L}\,u \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}'_{12}\langle\Delta\rangle[y\backslash u]\mathtt{L} & & \mathtt{C}'_{12}\langle\Delta'\rangle[y\backslash u]\mathtt{L} \end{array}$$

The remaining possibility is that the hole of $\mathtt{C}_{12}$ lies along the spine of $(\lambda y.s)\mathtt{L}$. More precisely, there exist substitution contexts $\mathtt{L}_1$ and $\mathtt{L}_2$ such that $\mathtt{L} = \mathtt{L}_1\mathtt{L}_2$, with $\Delta' = (\lambda y.s)\mathtt{L}_1$ and $\mathtt{C}_{12} = \mathtt{L}_2\,u$. Then the situation is:

$$t_1 = \mathtt{C}_{11}\langle\underbrace{\mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash t]}_{\Delta}\,\mathtt{L}_2\,u\rangle \xrightarrow{R} \mathtt{C}_{11}\langle\underbrace{\mathtt{C}_2\langle t\rangle[x\backslash t]}_{\Delta'}\,\mathtt{L}_2\,u\rangle = t_2$$

where $\mathtt{C}_2\langle t\rangle[x\backslash t] = (\lambda y.s)\mathtt{L}_1$.

To conclude, note that there are two possibilities in this case, depending on whether the term $\mathtt{C}_2\langle\!\langle x\rangle\!\rangle$ is of the form $(\lambda y.s_0)\mathtt{L}_0$ (an answer) or not.

2.1.1 **If $\mathtt{C}_2\langle\!\langle x\rangle\!\rangle$ is an answer.** Then S has an ancestor S_0. Indeed, the situation is:

$$\begin{array}{ccccccc} \mathtt{C}_{11}\langle(\lambda y.s_0)\mathtt{L}_0\mathtt{L}_2\,u\rangle & = & \mathtt{C}_{11}\langle\underbrace{\mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash t]}_{\Delta}\,\mathtt{L}_2\,u\rangle & \overset{R}{\succ} & \mathtt{C}_{11}\langle\underbrace{\mathtt{C}_2\langle t\rangle[x\backslash t]}_{\Delta'}\,\mathtt{L}_2\,u\rangle & = & \mathtt{C}_{11}\langle\underbrace{(\lambda y.s)\mathtt{L}_1\mathtt{L}_2\,u}_{\Sigma}\rangle \\ {\scriptstyle S_0}\downarrow & & & & & & \downarrow{\scriptstyle S} \\ \mathtt{C}_{11}\langle s_0[y\backslash u]\mathtt{L}_0\mathtt{L}_2\rangle & & & & & & \mathtt{C}_{11}\langle\underbrace{s[y\backslash u]\mathtt{L}_1\mathtt{L}_2}_{\Sigma'}\rangle \end{array}$$

2.1.2 **If $\mathtt{C}_2\langle\!\langle x\rangle\!\rangle$ is not an answer.** Since $\mathtt{C}_2\langle\!\langle x\rangle\!\rangle$ is not an answer but $\mathtt{C}_2\langle t\rangle$ is an answer, by Lem. A.76 it must be the case that $\mathtt{C}_2$ is a substitution context $\mathtt{L}_3$, and t is an answer, $t = (\lambda y.t')\mathtt{L}'$. Hence the situation is:

$$t_1 = \mathtt{C}_{11}\langle x\mathtt{L}_3[x\backslash(\lambda y.t')\mathtt{L}']\mathtt{L}_2\,u\rangle \xrightarrow{R} \mathtt{C}_{11}\langle(\lambda y.t')\mathtt{L}'\mathtt{L}_3[x\backslash(\lambda y.t')\mathtt{L}']\mathtt{L}_2\,u\rangle = t_2$$

and we are in **Creation case 4: $\mathtt{ls}$ creates $\mathtt{db}$ upwards**.

2.2 **If S is a $\mathtt{ls}$ redex.** Then $\Sigma = \mathtt{C}_3\langle\!\langle y\rangle\!\rangle[y\backslash s] = \mathtt{C}_{12}\langle\mathtt{C}_2\langle t\rangle[x\backslash t]\rangle$. We proceed by case analysis on the position of the hole of $\mathtt{C}_{12}$ inside Σ.

First, if the hole of $\mathtt{C}_{12}$ is to the left of the substitution and disjoint of the hole of $\mathtt{C}_3[y\backslash s]$, more precisely if there exists a two hole context $\widehat{\mathtt{C}}$ such that $\mathtt{C}_{12} = \widehat{\mathtt{C}}\langle\Box, y\rangle[y\backslash s]$ and $\mathtt{C}_3 = \widehat{\mathtt{C}}\langle\Delta', \Box\rangle$, then this case is impossible, since S has an ancestor S_0:

$$\begin{array}{ccc} \mathtt{C}_{11}\langle\widehat{\mathtt{C}}\langle\Delta, y\rangle[y\backslash s]\rangle & \xrightarrow{R} & \mathtt{C}_{11}\langle\widehat{\mathtt{C}}\langle\Delta', y\rangle[y\backslash s]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_{11}\langle\widehat{\mathtt{C}}\langle\Delta, s\rangle[y\backslash s]\rangle & & \mathtt{C}_{11}\langle\widehat{\mathtt{C}}\langle\Delta', s\rangle[y\backslash s]\rangle \end{array}$$

Second, if the hole of $\mathtt{C}_{12}$ is to the left the substitution and it is a prefix of $\mathtt{C}_3[y\backslash s]$, more precisely if $\mathtt{C}_{12} = \mathtt{C}'_{12}[y\backslash s]$ and $\mathtt{C}_3 = \mathtt{C}'_{12}\langle \mathtt{C}'_3 \rangle$, then we have that $\mathtt{C}'_3\langle\!\langle y \rangle\!\rangle = \mathtt{C}_2\langle t \rangle[x\backslash t]$. We consider three subcases, depending on whether the hole of $\mathtt{C}'_3$ lies inside the left copy of t, inside the right copy of t, or in a disjoint position in $\mathtt{C}_2$:

2.2.1 **If the hole of $\mathtt{C}'_3$ lies inside the left copy of t .** That is, $\mathtt{C}'_3 = \mathtt{C}_2\langle \mathtt{C}_4 \rangle[x\backslash t]$. Then this case is impossible, as S has an ancestor S_0. More precisely:

$$\begin{array}{ccc} \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \mathtt{C}_2\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{C}_4\langle\!\langle y \rangle\!\rangle]\rangle[y\backslash s]\rangle & \xrightarrow{R} & \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \mathtt{C}_2\langle \underbrace{\mathtt{C}_4\langle\!\langle y \rangle\!\rangle}_{t}\rangle[x\backslash \underbrace{\mathtt{C}_4\langle\!\langle y \rangle\!\rangle}_{t}]\rangle[y\backslash s]\rangle \\ \Big\downarrow{\scriptstyle S_0} & & \\ \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \mathtt{C}_2\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{C}_4\langle s \rangle]\rangle[y\backslash s]\rangle & & \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \mathtt{C}_2\langle \mathtt{C}_4\langle s \rangle\rangle[x\backslash \mathtt{C}_4\langle\!\langle y \rangle\!\rangle]\rangle[y\backslash s]\rangle \end{array} \quad \curlyvee S$$

2.2.2 **If the hole of $\mathtt{C}'_3$ lies inside the right copy of t .** That is, $\mathtt{C}'_3 = \mathtt{C}_2\langle t \rangle[x\backslash \mathtt{C}_4]$. Similar to the previous case.

2.2.3 **If the hole of $\mathtt{C}'_3$ lies in a disjoint position of $\mathtt{C}_2$.** Then there is a two hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle \Box, t \rangle[x\backslash t] = \mathtt{C}_3$ and $\widehat{\mathtt{C}}\langle y, \Box \rangle = \mathtt{C}_2$. Then this case is impossible, as S has an ancestor S_0. More precisely:

$$\begin{array}{ccc} \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \widehat{\mathtt{C}}\langle y, x \rangle[x\backslash t]\rangle[y\backslash s]\rangle & \xrightarrow{R} & \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \widehat{\mathtt{C}}\langle y, t \rangle[x\backslash t]\rangle[y\backslash s]\rangle \\ \Big\downarrow{\scriptstyle S_0} & & \Big\downarrow{\scriptstyle S} \\ \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \widehat{\mathtt{C}}\langle s, x \rangle[x\backslash t]\rangle[y\backslash s]\rangle & & \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \widehat{\mathtt{C}}\langle s, t \rangle[x\backslash t]\rangle[y\backslash s]\rangle \end{array}$$

Third, if the hole of $\mathtt{C}_{12}$ lies inside the substitution $[y\backslash s]$, that is, $\mathtt{C}_{12} = \mathtt{C}_3\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{C}'_{12}]$, then this case is impossible, since S has an ancestor S_0. More precisely:

$$\begin{array}{ccc} \mathtt{C}_{11}\langle \mathtt{C}_3\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{C}'_{12}\langle \Delta \rangle]\rangle & \xrightarrow{R} & \mathtt{C}_{11}\langle \mathtt{C}_3\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{C}'_{12}\langle \Delta' \rangle]\rangle \\ \Big\downarrow{\scriptstyle S_0} & & \Big\downarrow{\scriptstyle S} \\ \mathtt{C}_{11}\langle \mathtt{C}_3\langle \mathtt{C}'_{12}\langle \Delta \rangle\rangle[y\backslash \mathtt{C}'_{12}\langle \Delta \rangle]\rangle & & \mathtt{C}_{11}\langle \mathtt{C}_3\langle \mathtt{C}'_{12}\langle \Delta' \rangle\rangle[y\backslash \mathtt{C}'_{12}\langle \Delta' \rangle]\rangle \end{array}$$

The remaining possibility is that $\mathtt{C}_{12}$ is empty, that is, $\mathtt{C}_{12} = \Box$, $x = y$, $t = s$, and there is a two-hole context $\widehat{\mathtt{C}}$ such that $\widehat{\mathtt{C}}\langle \Box, x \rangle = \mathtt{C}_2$ and $\widehat{\mathtt{C}}\langle t, \Box \rangle = \mathtt{C}_3$. Then this case is impossible, since S has an ancestor S_0. The situation is:

$$\begin{array}{ccc} \mathtt{C}_{11}\langle \widehat{\mathtt{C}}\langle x, x \rangle[x\backslash t]\rangle & \xrightarrow{R} & \mathtt{C}_{11}\langle \widehat{\mathtt{C}}\langle t, x \rangle[x\backslash t]\rangle \\ \Big\downarrow{\scriptstyle S_0} & & \Big\downarrow{\scriptstyle S} \\ \mathtt{C}_{11}\langle \widehat{\mathtt{C}}\langle x, t \rangle[x\backslash t]\rangle & & \mathtt{C}_{11}\langle \widehat{\mathtt{C}}\langle t, t \rangle[x\backslash t]\rangle \end{array}$$

2.3 **If S is a $\mathtt{gc}$ redex.** Then $\Sigma = s[y\backslash u] = \mathtt{C}_{12}\langle \mathtt{C}_2\langle t \rangle[x\backslash t]\rangle$ where $y \notin \mathtt{fv}(s)$. We proceed by case analysis on the position of the hole of $\mathtt{C}_{12}$ inside Σ.

First, if the hole of $\mathtt{C}_{12}$ is to the left of the substitution, that is, $\mathtt{C}_{12} = \mathtt{C}'_{12}[y\backslash u]$, then this case is impossible since S has an ancestor S_0. More precisely:

$$\begin{array}{ccc} \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \Delta \rangle[y\backslash u]\rangle & \xrightarrow{R} & \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \Delta' \rangle[y\backslash u]\rangle \\ \Big\downarrow{\scriptstyle S_0} & & \Big\downarrow{\scriptstyle S} \\ \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \Delta \rangle\rangle & & \mathtt{C}_{11}\langle \mathtt{C}'_{12}\langle \Delta' \rangle\rangle \end{array}$$

Observe that $\mathtt{fv}(\Delta) = \mathtt{fv}(\mathtt{C}_2\langle\!\langle x \rangle\!\rangle[x\backslash t]) = \mathtt{fv}(\mathtt{C}_2\langle t\rangle[x\backslash t]) = \mathtt{fv}(\Delta')$ so from the fact that $y \notin \mathtt{fv}(\mathtt{C}'_{12}\langle\Delta'\rangle)$ we may conclude that $y \notin \mathtt{fv}(\mathtt{C}'_{12}\langle\Delta\rangle)$.

Second, if the hole of $\mathtt{C}_{12}$ is inside the substitution, that is, $\mathtt{C}_{12} = s[y\backslash\mathtt{C}'_{12}]$ then this case is impossible since S has an ancestor S_0. More precisely:

$$\begin{array}{ccc} \mathtt{C}_{11}\langle s[y\backslash\mathtt{C}'_{12}\langle\Delta\rangle]\rangle & \xrightarrow{R} & \mathtt{C}_{11}\langle s[y\backslash\mathtt{C}'_{12}\langle\Delta'\rangle]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_{11}\langle s\rangle & & \mathtt{C}_{11}\langle s\rangle \end{array}$$

The only remaining possibility is that $\mathtt{C}_{12}$ is empty, *i.e.* $\mathtt{C}_{12} = \Box$, $x = y$, $u = t$, and $s = \mathtt{C}_2\langle t\rangle$. Then the situation is the following, with $x \notin \mathtt{fv}(\mathtt{C}_2\langle t\rangle)$:

$$t_1 = \mathtt{C}_{11}\langle\mathtt{C}_2\langle\!\langle x \rangle\!\rangle[x\backslash t]\rangle \xrightarrow{R} \mathtt{C}_{11}\langle\mathtt{C}_2\langle t\rangle[x\backslash t]\rangle = t_2$$

and we are in **Creation case 6: `ls` creates `gc`.**

3. **If R is a `gc` redex.** Then $t_1 = \mathtt{C}\langle t[x\backslash s]\rangle \xrightarrow{R} \mathtt{C}\langle t\rangle = t_2$ with $x \notin \mathtt{fv}(t)$. Consider the position of the hole of $\mathtt{C}$, relative to the position of the pattern Σ of S.

   First, if the position of the hole of $\mathtt{C}$ is a prefix of the position of Σ, then $t = \mathtt{C}_1\langle\Sigma\rangle$. Then this case is impossible since S has an ancestor S_0. The situation is:

$$\begin{array}{ccc} \mathtt{C}\langle\mathtt{C}_1\langle\Sigma\rangle[x\backslash s]\rangle & \xrightarrow{R} & \mathtt{C}\langle\mathtt{C}_1\langle\Sigma\rangle\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}\langle\mathtt{C}_1\langle\Sigma'\rangle[x\backslash s]\rangle & & \mathtt{C}\langle\mathtt{C}_1\langle\Sigma'\rangle\rangle \end{array}$$

   Second, if the position of the holes of $\mathtt{C}$ and the position of Σ are disjoint, then there is a two-hole context $\widehat{\mathtt{C}}$ such that $\mathtt{C} = \widehat{\mathtt{C}}\langle\Box, \Sigma\rangle$. Then this case is impossible since S has an ancestor S_0. The situation is:

$$\begin{array}{ccc} \mathtt{C}\langle\widehat{\mathtt{C}}\langle t[x\backslash s], \Sigma\rangle\rangle & \xrightarrow{R} & \mathtt{C}\langle\widehat{\mathtt{C}}\langle t, \Sigma\rangle\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}\langle\widehat{\mathtt{C}}\langle t[x\backslash s], \Sigma'\rangle\rangle & & \mathtt{C}\langle\widehat{\mathtt{C}}\langle t, \Sigma'\rangle\rangle \end{array}$$

   Finally, if the position of Σ is a prefix of the position of the hole of $\mathtt{C}$, then $\mathtt{C} = \mathtt{C}_1\langle\mathtt{C}_2\rangle$ such that $\Sigma = \mathtt{C}_2\langle t\rangle$. We proceed by case analysis on the kind of redex S:

   3.1 **If S is a `db` redex.** That is $\Sigma = (\lambda y.u)\mathtt{L}r = \mathtt{C}_2\langle t\rangle$. Then the hole of $\mathtt{C}_2$ lies either inside u, inside r, inside one of the substitutions of $\mathtt{L}$, or along the spine of $(\lambda y.u)\mathtt{L}$ (*i.e.* there exist substitution contexts $\mathtt{L}_1$, $\mathtt{L}_2$ such that $\mathtt{L} = \mathtt{L}_1\mathtt{L}_2$ and $\mathtt{C}_2 = \mathtt{L}_2 r$). In any case, this case is impossible since S has an ancestor S_0. For example, if the hole of $\mathtt{C}_2$ lies inside u, then $\mathtt{C}_2 = (\lambda y.\mathtt{C}'_2)\mathtt{L}r$ and the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle(\lambda y.\mathtt{C}'_2\langle t[x\backslash s]\rangle)\mathtt{L}r\rangle & \xrightarrow{R} & \mathtt{C}_1\langle(\lambda y.\mathtt{C}'_2\langle t\rangle)\mathtt{L}r\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle\mathtt{C}'_2\langle t[x\backslash s]\rangle[y\backslash r]\mathtt{L}\rangle & & \mathtt{C}_1\langle\mathtt{C}'_2\langle t\rangle[y\backslash r]\mathtt{L}\rangle \end{array}$$

3.2 **If S is a $\mathtt{ls}$ redex.** That is $\Sigma = \mathtt{C}_3\langle\!\langle y \rangle\!\rangle[y\backslash u] = \mathtt{C}_2\langle t\rangle$. Then the hole of $\mathtt{C}_2$ lies either inside u, inside $\mathtt{C}_3$ disjoint from the variable y, or it is a prefix of $\mathtt{C}_3[y\backslash u]$. In any case, this case is impossible since S has an ancestor S_0. For example, if the hole of $\mathtt{C}_2$ lies inside u, then $\mathtt{C}_2 = \mathtt{C}_3\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{C}'_2]$ and the situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle \mathtt{C}_3\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{C}'_2\langle t[x\backslash s]\rangle]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle \mathtt{C}_3\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{C}'_2\langle t\rangle]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle \mathtt{C}_3\langle \mathtt{C}'_2\langle t[x\backslash s]\rangle\rangle[y\backslash \mathtt{C}'_2\langle t[x\backslash s]\rangle]\rangle & & \mathtt{C}_1\langle \mathtt{C}_3\langle \mathtt{C}'_2\langle t\rangle\rangle[y\backslash \mathtt{C}'_2\langle t\rangle]\rangle \end{array}$$

3.3 **If S is a $\mathtt{gc}$ redex.** That is $\Sigma = u[y\backslash r] = \mathtt{C}_2\langle t\rangle$ with $y \notin \mathtt{fv}(u)$. Then the hole of $\mathtt{C}_2$ lies either inside u or inside r. Let us consider each case.

First, if the hole of $\mathtt{C}_2$ lies inside u, that is $\mathtt{C}_2 = \mathtt{C}'_2[y\backslash r]$, there are two subcases, depending on whether $y \in \mathtt{fv}(s)$:

3.3.1 **If $y \in \mathtt{fv}(s)$.** Then the situation is:

$$t_1 = \mathtt{C}_1\langle \mathtt{C}'_2\langle t[x\backslash s]\rangle[y\backslash u]\rangle \xrightarrow{R} \mathtt{C}_1\langle \mathtt{C}'_2\langle t\rangle[y\backslash u]\rangle = t_2$$

where $y \in \mathtt{fv}(s)$ and $y \notin \mathtt{fv}(\mathtt{C}'_2\langle t\rangle)$, so we are in **Creation case 7: $\mathtt{gc}$ creates $\mathtt{gc}$.**

3.3.2 **If $y \notin \mathtt{fv}(s)$.** Then note that $y \notin \mathtt{fv}(\mathtt{C}'_2\langle t[x\backslash s]\rangle)$, so this case is impossible, since S has an ancestor S_0. The situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle \mathtt{C}'_2\langle t[x\backslash s]\rangle[y\backslash r]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle \mathtt{C}'_2\langle t\rangle[y\backslash r]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle \mathtt{C}'_2\langle t[x\backslash s]\rangle\rangle & & \mathtt{C}_1\langle \mathtt{C}'_2\langle t\rangle\rangle \end{array}$$

Second, if the hole of $\mathtt{C}_2$ lies inside r, that is $\mathtt{C}_2 = u[y\backslash \mathtt{C}'_2]$, then this case is impossible, since S has an ancestor S_0. The situation is:

$$\begin{array}{ccc} \mathtt{C}_1\langle u[y\backslash \mathtt{C}'_2\langle t[x\backslash s]\rangle]\rangle & \xrightarrow{R} & \mathtt{C}_1\langle u[y\backslash \mathtt{C}'_2\langle t\rangle]\rangle \\ {\scriptstyle S_0}\downarrow & & \downarrow{\scriptstyle S} \\ \mathtt{C}_1\langle u\rangle & & \mathtt{C}_1\langle u\rangle \end{array}$$

□

A.3.2 Strong permutation — proof of Prop. 6.30

For the proof of Prop. 6.30 we need an auxiliary technical tool. We already know that adding a label to a context is not always defined as a context. For instance, $\alpha : \Box$ is not a valid labeled context. Sometimes it will be convenient to allow this to stand for a generalized notion of contexts, which we call *pseudo-contexts*. Pseudo-contexts will be allowed to have a label decorating the hole. For instance, $\alpha : \Box$ will be a pseudo-context such that $(\alpha : \Box)\langle t\rangle = \alpha : t$.

Definition A.78 (Pseudo-contexts)**.** A pseudo-context P is given by the following grammar, where $\mathtt{C}$ is a regular context:

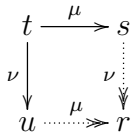
$$P ::= \mathtt{C} \mid \mathtt{C}\langle \alpha : \Box \rangle$$

The operation of plugging a term into a pseudo-context is defined as follows:

$$P\langle t \rangle \stackrel{\text{def}}{=} \begin{cases} \mathtt{C}\langle t \rangle & \text{if } P = \mathtt{C} \\ \mathtt{C}\langle \alpha : t \rangle & \text{if } P = \mathtt{C}\langle \alpha : \Box \rangle \end{cases}$$

The result of adding a label α to the empty context $\Box$ can be defined to be precisely the pseudo-context $\alpha : \Box$. With this extension, the operation $\alpha : \mathtt{C}$ can always be thought as yielding a pseudo-context.

Proposition A.79 (Full proof of Prop. 6.30—Strong permutation)**.** *Let $R : t \xrightarrow{\mu}_\ell s$ and $S : t \xrightarrow{\nu}_\ell u$ be steps in the LLSC. Then there exists a term r and two derivations $\sigma : s \xrightarrow{\!\!\nu\!\!\!\!\rightarrow}_\ell r$ and $\rho : u \xrightarrow{\!\!\mu\!\!\!\!\rightarrow}_\ell r$. Diagrammatically:*



Moreover:

1. *If R is a* $\mathtt{db}$ *step, σ consists of exactly one step.*
2. *If R is a* $\mathtt{ls}$ *step, σ may consist of one or two steps.*
3. *If R is a* $\mathtt{gc}$ *step, σ may consist of zero or one steps.*

And symmetrically for S and ρ.

Proof. If R and S lie in disjoint positions, the result is immediate. The non-trivial case is, without loss of generality, when the position of the redex occurrence R is a prefix of the position of the redex occurrence S. We only consider the case when R and S are different redexes. Note that even if R and S are different, they might lie in the same position; for instance $(x^{\mathbf{a}}x^{\mathbf{b}})[x\backslash y^{\mathbf{c}}]_\Omega$ has two $\mathtt{ls}$ redexes at the root. The proof is by induction on the context $\mathtt{C}$ under which the redex occurrence R is contracted:

1. *Base case,* $\mathtt{C} = \Box$. Depending on the kind of the redex R:

   1.1 *R is a* $\mathtt{db}$*-redex.* That is $t = @^\alpha((\lambda^\beta_\Omega x.t')\mathtt{L}, s')$ and $\mu = \mathtt{db}(\beta)$. If S is internal to t', s' or $\mathtt{L}$ (*i.e.* without overlapping the hole of $\mathtt{L}$), the steps are disjoint, and it is immediate. Furthermore, since any application must be internal to t', s' or $\mathtt{L}$, we have already considered all the possible cases of S being a $\mathtt{db}$-redex.

   The remaining possibilities are that S is a $\mathtt{ls}$-redex or a $\mathtt{gc}$-redex, involving one of the substitutions in $\mathtt{L}$. That is, $\mathtt{L}$ must have the form $\mathtt{L}_1[y\backslash u']_\Theta\mathtt{L}_2$, and one of the two following cases applies:

   1.1.1 *S is a* $\mathtt{ls}$*-redex, contracting* $[y\backslash u']_\Theta$.

1.1.2 *S is a* gc*-redex, erasing* $[y\backslash u']_\Theta$.

Let us prove each case separately:

1.1.1 *S is a* ls*-redex, contracting* $[y\backslash u']_\Theta$. Since there is a ls-redex, we know that $(\lambda^\alpha_\Omega x.t')\mathtt{L}_1$ must be of the form $\mathtt{C}'\langle\!\langle y^\gamma \rangle\!\rangle$. The contracted occurrence of y can be either inside t' or inside $\mathtt{L}_1$. The name of the redex S is $\nu = \downarrow(\gamma) \bullet \uparrow(u')$. We consider two subcases:

1.1.1.1 If the affected occurrence is inside t', we have $t' = \mathtt{C}_1\langle\!\langle y^\gamma \rangle\!\rangle$. Let $\widehat{t'}$ be the corresponding term after contracting the affected occurrence of y, *i.e.*: $\widehat{t'} := \mathtt{C}_1\langle \gamma \bullet : u' \rangle$. Then:

$$\begin{array}{ccc} @^\alpha((\lambda^\beta_\Omega x.t')\mathtt{L}_1[y\backslash u']_\Theta \mathtt{L}_2, s') & \xrightarrow{\mathtt{db}(\beta)} & \alpha\lceil \mathtt{db}(\beta)\rceil : t'[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : s']_\Omega \mathtt{L}_1[y\backslash u']_\Theta \mathtt{L}_2 \\ {\scriptstyle \downarrow(\gamma)\bullet\uparrow(u')}\big\downarrow & & \big\downarrow{\scriptstyle \downarrow(\gamma)\bullet\uparrow(u')} \\ @^\alpha((\lambda^\beta_\Omega x.\widehat{t'})\mathtt{L}_1[y\backslash u']_\Theta \mathtt{L}_2, s') & \xrightarrow{\mathtt{db}(\beta)} & \alpha\lceil \mathtt{db}(\beta)\rceil : \widehat{t'}[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : s']_\Omega \mathtt{L}_1[y\backslash u']_\Theta \mathtt{L}_2 \end{array}$$

Note that on the right hand side we are using Lem. 6.9 to conclude that:

$$\alpha\lceil \mathtt{db}(\beta)\rceil : t' = \alpha\lceil \mathtt{db}(\beta)\rceil : \mathtt{C}_1\langle\!\langle y^\gamma \rangle\!\rangle \quad \text{is of the form} \quad \mathtt{C}'_1\langle\!\langle y^{\gamma'} \rangle\!\rangle$$

where $\downarrow(\gamma) = \downarrow(\gamma')$ and, moreover:

$$\alpha\lceil \mathtt{db}(\beta)\rceil : \widehat{t'} = \alpha\lceil \mathtt{db}(\beta)\rceil : \mathtt{C}_1\langle \gamma \bullet : u' \rangle = \mathtt{C}'_1\langle \gamma' \bullet : u' \rangle$$

1.1.1.2 If the affected occurrence is inside $\mathtt{L}_1$, we have that $\mathtt{L}_1 = \mathtt{L}_\mathtt{A}[z\backslash \mathtt{C}_1\langle\!\langle y^\gamma \rangle\!\rangle]_\Psi \mathtt{L}_\mathtt{B}$. Let $\widehat{\mathtt{L}_1}$ be the corresponding substitution context after contracting the affected occurrence of y, *i.e.* $\widehat{\mathtt{L}_1} := \mathtt{L}_\mathtt{A}[z\backslash \mathtt{C}_1\langle \gamma \bullet : u' \rangle]_\Psi \mathtt{L}_\mathtt{B}$. The situation is then:

$$\begin{array}{ccc} @^\alpha((\lambda^\beta_\Omega x.t')\mathtt{L}_1[y\backslash u']_\Theta \mathtt{L}_2, s') & \xrightarrow{\mathtt{db}(\beta)} & \alpha\lceil \mathtt{db}(\beta)\rceil : t'[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : s']_\Omega \mathtt{L}_1[y\backslash u']_\Theta \mathtt{L}_2 \\ {\scriptstyle \downarrow(\gamma)\bullet\uparrow(u')}\big\downarrow & & \big\downarrow{\scriptstyle \downarrow(\gamma)\bullet\uparrow(u')} \\ @^\alpha((\lambda^\beta_\Omega x.t')\widehat{\mathtt{L}_1}[y\backslash u']_\Theta \mathtt{L}_2, s') & \xrightarrow{\mathtt{db}(\beta)} & \alpha\lceil \mathtt{db}(\beta)\rceil : t'[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : s']_\Omega \widehat{\mathtt{L}_1}[y\backslash u']_\Theta \mathtt{L}_2 \end{array}$$

1.1.2 *S is a* gc*-redex, erasing* $[y\backslash u']_\Theta$. Since it is a gc-redex, we know $(\lambda^\beta_\Omega x.t')\mathtt{L}_1$ has no free occurrences of y. The name of the redex S is then: $\nu = \{\mathbf{a} \bullet \uparrow(u') \mid \mathbf{a} \in \Theta\}$. By the usual fact that reduction cannot create free variables, we have:

$$\begin{array}{ccc} @^\alpha((\lambda^\beta_\Omega x.t')\mathtt{L}_1[y\backslash u']_\Theta \mathtt{L}_2, s') & \xrightarrow{\mathtt{db}(\beta)} & \alpha\lceil \mathtt{db}(\beta)\rceil : t'[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : s']_\Omega \mathtt{L}_1[y\backslash u']_\Theta \mathtt{L}_2 \\ {\scriptstyle \{\mathbf{a}\bullet\uparrow(u') \mid \mathbf{a}\in\Theta\}}\big\downarrow & & \big\downarrow{\scriptstyle \{\mathbf{a}\bullet\uparrow(u') \mid \mathbf{a}\in\Theta\}} \\ @^\alpha((\lambda^\beta_\Omega x.t')\mathtt{L}_1\mathtt{L}_2, s') & \xrightarrow{\mathtt{db}(\beta)} & \alpha\lceil \mathtt{db}(\beta)\rceil : t'[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : s']_\Omega \mathtt{L}_1\mathtt{L}_2 \end{array}$$

1.2 *R is a* ls*-redex.* That is $t = \mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle[x\backslash t']_\Omega$ and $\mu = \downarrow(\alpha) \bullet \uparrow(t')$. Note that if S is internal to $\mathtt{C}$ (*i.e.* without overlapping the hole of $\mathtt{C}$), the steps are disjoint, and the proof is direct. If S is internal to t', it is also straightforward to close the diagram, although the ls-step duplicates t', which requires contracting two residuals of S.

More precisely, suppose $t' \xrightarrow{\nu}_{\ell} \widehat{t'}$; then:

$$\begin{array}{ccc}
\mathtt{C}\langle\!\langle x^{\alpha} \rangle\!\rangle[x\backslash t']_{\Omega} & \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t')} & \mathtt{C}\langle\!\langle \alpha \bullet : t' \rangle\!\rangle[x\backslash t']_{\Omega} \\
\Big\downarrow{\scriptstyle \nu} & & \Big\downarrow{\scriptstyle \nu}\ \star \\
\mathtt{C}\langle\!\langle x^{\alpha} \rangle\!\rangle[x\backslash \widehat{t'}]_{\Omega} & \xrightarrow{\downarrow(\alpha)\bullet\uparrow(\widehat{t'})} & \mathtt{C}\langle\!\langle \alpha \bullet : \widehat{t'} \rangle\!\rangle[x\backslash \widehat{t'}]_{\Omega}
\end{array}$$

Note that the name ν in the step marked with $\star$ is not changed, by the fact that adding labels preserves redex names (Lem. 6.10). To close this diagram, note also that $\uparrow(\widehat{t'}) = \uparrow(t')$ by the fact that reduction preserves the first label of a term (Lem. 6.11).

We have already considered the cases when S is internal to $\mathtt{C}$ and internal to t'. The remaining cases are that the redex occurrence S contains as a subterm either the affected variable x^{α} or the affected substitution $[x\backslash t']_{\Omega}$. Some situations are impossible and can be dismissed:

- A db-step cannot possibly involve $[x\backslash t']_{\Omega}$, since there is no application node that contains such substitution.
- A gc-step cannot erase $[x\backslash t']_{\Omega}$, since there is at least one free occurrence of x in $\mathtt{C}\langle\!\langle x^{\alpha} \rangle\!\rangle$.

So we are left to check the following cases:

1.2.1 *S is a* db*-redex, including x^{α} as a subterm*

1.2.2 *S is a* ls*-redex, including x^{α} as a subterm*

1.2.3 *S is a* ls*-redex, contracting $[x\backslash t']_{\Omega}$*

1.2.4 *S is a* gc*-redex, including x^{α} as a subterm*

Let us prove each of these separately:

1.2.1 *S is a* db*-redex, including x^{α} as a subterm.* Let q be the subterm corresponding to the db-redex S. Since q is a subterm of $\mathtt{C}\langle\!\langle x^{\alpha} \rangle\!\rangle$, we know $\mathtt{C}\langle\!\langle x^{\alpha} \rangle\!\rangle = \mathtt{C}_1\langle q \rangle$. Also, since q is a db-redex, it has the form $q = @^{\beta}((\lambda^{\gamma}_{\Theta} y.s')\mathtt{L}, u')$ where $\nu = \mathtt{db}(\gamma)$. Note that for this case we are also assuming that the occurrence of the affected variable x^{α} lies inside this subterm q. This leads to three cases for $\mathtt{C}$, depending on whether the hole of $\mathtt{C}$ corresponds to a position inside s', inside u' or inside $\mathtt{L}$:

$$\mathtt{C} = \begin{cases}
\mathtt{C}_1\langle\, @^{\beta}((\lambda^{\gamma}_{\Theta} y.\mathtt{C}_2)\mathtt{L}, u')\,\rangle & \text{with } \mathtt{C}_2\langle\!\langle x^{\alpha} \rangle\!\rangle = s' \\
 & \textit{“the hole is internal to } s'\textit{”} \\
\mathtt{C}_1\langle\, @^{\beta}((\lambda^{\gamma}_{\Theta} y.s')\mathtt{L}, \mathtt{C}_2)\,\rangle & \text{with } \mathtt{C}_2\langle\!\langle x^{\alpha} \rangle\!\rangle = u' \\
 & \textit{“the hole is internal to } u'\textit{”} \\
\mathtt{C}_1\langle\, @^{\beta}((\lambda^{\gamma}_{\Theta} y.s')\mathtt{L}_1[z\backslash \mathtt{C}_2]_{\Psi}\mathtt{L}_2, u')\,\rangle & \text{with } \mathtt{L}_1[z\backslash \mathtt{C}_2\langle\!\langle x^{\alpha} \rangle\!\rangle]_{\Psi}\mathtt{L}_2 = \mathtt{L} \\
 & \textit{“the hole is internal to } \mathtt{L}\textit{”}
\end{cases}$$

Respectively for each case, let $\widehat{\mathtt{C}}$ be the *pseudo-context* that results after contracting the db-redex in $\mathtt{C}$. Respectively in each of the three cases above:

$$\mathtt{C} \xrightarrow{\mathtt{db}(\gamma)}_{\ell} \widehat{\mathtt{C}} = \begin{cases} \beta\lceil \mathtt{db}(\gamma)\rceil : \mathtt{C}_2[y\backslash\lfloor \mathtt{db}(\gamma)\rfloor : u']_{\Theta}\mathtt{L} \\ \beta\lceil \mathtt{db}(\gamma)\rceil : s'[y\backslash\lfloor \mathtt{db}(\gamma)\rfloor : \mathtt{C}_2]_{\Theta}\mathtt{L} \\ \beta\lceil \mathtt{db}(\gamma)\rceil : s'[y\backslash\lfloor \mathtt{db}(\gamma)\rfloor : u']_{\Theta}\mathtt{L}_1[z\backslash\mathtt{C}_2]_{\Psi}\mathtt{L}_2 \end{cases}$$

Having defined $\widehat{\mathtt{C}}$, we have:

$$\begin{array}{ccc} \mathtt{C}_1\langle q\rangle[x\backslash t']_{\Omega} = \mathtt{C}\langle\!\langle x^{\alpha}\rangle\!\rangle[x\backslash t']_{\Omega} & \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t')} & \mathtt{C}\langle \alpha\bullet : t'\rangle[x\backslash t']_{\Omega} \\ \Big\downarrow{\scriptstyle \mathtt{db}(\gamma)} & & \Big\downarrow{\scriptstyle \mathtt{db}(\gamma)} \\ \widehat{\mathtt{C}}\langle\!\langle x^{\alpha}\rangle\!\rangle[x\backslash t']_{\Omega} & \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t')} & \widehat{\mathtt{C}}\langle \alpha\bullet : t'\rangle[x\backslash t']_{\Omega} \end{array}$$

Note that since $\widehat{\mathtt{C}}$ is a pseudo-context, the expression $\widehat{\mathtt{C}}\langle x^{\alpha}\rangle$ could prepend additional labels to the label α decorating the variable node x. However, the residual of the step R has the same name as R, namely $\downarrow(\alpha)\bullet\uparrow(t')$, since $\downarrow(\delta\alpha) = \downarrow(\alpha)$ for any label δ.

1.2.2 *S is a $\mathtt{ls}$-redex, including x^{α} as a subterm.* Since the redex occurrence S includes x^{α} as a subterm, it cannot contract the same substitution as the redex occurrence R, for in that case they would be the same redex, substituting the same occurrence of x^{α}. Let q be the subterm corresponding to the $\mathtt{ls}$-redex S. Since q is a subterm of $\mathtt{C}\langle\!\langle x^{\alpha}\rangle\!\rangle$, we know $\mathtt{C}\langle\!\langle x^{\alpha}\rangle\!\rangle = \mathtt{C}_1\langle q\rangle$. Moreover, since q is a $\mathtt{ls}$-redex, it has the form: $q = \mathtt{C}'\langle\!\langle y^{\beta}\rangle\!\rangle[y\backslash s']_{\Theta}$ where $\nu = \downarrow(\beta)\bullet\uparrow(s')$. Here we are also assuming that the affected occurrence of x^{α} is internal to q. This leads to two cases for $\mathtt{C}$, depending on whether the hole corresponds to a position inside $\mathtt{C}'$ or inside s':

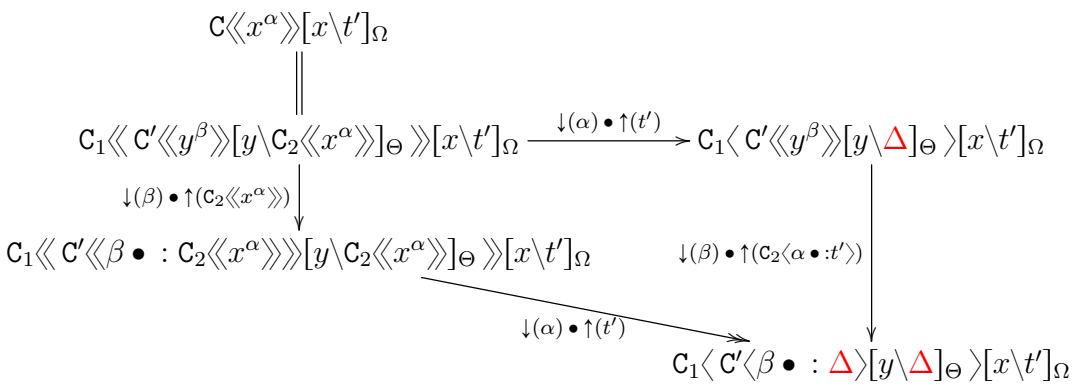
$$\mathtt{C} = \begin{cases} \mathtt{C}_1\langle\, \mathtt{C}_2\langle\!\langle \Box, y^{\beta}\rangle\!\rangle[y\backslash s']_{\Theta}\,\rangle & \text{where } \mathtt{C}_2 \text{ is a two-hole context} \\ & \text{such that } \mathtt{C}_2\langle\!\langle x^{\alpha}, \Box\rangle\!\rangle = \mathtt{C}' \\ & \textit{“the hole is internal to } \mathtt{C}'\textit{”} \\ \mathtt{C}_1\langle\, \mathtt{C}'\langle\!\langle y^{\beta}\rangle\!\rangle[y\backslash \mathtt{C}_2]_{\Theta}\,\rangle & \text{where } \mathtt{C}_2\langle\!\langle x^{\alpha}\rangle\!\rangle = s' \\ & \textit{“the hole is internal to } s'\textit{”} \end{cases}$$

We analyze these two subcases separately:

1.2.2.1 $\mathtt{C} = \mathtt{C}_1\langle\, \mathtt{C}_2\langle\!\langle \Box, y^{\beta}\rangle\!\rangle[y\backslash s']_{\Theta}\,\rangle$. This case is straightforward:

$$\begin{array}{ccc} & \mathtt{C}\langle\!\langle x^{\alpha}\rangle\!\rangle[x\backslash t']_{\Omega} & \\ & \| & \\ \mathtt{C}_1\langle\!\langle\, \mathtt{C}_2\langle\!\langle x^{\alpha}, y^{\beta}\rangle\!\rangle[y\backslash s']_{\Theta}\,\rangle\!\rangle[x\backslash t']_{\Omega} & \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t')} & \mathtt{C}_1\langle\!\langle\, \mathtt{C}_2\langle\!\langle \alpha\bullet : t', y^{\beta}\rangle\!\rangle[y\backslash s']_{\Theta}\,\rangle\!\rangle[x\backslash t']_{\Omega} \\ \Big\downarrow{\scriptstyle \downarrow(\beta)\bullet\uparrow(s')} & & \Big\downarrow{\scriptstyle \downarrow(\beta)\bullet\uparrow(s')} \\ \mathtt{C}_1\langle\!\langle\, \mathtt{C}_2\langle\!\langle x^{\alpha}, \beta\bullet : s'\rangle\!\rangle[y\backslash s']_{\Theta}\,\rangle\!\rangle[x\backslash t']_{\Omega} & \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t')} & \mathtt{C}_1\langle\!\langle\, \mathtt{C}_2\langle\!\langle \alpha\bullet : t', \beta\bullet : s'\rangle\!\rangle[y\backslash s']_{\Theta}\,\rangle\!\rangle[x\backslash t']_{\Omega} \end{array}$$

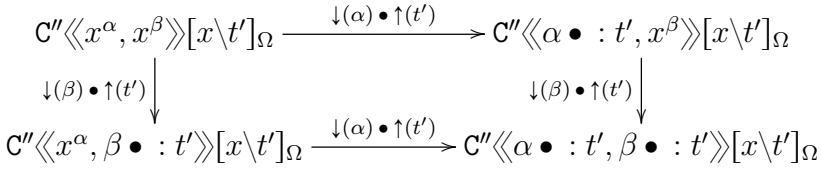
1.2.2.2 $\mathtt{C} = \mathtt{C}_1\langle\, \mathtt{C}'\langle\!\langle y^\beta \rangle\!\rangle[y\backslash \mathtt{C}_2]_\Theta \,\rangle$. Let us abbreviate $\Delta := \mathtt{C}_2\langle \alpha \bullet \,:\, t' \rangle$. Then:



To be able to close the diagram, we need the two following observations:

- By Lem. 6.9, $\uparrow(\mathtt{C}_2\langle\!\langle x^\alpha \rangle\!\rangle) = \uparrow(\mathtt{C}_2\langle\!\langle \alpha \bullet \,:\, t' \rangle\!\rangle)$.
- By Lem. 6.9: $\beta \bullet \,:\, \mathtt{C}_2\langle\!\langle x^\alpha \rangle\!\rangle$ is of the form $\mathtt{C}'_2\langle\!\langle x^{\alpha'} \rangle\!\rangle$, where $\downarrow(\alpha) = \downarrow(\alpha')$ and $\beta \bullet \,:\, \mathtt{C}_2\langle \alpha : t' \rangle = \mathtt{C}'_2\langle \alpha' : t' \rangle$.

1.2.3 *S is a* $\mathtt{ls}$*-redex, contracting* $[x\backslash t']_\Omega$. That is, $\mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle = \mathtt{C}'\langle\!\langle x^\beta \rangle\!\rangle$, where x^α is the occurrence affected by R, and x^β is the occurrence affected by S. Since the two occurrences are distinct, there is a two-hole context $\mathtt{C}''$ such that $\mathtt{C}''\langle\!\langle \Box, x^\beta \rangle\!\rangle = \mathtt{C} \quad \mathtt{C}''\langle\!\langle x^\alpha, \Box \rangle\!\rangle = \mathtt{C}'$. It is then immediate to close the diagram:

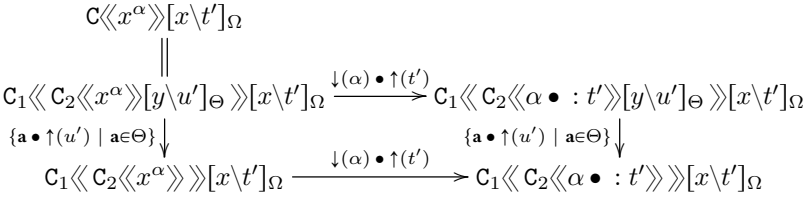


1.2.4 *S is a* $\mathtt{gc}$*-redex, including x^α as a subterm.* Let q be the subterm corresponding to the gc-redex. As in the previous cases, since q is a subterm of $\mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle$, we have that $\mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle = \mathtt{C}_1\langle q \rangle$. Moreover, q must have the form $q = s'[y\backslash u']_\Theta$, with $y \notin \mathtt{fv}(s')$. As we are also assuming that the affected occurrence of x^α is in q, there are two possibilities for $\mathtt{C}$, depending on whether the hole of $\mathtt{C}$ lies in a position inside s' or inside u':

$$\mathtt{C} = \begin{cases} \mathtt{C}_1\langle\, \mathtt{C}_2[y\backslash u']_\Theta \,\rangle & \text{where } \mathtt{C}_2\langle\!\langle x^\alpha \rangle\!\rangle = s' \\ & \textit{“the hole is internal to } s'\textit{”} \\ \mathtt{C}_1\langle\, s'[y\backslash \mathtt{C}_2]_\Theta \,\rangle & \text{where } \mathtt{C}_2\langle\!\langle x^\alpha \rangle\!\rangle = u' \\ & \textit{“the hole is internal to } u'\textit{”} \end{cases}$$

We analyze these two subcases separately:

1.2.4.1 $\mathtt{C} = \mathtt{C}_1\langle\, \mathtt{C}_2[y\backslash u']_\Theta \,\rangle$



1.2.4.2 $\mathtt{C} = \mathtt{C}_1\langle\, s'[y\backslash \mathtt{C}_2]_\Theta \,\rangle$

$$\begin{array}{ccc}
\mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle[x\backslash t']_\Omega & & \\
\| & & \\
\mathtt{C}_1\langle\!\langle\, s'[y\backslash \mathtt{C}_2\langle\!\langle x^\alpha\rangle\!\rangle]_\Theta \,\rangle\!\rangle[x\backslash t']_\Omega & \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t')} & \mathtt{C}_1\langle\!\langle\, s'[y\backslash \mathtt{C}_2\langle\!\langle \alpha\bullet : t'\rangle\!\rangle]_\Theta \,\rangle\!\rangle[x\backslash t']_\Omega \\
{\scriptstyle \{\mathbf{a}\bullet\uparrow(\mathtt{C}_2\langle\!\langle x^\alpha\rangle\!\rangle)\,\mid\,\mathbf{a}\in\Theta\}}\downarrow & & {\scriptstyle \{\mathbf{a}\bullet\uparrow(\mathtt{C}_2\langle\!\langle \alpha\bullet :t'\rangle\!\rangle)\,\mid\,\mathbf{a}\in\Theta\}}\downarrow \\
\mathtt{C}_1\langle\!\langle\, s'\,\rangle\!\rangle[x\backslash t']_\Omega & = & \mathtt{C}_1\langle\!\langle\, s'\,\rangle\!\rangle[x\backslash t']_\Omega
\end{array}$$

We conclude this subcase noting that $\uparrow(\mathtt{C}_2\langle\!\langle x^\alpha\rangle\!\rangle) = \uparrow(\mathtt{C}_2\langle\!\langle \alpha\bullet : t'\rangle\!\rangle)$ by Lem. 6.9.

1.3 R *is a* gc*-redex.* That is: $t = t'[x\backslash s']_\Omega \quad x \notin \mathtt{fv}(t')$ and $\mu = \{\mathbf{a}\bullet\uparrow(s') \mid \mathbf{a}\in\Omega\}$. If the redex occurrence S is internal to t', it is straightforward to close the diagram. If it is internal to s', it is also straightforward, taking in account the fact that the contraction of R erases S. More precisely, if we suppose that $s' \xrightarrow{\nu}_\ell \widehat{s}'$, we have:

$$\begin{array}{ccc}
t'[x\backslash s']_\Omega & \xrightarrow{\{\mathbf{a}\bullet\uparrow(s')\,\mid\,\mathbf{a}\in\Omega\}} & t' \\
{\scriptstyle\nu}\downarrow & & \| \\
t'[x\backslash \widehat{s}']_\Omega & \xrightarrow{\{\mathbf{a}\bullet\uparrow(\widehat{s}')\,\mid\,\mathbf{a}\in\Omega\}} & t'
\end{array}$$

Note that $\uparrow(s') = \uparrow(\widehat{s}')$ by the fact that reduction preserves the first label of a term (Lem. 6.11).

The remaining possibility is that the redex occurrence S involves the substitution at the root of t. This cannot happen:

- The redex occurrence S cannot be a db-redex, as there is no application node at the root.
- The redex occurrence S cannot be a ls-redex, since that would require at least one free occurrence of x in t'.
- If S is a gc-redex, then R and S are the same redex ocurrence, which is trivial and was already considered.

2. *Inductive case.* All the inductive cases are trivial, since we only care about the case when the position of R is a prefix of the position of S. Hence both redex occurrences R and S must be internal to the same subcontext of C, and we conclude by *i.h.*.

□

A.3.3 Postponement of gc in the LLSC-calculus — proof of Lem. 6.50

Lemma A.80 (Full proof of Lem. 6.50—Postponement of gc in the LLSC-calculus)**.** *Let* $\rho : t \twoheadrightarrow_\ell s$ *be a reduction sequence. Then there exists a term* u *and a reduction sequence* $\sigma : t \twoheadrightarrow_{\ell\,\mathtt{db}\cup\mathtt{ls}} u \twoheadrightarrow_{\ell\,\mathtt{gc}} s$.

Moreover, let $\#_\mu(\rho)$ *denote the number of redexes named* μ *that are contracted along the reduction sequence* ρ*. Then:*

1. *The number of* $\mathtt{db}$ *and* $\mathtt{ls}$ *redexes is preserved:*

$$\#_\mu(\rho) = \#_\mu(\sigma) \text{ if } \mu \text{ is the name of a } \mathtt{db} \text{ or } \mathtt{ls} \text{ redex}$$

2. *The number of* $\mathtt{gc}$ *redexes may increase:*

$$\#_\mu(\rho) \leqslant \#_\mu(\sigma) \text{ if } \mu \text{ is the name of a } \mathtt{gc} \text{ redex}$$

3. *The reduction* σ *contracts the same names as* ρ*:*

$$\#_\mu(\sigma) > 0 \implies \#_\mu(\rho) > 0 \text{ for any redex name } \mu$$

Proof. The proof is split in two parts:

1. First we show that any two steps $t_1 \xrightarrow{\mu}_{\ell\,\mathtt{gc}} t_2 \xrightarrow{\nu}_{\ell\,\mathtt{db}\cup\mathtt{ls}} t_3$ can be swapped in such a way that $t_1 \xrightarrow{\nu}_{\ell\,\mathtt{db}\cup\mathtt{ls}} t_2' \,(\xrightarrow{\mu}_{\ell\,\mathtt{gc}})^n\, t_3$, requiring $1 \leqslant n \leqslant 2$ steps of $\mathtt{gc}$ to close the diagram.

2. We then argue that this process terminates. It is easy to check that the conditions on $\#_\mu(\rho)$ are preserved by the swapping operation.

Swapping. Let $t_1 \xrightarrow{\mu}_{\ell\,\mathtt{gc}} t_2 \xrightarrow{\nu}_{\ell\,\mathtt{db}\cup\mathtt{ls}} t_3$. By induction on the context $\mathtt{C}$ under which the $\to_{\ell\,\mathtt{gc}}$ redex in t_1 is contracted, we show that the following diagram can be closed with $1 \leqslant n \leqslant 2$ steps of $\mathtt{gc}$:

$$\begin{array}{ccc} t_1 & \xrightarrow{\mu} & t_2 \\ \vdots_{\nu} & & \downarrow_{\nu} \\ \widehat{t_2} & \overset{\mu}{\twoheadrightarrow} & t_3 \end{array}$$

1. *Base case,* $\mathtt{C} = \Box$. The situation is:

$$\begin{array}{ccc} t_1 = t_2[x\backslash s]_\Omega & \xrightarrow{\{\mathbf{a}\bullet\uparrow(s)\ \mid\ \mathbf{a}\in\Omega\}} & t_2 \\ \vdots_{\nu} & & \downarrow_{\nu} \\ t_3[x\backslash s]_\Omega & \overset{\{\mathbf{a}\bullet\uparrow(s)\ \mid\ \mathbf{a}\in\Omega\}}{\twoheadrightarrow} & t_3 \end{array}$$

   Since we know $x \notin \mathtt{fv}(t_2)$, we use the fact that $t_2 \xrightarrow{\nu}_{\ell\ \mathtt{db}\cup\mathtt{ls}} t_3$ does not create free variables to conclude $x \notin \mathtt{fv}(t_3)$.

2. *Inductive case, under an abstraction,* $\mathtt{C} = \lambda^\alpha_\Omega x.\mathtt{C}'$. Direct by *i.h.*, since both steps must be internal to $\mathtt{C}'$.

3. *Inductive case, left of an application,* $\mathtt{C} = @^\alpha(\mathtt{C}', s)$. The situation is:

$$@^\alpha(t_1', s) \xrightarrow{\mu}_{\ell\,\mathtt{gc}} @^\alpha(t_2', s) \xrightarrow{\nu}_{\ell\,\mathtt{db}\cup\mathtt{ls}} t_3$$

   with $t_1' \xrightarrow{\mu}_{\ell\,\mathtt{gc}} t_2'$. There are three subcases, depending on whether the $\to_{\ell\,\mathtt{db}\cup\mathtt{ls}}$ step is internal to t_2', internal to s, or at the root.

3.1 $\to_{\ell\,\mathtt{db}\cup\mathtt{ls}}$ *step internal to* t_2'. Then t_3 is of the form $@^\alpha(t_3', s)$ and $t_1' \xrightarrow{\mu}_{\ell\,\mathtt{gc}} t_2' \xrightarrow{\nu}_{\ell\,\mathtt{db}\cup\mathtt{ls}} t_3'$. We conclude by *i.h.*:

$$\begin{array}{ccc} @^\alpha(t_1', s) & \xrightarrow{\mu} & @^\alpha(t_2', s) \\ \vdots{\scriptstyle \nu} & & \downarrow{\scriptstyle \nu} \\ @^\alpha(\widehat{t_2'}, s) & \overset{\mu}{\twoheadrightarrow} & @^\alpha(t_3', s) \end{array}$$

3.2 $\to_{\ell\,\mathtt{db}\cup\mathtt{ls}}$ *step internal to* s. Then t_3 is of the form $@^\alpha(t_2', s')$ with $t_1' \xrightarrow{\mu}_{\ell\,\mathtt{gc}} t_2'$ and $s \xrightarrow{\nu}_{\ell\,\mathtt{db}\cup\mathtt{ls}} s'$. The steps are disjoint and can be swapped trivially:

$$\begin{array}{ccc} @^\alpha(t_1', s) & \xrightarrow{\mu} & @^\alpha(t_2', s) \\ \vdots{\scriptstyle \nu} & & \downarrow{\scriptstyle \nu} \\ @^\alpha(t_1', s') & \overset{\mu}{\twoheadrightarrow} & @^\alpha(t_2', s') \end{array}$$

3.3 $\to_{\ell\,\mathtt{db}\cup\mathtt{ls}}$ *step at the root.* The $\to_{\ell\,\mathtt{db}\cup\mathtt{ls}}$ involves the topmost application, so it must be a $\mathtt{db}$ step. Then t_2' must be of the form $(\lambda^\beta_\Omega x.u')\mathrm{L}$. We have $\nu = \mathtt{db}(\beta)$, and the situation is as follows:

$$\begin{array}{ccc} @^\alpha(t_1', s') & \xrightarrow{\mu} & @^\alpha((\lambda^\beta_\Omega x.u')\mathrm{L}, s') \\ & & \downarrow{\scriptstyle \mathtt{db}(\beta)} \\ & & \alpha\lceil\mathtt{db}(\beta)\rceil : u'[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s']_\Omega\mathrm{L} \end{array}$$

where $t_1' \xrightarrow{\mu}_{\ell\,\mathtt{gc}} t_2' = (\lambda^\beta_\Omega x.u')\mathrm{L}$. Then t_1' must be of the form $(\lambda^\beta_\Omega x.u_0)\mathrm{L}_0$ and there are three possibilities:

- *The* $\mathtt{gc}$ *step is internal to* u_0. That is, $\mathrm{L} = \mathrm{L}_0$, and:

$$u_0 = \mathtt{C}\langle r_1'[y\backslash r_2']_\Theta\rangle \xrightarrow{\{\mathbf{a}\bullet\uparrow(r_2')\ \mid\ \mathbf{a}\in\Theta\}}_{\ell\,\mathtt{gc}} \mathtt{C}\langle r_1'\rangle = u'$$

  Then by Lem. 6.10, we have that

$$\alpha\lceil\mathtt{db}(\beta)\rceil : u_0 \xrightarrow{\{\mathbf{a}\bullet\uparrow(r_2')\ \mid\ \mathbf{a}\in\Theta\}}_{\ell\,\mathtt{gc}} \alpha\lceil\mathtt{db}(\beta)\rceil : u'$$

  And so we conclude:

$$\begin{array}{ccc} @^\alpha((\lambda^\beta_\Omega x.u_0)\mathrm{L}, s') & \xrightarrow{\{\mathbf{a}\bullet\uparrow(r_2')\ \mid\ \mathbf{a}\in\Theta\}} & @^\alpha((\lambda^\beta_\Omega x.u')\mathrm{L}, s') \\ \vdots{\scriptstyle \beta} & & \downarrow{\scriptstyle \beta} \\ \alpha\lceil\mathtt{db}(\beta)\rceil : u_0[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s']_\Omega\mathrm{L} & \overset{\{\mathbf{a}\bullet\uparrow(r_2')\ \mid\ \mathbf{a}\in\Theta\}}{\cdots\!\!\twoheadrightarrow} & \alpha\lceil\mathtt{db}(\beta)\rceil : u'[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s']_\Omega\mathrm{L} \end{array}$$

- *The* $\mathtt{gc}$ *step is internal to one of the arguments of* L_0. That is, $u' = u_0$ and:

$$\mathrm{L}_0 = \mathrm{L}_1[y\backslash r']_\Theta\mathrm{L}_2 \quad \mathrm{L} = \mathrm{L}_1[y\backslash r'']_\Theta\mathrm{L}_2$$

$$r' \xrightarrow{\mu}_{\ell\,\mathtt{gc}} r''$$

  Then:

$$\begin{array}{ccc} @^\alpha((\lambda^\beta_\Omega x.u')\mathrm{L}_0, s') & \xrightarrow{\mu} & @^\alpha((\lambda^\beta_\Omega x.u')\mathrm{L}, s') \\ \vdots{\scriptstyle \beta} & & \downarrow{\scriptstyle \beta} \\ \alpha\lceil\mathtt{db}(\beta)\rceil : u'[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s']_\Omega\mathrm{L}_0 & \overset{\mu}{\cdots\!\!\twoheadrightarrow} & \alpha\lceil\mathtt{db}(\beta)\rceil : u'[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s']_\Omega\mathrm{L} \end{array}$$

- *The* gc *step erases one of the substitutions in* $\mathtt{L}_0$. That is, $u' = u_0$ and:

$$\mathtt{L}_0 = \mathtt{L}_1[x\backslash r']_\Theta \mathtt{L}_2 \quad \mathtt{L} = \mathtt{L}_1\mathtt{L}_2$$
$$q\mathtt{L}_0 \xrightarrow{\{\mathbf{a}\bullet\uparrow(r') \mid \mathbf{a}\in\Theta\}}_{\ell\,\mathtt{gc}} q\mathtt{L} \quad \text{for any term } q$$

  The diagram is closed exactly as in the previous case, taking $\mu := \{\mathbf{a}\bullet\uparrow(r') \mid \mathbf{a}\in\Theta\}$.

4. *Inductive case, right of an application,* $\mathtt{C} = @^\alpha(s, \mathtt{C}')$. The situation is:

$$@^\alpha(s, t_1') \xrightarrow{\mu}_{\ell\,\mathtt{gc}} @^\alpha(s, t_2') \xrightarrow{\nu}_{\ell\,\mathtt{db}\cup\mathtt{ls}} t_3$$

   If the $\to_{\ell\ \mathtt{db}\cup\mathtt{ls}}$ step is internal to s or t_2', the situation is analogous to the *left of an application* case (points 3.1 and 3.2 of this lemma). The non-trivial case is when there is a $\to_{\ell\,\mathtt{db}}$ step at the root. That is:

$$s = (\lambda^\beta_\Omega x.s')\mathtt{L} \quad \nu = \beta$$

   By Lem. 6.10, since we had a step $t_1' \xrightarrow{\mu}_{\ell\ \mathtt{gc}} t_2'$, we also have a step $\lfloor\mathtt{db}(\beta)\rfloor : t_1' \xrightarrow{\mu}_{\ell\ \mathtt{gc}} \lfloor\mathtt{db}(\beta)\rfloor : t_2'$. Then:

$$\begin{array}{ccc} @^\alpha((\lambda^\beta_\Omega x.s')\mathtt{L}, t_1') & \xrightarrow{\quad\mu\quad} & @^\alpha((\lambda^\beta_\Omega x.s')\mathtt{L}, t_2') \\ \vdots\,\beta & & \downarrow\beta \\ \alpha\lceil\mathtt{db}(\beta)\rceil : s'[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : t_1']_\Omega\mathtt{L} & \xrightarrow{\mu} & \alpha\lceil\mathtt{db}(\beta)\rceil : s'[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : t_2']_\Omega\mathtt{L} \end{array}$$

5. *Inductive case, left of a substitution,* $\mathtt{C} = \mathtt{C}'[x\backslash s]_\Omega$. The situation is:

$$t_1'[x\backslash s]_\Omega \xrightarrow{\mu}_{\ell\,\mathtt{gc}} t_2'[x\backslash s]_\Omega \xrightarrow{\nu}_{\ell\,\mathtt{db}\cup\mathtt{ls}} t_3$$

   If the $\to_{\ell\ \mathtt{db}\cup\mathtt{ls}}$ step is internal to t_2' or s, the situation is analogous to the *left of an application* case (points 3.1 and 3.2 of this lemma). The non-trivial case is when there is a $\to_{\ell\,\mathtt{ls}}$ step at the root. Then t_2' must be of the form $\mathtt{C}\langle\!\langle x^\alpha\rangle\!\rangle$ and:

$$t_2'[x\backslash s]_\Omega = \mathtt{C}\langle\!\langle x^\alpha\rangle\!\rangle[x\backslash s]_\Omega \xrightarrow{\downarrow(\alpha)\bullet\uparrow(s)}_{\ell\,\mathtt{ls}} \mathtt{C}\langle\alpha\bullet : s\rangle[x\backslash s]_\Omega = t_3$$

   Moreover, since $t_1' \xrightarrow{\mu}_{\ell\ \mathtt{gc}} t_2'$, we have that t_1' must be of the form $\mathtt{C}'\langle u'[y\backslash r']_\Theta\rangle$, with $y \notin \mathtt{fv}(u')$, so that:

$$t_1' = \mathtt{C}'\langle u'[y\backslash r']_\Theta\rangle \xrightarrow{\{\mathbf{a}\bullet\uparrow(r') \mid \mathbf{a}\in\Theta\}}_{\ell\,\mathtt{gc}} \mathtt{C}'\langle u'\rangle = t_2' = \mathtt{C}\langle\!\langle x^\alpha\rangle\!\rangle$$

   This implies that the substituted occurrence of x^α in t_2' lies either in $\mathtt{C}'$ or in u'. Let $\widehat{\mathtt{C}}'$ and $\widehat{u}'$ denote the result of replacing the affected occurrence of x^α (if any) by $(\alpha\bullet : s)$, in $\mathtt{C}'$ and u' respectively. Then:

$$\begin{array}{ccccc} t_1'[x\backslash s]_\Omega & & t_2'[x\backslash s]_\Omega & & \\ \| & & \| & & \\ \mathtt{C}'\langle u'[y\backslash r']_\Theta\rangle[x\backslash s]_\Omega & \xrightarrow{\{\mathbf{a}\bullet\uparrow(r') \mid \mathbf{a}\in\Theta\}} & \mathtt{C}'\langle u'\rangle[x\backslash s]_\Omega & = & \mathtt{C}\langle\!\langle x^\alpha\rangle\!\rangle[x\backslash s]_\Omega \\ \vdots\,\downarrow(\alpha)\bullet\uparrow(s) & & & & \downarrow\downarrow(\alpha)\bullet\uparrow(s) \\ \widehat{\mathtt{C}}'\langle\widehat{u}'[y\backslash r']_\Theta\rangle[x\backslash s]_\Omega & \xrightarrow{\{\mathbf{a}\bullet\uparrow(r') \mid \mathbf{a}\in\Theta\}} & \widehat{\mathtt{C}}'\langle\widehat{u}'\rangle[x\backslash s]_\Omega & = & \mathtt{C}\langle\alpha\bullet : s\rangle[x\backslash s]_\Omega \end{array}$$

   We use the fact that $y \notin \mathtt{fv}(u')$ and $y \notin \mathtt{fv}(s)$ to conclude that $y \notin \mathtt{fv}(\widehat{u}')$, and thus be able to apply the gc step on the bottom of the diagram.

6. *Inductive case, right of a substitution,* $\mathtt{C} = s[x\backslash \mathtt{C}']_\Omega$. The situation is:

$$s[x\backslash t_1']_\Omega \xrightarrow{\mu}_{\ell\,\mathtt{gc}} s[x\backslash t_2']_\Omega \xrightarrow{\nu}_{\ell\,\mathtt{db}\cup\mathtt{ls}} t_3$$

If the $\to_{\ell\ \mathtt{db}\cup\mathtt{ls}}$ step is internal to s or t_2', the situation is analogous to the *left of an application* case (points 3.1 and 3.2 of this lemma). The non-trivial case is when there is a $\to_{\ell\,\mathtt{ls}}$ step at the root. Then $s = \mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle$ and:

$$\begin{array}{ccc} \mathtt{C}\langle\!\langle x^\alpha\rangle\!\rangle[x\backslash t_1']_\Omega & \xrightarrow{\qquad\qquad\mu\qquad\qquad} & \mathtt{C}\langle\!\langle x^\alpha\rangle\!\rangle[x\backslash t_2']_\Omega \\ \Big\downarrow{\scriptstyle \downarrow(\alpha)\bullet\uparrow(t_1')} & & \Big\downarrow{\scriptstyle \downarrow(\alpha)\bullet\uparrow(t_2')} \\ \mathtt{C}\langle\!\langle \alpha\bullet : t_1'\rangle\!\rangle[x\backslash t_1']_\Omega & \xrightarrow{\mu} \mathtt{C}\langle\!\langle \alpha\bullet : t_2'\rangle\!\rangle[x\backslash t_1']_\Omega & \xrightarrow{\mu} \mathtt{C}\langle\!\langle \alpha\bullet : t_2'\rangle\!\rangle[x\backslash t_2']_\Omega \end{array}$$

We use the fact that reduction preserves the first label (Lem. 6.11) to conclude that since $t_1' \xrightarrow{\mu}_{\ell\,\mathtt{gc}} t_2'$ then $\uparrow(t_1') = \uparrow(t_2')$; this is to ensure that the `ls` steps both have the same name. Moreover, we use the fact that adding labels preserves redex names (Lem. 6.10) to conclude that $\alpha\bullet : t_1' \xrightarrow{\mu}_{\ell\,\mathtt{gc}} \alpha\bullet : t_2'$; this is to ensure that the first `gc` step at the bottom of the diagram has the right name.

Finally, note that this is the only case throughout the proof by induction that duplicates the number of `gc` steps that are required to close the diagram. All the other cases either resort to the inductive hypothesis or require exactly one `gc` step. Since this case corresponds to having a `ls` step at the root, it can be applied at most once, implying that the number n of `gc` steps that are required to close the diagram must be $1 \leqslant n \leqslant 2$.

Termination. The process of swapping `gc` and `db` $\cup$ `ls` steps is modeled by the string rewriting system:

$$\mathcal{R} : \begin{cases} \mathbf{b}\,\mathbf{a} \to \mathbf{a}\,\mathbf{b} \\ \mathbf{b}\,\mathbf{a} \to \mathbf{a}\,\mathbf{b}\,\mathbf{b} \end{cases}$$

where $\mathbf{a}$ represents `db` $\cup$ `ls` steps and $\mathbf{b}$ represents `gc` steps. To see that $\mathcal{R}$ is strongly normalizing consider the decreasing measure $m : \{\mathbf{a},\mathbf{b}\}^\star \to \mathbb{N}$ given by:

$$m(s) := \sum_{i, s_i = \mathbf{b}} 3^{\#\{j > i \mid s_j = \mathbf{a}\}}$$

□

A.4 Proofs of Chapter 7 – Applications of the Labeled Linear Substitution Calculus

A.4.1 Contribution — auxiliary lemmas for Prop. 7.12

In this section we state and prove auxiliary results that are needed to prove that the LSC without `gc` verifies the Contribution axiom. Note that, even though Prop. 7.12 is about the LSC without `gc`, the LLSC (with labels) is used as an auxiliary tool, so most lemmas in this

section concerns the labeled version of the LSC. Since the gc rule is never used here, in order to alleviate notation, we omit the sets of labels on abstractions and substitutions, which are irrelevant in this context. That is, we write $\lambda^\alpha x.t$ and $t[x\backslash s]$ rather than $\lambda^\alpha_\Omega x.t$ and $t[x\backslash s]_\Omega$, omitting the "Ω" subscripts.

Correctness

The following definitions and lemmas are used to prove the implication $(1 \implies 2)$ (**Correctness**) of Prop. 7.12.

Definition A.81 (Inclusion of labels)**.** The order relation of *inclusion* between labels α, β, written $\alpha \subseteq \beta$, is given by the reflexive and transitive closure of the following rules:

$$\alpha \subseteq \lceil\alpha\rceil \quad \alpha \subseteq \lfloor\alpha\rfloor \quad \alpha \subseteq \mathtt{db}(\alpha) \quad \alpha \subseteq \alpha\beta \quad \alpha \subseteq \beta\alpha$$

Definition A.82 (All labels in a term)**.** Given a labeled term $t \in \mathcal{T}^\ell$, the set of all labels decorating nodes in t is written $labels(t)$. Formally:

$$\begin{array}{rcl} labels(x^\alpha) & \stackrel{\text{def}}{=} & \{\alpha\} \\ labels(@^\alpha(t,s)) & \stackrel{\text{def}}{=} & \{\alpha\} \cup labels(t) \cup labels(s) \\ labels(\lambda^\alpha x.t) & \stackrel{\text{def}}{=} & \{\alpha\} \cup labels(t) \\ labels(t[x\backslash s]) & \stackrel{\text{def}}{=} & labels(t) \cup labels(s) \end{array}$$

This definition is also extended to contexts, by setting $labels(\Box) \stackrel{\text{def}}{=} \varnothing$.

Lemma A.83 (Redex names that contribute to a step must occur in the source)**.** *Let $R^\ell : t^\ell_0 \to t^\ell_1$ be a step in the LLSC without* gc*. Let ν be the name of R^ℓ, and let μ be another redex name such that $\mu \overset{\text{Name}}{\hookrightarrow} \nu$. Then there exists a label $\alpha \in labels(t^\ell_0)$ such that $\mu \subseteq \alpha$.*

Proof. It is easy to check that name contribution implies label inclusion, so from the hypothesis $\mu \overset{\text{Name}}{\hookrightarrow} \nu$ we have that $\mu \subseteq \nu$. Moreover, the inclusion is proper, *i.e.* $\mu \neq \nu$ since, by definition, $\mu \overset{\text{Name}}{\hookrightarrow} \mu$ does not hold. We proceed by case analysis on the kind of step R^ℓ:

1. db *step.* Then we have:

$$t^\ell_0 = \mathtt{C}\langle @^\alpha((\lambda^\beta x.t^\ell)\mathtt{L}, s^\ell)\rangle \xrightarrow{\mathtt{db}(\beta)}_\ell \mathtt{C}\langle \alpha\lceil\mathtt{db}(\beta)\rceil : t^\ell[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s^\ell]\mathtt{L}\rangle = t^\ell_1$$

   Since $\mu \subseteq \nu = \mathtt{db}(\beta)$ and $\mu \neq \nu$, it must be the case that $\mu \subseteq \beta$ and $\beta \in labels(t^\ell_0)$, so we are done.

2. ls *step.* Then we have:

$$t^\ell_0 = \mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x^\alpha \rangle\!\rangle[x\backslash t^\ell]\rangle \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t^\ell)}_\ell \mathtt{C}_1\langle \mathtt{C}_2\langle \alpha : t^\ell\rangle[x\backslash t^\ell]\rangle = t^\ell_1$$

   Observe that $\mu \subseteq \nu = \downarrow(\alpha) \bullet \uparrow(t^\ell)$ and that $\downarrow(\alpha)$ and $\uparrow(t^\ell)$ are atomic labels. We claim that either $\mu \subseteq \downarrow(\alpha)$ or $\mu \subseteq \uparrow(t^\ell)$. Indeed:

2.1 If μ is the name of a db redex, then it is of the form $\mathtt{db}(\gamma)$ so necessarily $\mathtt{db}(\gamma) \subseteq \downarrow(\alpha)$ or $\mathtt{db}(\gamma) \subseteq \uparrow(t^\ell)$.

2.2 If μ is the name of a ls redex, then it is of the form $\gamma \bullet \delta$ where γ and δ are atomic labels. Since we already know that $\mu \neq \nu$, the " $\bullet$ " in μ must occur either in $\downarrow(\alpha)$ or in $\uparrow(t^\ell)$. This in turn implies that $\gamma \bullet \delta \subseteq \downarrow(\alpha)$ or $\gamma \bullet \delta \subseteq \uparrow(t^\ell)$.

Now there are two possibilities: if $\mu \subseteq \downarrow(\alpha)$ then $\mu \subseteq \alpha \in labels(t_0^\ell)$, and we are done. Otherwise, it must be the case that $\mu \subseteq \uparrow(t^\ell)$. Let us write the term t^ℓ as of the form $t^\ell = s^\ell \mathtt{L}$ where L is a list of substitutions and s^ℓ is a term whose root is not a substitution node (*i.e.* it is an application, an abstraction or a variable). Then by definition $\uparrow(t^\ell) = \uparrow(\beta)$ where β is the label decorating s^ℓ. Thus we obtain that $\mu \subseteq \uparrow(t^\ell) = \uparrow(\beta) \subseteq \beta \in labels(t^\ell)$, as required.

□

Definition A.84 (Labels of variables)**.** Let $vl(t^\ell)$ be defined as the following set:

$$vl(t^\ell) = \{\downarrow(\alpha) \mid x^\alpha \text{ is a subterm of } t^\ell \text{ for some } x\}$$

Inductively:

$$\begin{array}{lcl} vl(x^\alpha) & \stackrel{\text{def}}{=} & \{\downarrow(\alpha)\} \\ vl(\lambda^\alpha x.t^\ell) & \stackrel{\text{def}}{=} & vl(t^\ell) \\ vl(@^\alpha(t^\ell, s^\ell)) & \stackrel{\text{def}}{=} & vl(t^\ell) \cup vl(s^\ell) \\ vl(t^\ell[x\backslash s^\ell]) & \stackrel{\text{def}}{=} & vl(t^\ell) \cup vl(s^\ell) \end{array}$$

This definition is also extended to contexts by setting $vl(\Box) = \varnothing$.

Remark A.85. $vl(\mathtt{C}\langle t^\ell\rangle) = vl(\mathtt{C}) \cup vl(t^\ell)$

Remark A.86. $vl(\alpha : t^\ell) = vl(t^\ell)$

Lemma A.87 (Labels of variables are not created)**.** *Let $t_0^\ell \to_\ell t_1^\ell$ be a step in the labeled calculus. Then $vl(t_0^\ell) \supseteq vl(t_1^\ell)$.*

Proof. Straightforward by case analysis on the kind of contracted redex. □

Definition A.88 (Labels of substitutions)**.** Let $sl(t^\ell)$ be defined as the following set:

$$sl(t^\ell) = \{\uparrow(s^\ell) \mid [x\backslash s^\ell] \text{ is a substitution occurring in } t^\ell \text{ for some } x\}$$

Inductively:

$$\begin{array}{rcl} sl(x^\alpha) & \stackrel{\text{def}}{=} & \varnothing \\ sl(\lambda^\alpha x.t^\ell) & \stackrel{\text{def}}{=} & sl(t^\ell) \\ sl(@^\alpha(t^\ell, s^\ell)) & \stackrel{\text{def}}{=} & sl(t^\ell) \cup sl(s^\ell) \\ sl(t^\ell[x\backslash s^\ell]) & \stackrel{\text{def}}{=} & \{\uparrow(s^\ell)\} \cup sl(t^\ell) \cup sl(s^\ell) \end{array}$$

This definition is also extended to contexts by setting $sl(\Box) = \varnothing$.

Remark A.89. For any context $\mathtt{C}$ and any term t^ℓ we have:

$$sl(\mathtt{C}\langle t^\ell\rangle) = sl(\mathtt{C}) \cup sl(t^\ell) \cup \begin{cases} \{\uparrow (t^\ell)\} & \text{if } \mathtt{C} \text{ is of the form } \mathtt{C}'\langle s^\ell[x\backslash\Box\mathtt{L}]\rangle \\ \varnothing & \text{otherwise} \end{cases}$$

Remark A.90. $sl(\alpha : t^\ell) = sl(t^\ell)$

Lemma A.91 (Creation of labels of substitutions)**.** *Let $t_0^\ell \to_\ell t_1^\ell$ be a step in the labeled calculus. Then:*

1. *If it is a* $\mathtt{db}$ *step of name* $\mathtt{db}(\beta)$*, then* $sl(t_0^\ell) \cup \{\lfloor \mathtt{db}(\beta)\rfloor\} = sl(t_1^\ell)$.
2. *If it is a* $\mathtt{ls}$ *step, then* $sl(t_0^\ell) = sl(t_1^\ell)$.

Proof. Straightforward by case analysis on the kind of contracted redex. □

Lemma A.92 (Possible shapes of redex names)**.** *Let $\rho^\ell : t_0^\ell \twoheadrightarrow_\ell t_1^\ell$ be a derivation in the LLSC without* $\mathtt{gc}$*, where t_0^ℓ is an initially labeled term. Let μ be the name of a redex contracted along ρ^ℓ. Then μ must have one of the following three forms:*

$$\mathtt{db}(\alpha) \qquad \mathbf{a} \bullet \mathbf{b} \qquad \mathbf{a} \bullet \lfloor \mathtt{db}(\alpha)\rfloor$$

Proof. We claim that if $t_0^\ell \twoheadrightarrow_\ell u^\ell$ and t_0^ℓ is an initially labeled term, then the following properties hold for u^ℓ:

(I1) If x^α is a subterm of u^ℓ, *i.e.* $\downarrow(\alpha) \in vl(u^\ell)$, then $\downarrow(\alpha)$ is an initial label $\mathbf{a}$.

(I2) If $[x\backslash t^\ell]$ is a substitution occurring in u^ℓ, *i.e.* $\uparrow(t^\ell) \in sl(u^\ell)$, then $\uparrow(t^\ell)$ is an initial label $\mathbf{a}$ or a label of the form $\lfloor \mathtt{db}(\alpha)\rfloor$.

By induction on the length of the derivation $t_0^\ell \twoheadrightarrow_\ell u^\ell$, it can be checked that this invariant is preserved More precisely, let $s_0^\ell \to_\ell s_1^\ell$ be a labeled step and suppose that the invariant holds for s_1^ℓ. By the fact that variable labels are not created, as shown in Lem. A.87, condition **(I1)** is preserved. By the fact that substitution labels are only created with the form $\lfloor \mathtt{db}(\alpha)\rfloor$, as shown in Lem. A.91, condition **(I2)** is preserved.
Moreover, if the invariant holds for a term s_0^ℓ, the name of any labeled step $s_0^\ell \to_\ell s_1^\ell$ has one of the forms in the statement. Indeed, by case analysis on the kind of step taken:

1. $\mathtt{db}$ *step.* Then the name of the step is $\mathtt{db}(\beta)$ and it has the first of the forms in the statement.
2. $\mathtt{ls}$ *step.* Then the step is of the form:

$$s_0^\ell = \mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x^\alpha \rangle\!\rangle[x\backslash t^\ell]\rangle \xrightarrow{\downarrow(\alpha)\bullet\uparrow(t^\ell)}_\ell \mathtt{C}_1\langle \mathtt{C}_2\langle \alpha \bullet : t^\ell\rangle[x\backslash t^\ell]\rangle = s_1^\ell$$

The name of the step is $\downarrow(\alpha) \bullet \uparrow(t^\ell)$. Since the invariant holds for s_0^ℓ, condition **(I1)** ensures that $\downarrow(\alpha)$ is an initial label and condition **(I2)** ensures that $\uparrow(t^\ell)$ is either an initial label or of the form $\lfloor \mathtt{db}(\alpha)\rfloor$. Hence the name of the step has either the second form or the third form in the statement.

□

Lemma A.93 (Set of labels after a step)**.** *Let $R^\ell : t_0^\ell \to_\ell t_1^\ell$ be a step in the LLSC without* gc*, let μ be the name of R^ℓ, and let $\alpha \in labels(t_1^\ell)$. Then at least one of the following conditions hold:*

(I) $\alpha \in labels(t_0^\ell)$

(II) $\mu = \mathtt{db}(\beta)$ *and* $\alpha = \gamma\lceil\mathtt{db}(\beta)\rceil\delta$ *with* $\gamma, \beta, \delta \in labels(t_0^\ell)$

(III) $\mu = \mathtt{db}(\beta)$ *and* $\alpha = \lfloor\mathtt{db}(\beta)\rfloor\gamma$ *with* $\beta, \gamma \in labels(t_0^\ell)$

(IV) $\mu = \downarrow(\beta) \bullet \uparrow(\gamma)$ *and* $\alpha = \beta \bullet \gamma$ *with* $\beta, \gamma \in labels(t_0^\ell)$

Proof. By case analysis on the kind of redex R^ℓ:

1. `db` *redex.* Then $\mu = \mathtt{db}(\beta)$ and we have:

$$t_0^\ell = \mathtt{C}\langle @^\gamma((\lambda^\beta x.t^\ell)\mathtt{L}, s^\ell)\rangle \xrightarrow{\mathtt{db}(\beta)}_\ell \mathtt{C}\langle\gamma\lceil\mathtt{db}(\beta)\rceil : t^\ell[x\backslash\lfloor\mathtt{db}(\beta)\rfloor : s^\ell]\mathtt{L}\rangle = t_1^\ell$$

   Let $\alpha \in labels(t_1^\ell)$. We consider the following cases, depending on the position where α occurs in t_1^ℓ:

   1.1 *Internal to* `C`, i.e. $\alpha \in labels(\mathtt{C})$. Then $\alpha \in labels(t_0^\ell)$ and we are in the situation (I).

   1.2 *Internal to* `L`, i.e. $\alpha \in labels(\mathtt{L})$. Then $\alpha \in labels(t_0^\ell)$ and we are in the situation (I).

   1.3 *Internal to* $(\gamma\lceil\mathtt{db}(\beta)\rceil : t^\ell)$, i.e. $\alpha \in labels(\gamma\lceil\mathtt{db}(\beta)\rceil : t^\ell)$. Let δ be the external label of t^ℓ, *i.e.* the one decorating the outermost node of t^ℓ which is not a substitution. More precisely, let $\delta = \ell(t^\ell)$. Note that:

$$labels(\gamma\lceil\mathtt{db}(\beta)\rceil : t^\ell) = labels(t^\ell)\backslash\{\delta\} \cup \{\gamma\lceil\mathtt{db}(\beta)\rceil\delta\}$$

   So either $\alpha \in labels(t^\ell) \subseteq labels(t_0^\ell)$ and we are in the situation (I), or $\alpha = \gamma\lceil\mathtt{db}(\beta)\rceil\delta$ and we are in the situation (II) since $\gamma, \beta, \delta \in labels(t_0^\ell)$.

   1.4 *Internal to* $(\lfloor\mathtt{db}(\beta)\rfloor : s^\ell)$, i.e. $\alpha \in labels(\lfloor\mathtt{db}(\beta)\rfloor : s^\ell)$. Let γ be the external label of s^ℓ, *i.e.* the one decorating the outermost node of s^ℓ which is not a substitution. More precisely, let $\delta = \ell(s^\ell)$. Note that:

$$labels(\lfloor\mathtt{db}(\beta)\rfloor : s^\ell) = labels(s^\ell)\backslash\{\delta\} \cup \{\lfloor\mathtt{db}(\beta)\rfloor\delta\}$$

   So either $\alpha \in labels(s^\ell) \subseteq labels(t_0^\ell)$ and we are in the situation (I), or $\alpha = \lfloor\mathtt{db}(\beta)\rfloor\delta$ and we are in the situation (III) since $\beta, \delta \in labels(t_0^\ell)$.

2. `ls` *redex.* Then $\mu = \downarrow(\beta) \bullet \uparrow(\gamma)$ and we have:

$$t_0^\ell = \mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle x^\beta\rangle\!\rangle[x\backslash t^\ell]\rangle \xrightarrow{\downarrow(\beta)\bullet\uparrow(\gamma)}_\ell \mathtt{C}_1\langle\mathtt{C}_2\langle\beta\bullet : t^\ell\rangle[x\backslash t^\ell]\rangle = t_1^\ell$$

   where γ is the external label of t^ℓ, *i.e.* the label decorating the outermost node of t^ℓ which is not a substitution, that is $\gamma = \ell(t^\ell)$. Let $\alpha \in labels(t_1^\ell)$. We consider the following cases, depending on the position where α occurs in t_1^ℓ:

2.1 *Internal to* $\mathtt{C}_1$, i.e. $\alpha \in labels(\mathtt{C}_1)$. Then $\alpha \in labels(t_0^\ell)$ and we are in the situation (I).

2.2 *Internal to* $\mathtt{C}_2$, i.e. $\alpha \in labels(\mathtt{C}_2)$. Then $\alpha \in labels(t_0^\ell)$ and we are in the situation (I).

2.3 *Internal to* t^ℓ, i.e. $\alpha \in labels(t^\ell)$. Then $\alpha \in labels(t_0^\ell)$ and we are in the situation (I).

2.4 *Internal to* $(\beta \bullet : t^\ell)$, i.e. $\alpha \in labels(\beta \bullet : t^\ell)$. Since γ is the most external label of t^ℓ we have that:

$$labels(\beta \bullet : t^\ell) = labels(t^\ell) \backslash \{\gamma\} \cup \{\beta \bullet \gamma\}$$

So either $\alpha \in labels(t^\ell) \subseteq labels(t_0^\ell)$ and we are in the situation (I), or $\alpha = \beta \bullet \gamma$ and we are in the situation (IV), since $\beta, \gamma \in labels(t_0^\ell)$.

□

Lemma A.94 (Redex names in a term result from contracting a redex of that name)**.** *Let* $\rho^\ell : t_0^\ell \twoheadrightarrow_\ell t_1^\ell$ *be a derivation in the LLSC without* gc*, where* t_0^ℓ *is an initially labelled term. Let* μ *be a redex name such that* $\mu \subseteq \alpha$ *for some label* $\alpha \in labels(t_1^\ell)$*. Then* ρ^ℓ *has a step whose name is* μ*,* i.e. ρ^ℓ *can be written as of the form* $\rho_1^\ell R^\ell \rho_2^\ell$*, where the name of* R^ℓ *is* μ*.*

Proof. By induction on the length of ρ^ℓ:

1. *Base case,* i.e. ρ^ℓ *empty.* We claim that this case is impossible. By hypothesis, there exists a label α such that $\mu \subseteq \alpha \in labels(t_0^\ell)$. Since t_0^ℓ is an initially labelled term, the label α must be an initial label, *i.e.* $\alpha = \mathbf{a}$. Then we have that $\mu \subseteq \mathbf{a}$. If μ is the name of a $\mathtt{db}$ redex, we have that $\mu = \mathtt{db}(\beta) \subseteq \mathbf{a}$, which is a contradiction. Similarly, if μ is the name of a $\mathtt{ls}$ redex, we have that $\mu = \beta \bullet \gamma \subseteq \mathbf{a}$, which is also not possible.

2. *Induction,* i.e. $\rho^\ell = \sigma^\ell S^\ell$. Let $s^\ell = \mathrm{tgt}(\sigma^\ell) = \mathrm{src}(S^\ell)$ and let ν be the name of σ^ℓ. Since $\alpha \in labels(t_1^\ell)$, by Lem. A.93 at least one of the following cases applies:

   (I) *If* $\alpha \in labels(s^\ell)$. Then by *i.h.* σ^ℓ can be written as of the form $\sigma^\ell = \sigma_1^\ell R^\ell \sigma_2^\ell$ where the name of R^ℓ is μ, so $\rho^\ell = \sigma_1^\ell R^\ell \sigma_2^\ell S^\ell$ and we conclude.

   (II) *If* $\nu = \mathtt{db}(\beta)$ *with* $\alpha = \gamma\lceil\mathtt{db}(\beta)\rceil\delta$ *and* $\gamma, \beta, \delta \in labels(s^\ell)$. Since $\mu \subseteq \alpha = \gamma\lceil\mathtt{db}(\beta)\rceil\delta$ we consider three posibilities, depending on the position where μ occurs in α:

      2.0.1 *If* $\mu \subseteq \gamma$ *or* $\mu \subseteq \beta$ *or* $\mu \subseteq \delta$. Then since $\gamma, \beta, \delta \in labels(s^\ell)$, we may apply the *i.h.* to obtain that σ^ℓ can be written as of the form $\sigma^\ell = \sigma_1^\ell R^\ell \sigma_2^\ell$ where the name of R^ℓ is μ, so we conclude.

      2.0.2 *If* $\mu = \mathtt{db}(\beta)$. Then the name of the step S^ℓ is $\nu = \mu$, so we conclude.

      2.0.3 *Otherwise.* Note that μ cannot be of the form $\lceil\mathtt{db}(\beta)\rceil$, because μ is the name of a redex. Hence the only remaining possibility is that μ overlaps at least one of the two boundaries in the expression $\gamma\lceil\mathtt{db}(\beta)\rceil\delta$. By "overlapping one of the two boundaries" we mean that μ is a label of the form $\mu = \varepsilon\zeta$ where either:

$$\varepsilon \text{ is a suffix of } \gamma \text{ and } \zeta \text{ is a prefix of } \lceil\mathtt{db}(\beta)\rceil\delta$$

or

$$\varepsilon \text{ is a suffix of } \gamma\lceil \mathtt{db}(\beta)\rceil \text{ and } \zeta \text{ is a prefix of } \delta.$$

Note that the label μ cannot be the name of a `db` redex, and it must be the name of an `ls` redex. This leaves us with only two possibilities, namely that $\mu = \gamma' \bullet \lceil \mathtt{db}(\beta)\rceil$ or that $\mu = \lceil \mathtt{db}(\beta)\rceil \bullet \delta'$. This contradicts the fact that the shape of the name of a `ls` step, when starting from an initially labeled term, is either of the form $\mathbf{a} \bullet \mathbf{b}$ or of the form $\mathbf{a} \bullet \lfloor \mathtt{db}(\beta')\rfloor$, as has been shown in Lem. A.92.

(III) *If* $\nu = \mathtt{db}(\beta)$ *with* $\alpha = \lfloor \mathtt{db}(\beta)\rfloor\gamma$ *and* $\beta, \gamma \in labels(s^\ell)$. Since $\mu \subseteq \alpha = \lfloor \mathtt{db}(\beta)\rfloor\gamma$ we consider three possibilities, depending on the position where μ occurs in α:

2.0.1 *If* $\mu \subseteq \beta$ *or* $\mu \subseteq \gamma$. Then since $\beta, \gamma \in labels(s^\ell)$, we may apply the *i.h.* to obtain that σ^ℓ can be written as of the form $\sigma^\ell = \sigma_1^\ell R^\ell \sigma_2^\ell$ where the name of R^ℓ is μ, so we conclude.

2.0.2 *If* $\mu = \mathtt{db}(\beta)$. Then the name of the step S^ℓ is $\nu = \mu$, so we conclude.

2.0.3 *Otherwise.* Note that μ cannot be equal to $\lfloor \mathtt{db}(\beta)\rfloor$ since μ is the name of a redex. So μ must necessarily overlap $\lfloor \mathtt{db}(\beta)\rfloor$ and γ. This implies that μ cannot possibly be the name of a `db` redex, and it must be the name of a `ls` redex. In particular, μ must be of the form $\lfloor \mathtt{db}(\beta)\rfloor \circ \gamma'$. This contradicts the fact that the shape of the name of a `ls` step, when starting from an initially labeled term, is either of the form $\mathbf{a} \bullet \mathbf{b}$ or of the form $\mathbf{a} \bullet \lfloor \mathtt{db}(\beta')\rfloor$, as has been shown in Lem. A.92.

(IV) *If* $\nu = \downarrow(\beta) \bullet \uparrow(\gamma)$ *with* $\alpha = \beta \bullet \gamma$ *and* $\beta, \gamma \in labels(s^\ell)$. Since $\mu \subseteq \alpha = \beta \bullet \gamma$ we consider two possibilities, depending on the position where μ occurs in α:

2.0.1 *If* $\mu \subseteq \beta$ *or* $\mu \subseteq \gamma$. Then since $\beta, \gamma \in labels(s^\ell)$, we may apply the *i.h.* to obtain that σ^ℓ can be written as of the form $\sigma^\ell = \sigma_1^\ell R^\ell \sigma_2^\ell$ where the name of R^ℓ is μ, so we conclude.

2.0.2 *Otherwise.* In any other case, μ must overlap one of the two boundaries in the expression $\beta \bullet \gamma$. Then μ cannot be the name of a `db` redex, and it must be the name of a `ls` redex, *i.e.* $\mu = \delta \bullet \varepsilon$ where δ and ε are atomic labels. The only remaining possibility is that $\delta = \downarrow(\beta)$ and $\varepsilon = \downarrow(\gamma)$. Hence the name of the step S^ℓ is $\nu = \downarrow(\beta) \bullet \downarrow(\gamma) = \mu$, and we conclude.

□

Completeness

The following definitions and lemmas are used to prove the implication $(2 \implies 1)$ (**Completeness**) in Prop. 7.12.

Lemma A.95 (Any redex has a residual after a derivation not including its name)**.** *In the LSC without* `gc`*, let* R_0 *be a step and* ρ *be a coinitial derivation. Let* t^ℓ *be an initially reachable variant of the source, and consider the labeled variants* R_0^ℓ *and* ρ^ℓ *of* R_0 *and* ρ *respectively, whose source is*

t^ℓ. Let μ be the name of R_0^ℓ, and suppose that μ is not among the names of the redexes contracted by ρ^ℓ. Then there exists a step $R_1 \in R_0/\rho$. Moreover, the name of its labeled variant R_1^ℓ is also μ.

Proof. By induction on ρ. If ρ is empty it is immediate by taking $R_1 := R_0$. Otherwise, ρ^ℓ is of the form $S^\ell\sigma^\ell$. The names of R_0^ℓ and S^ℓ are different by hypothesis, hence R_0 and S must be different steps. Since in the LSC without gc there is no erasure, there is at least one residual $R_2 \in R_0/S$, and given that residuals have the same name as their ancestors (Lem. 6.33), the name of the labeled variant R_2^ℓ of R_2 is also μ. Applying the *i.h.*, we conclude that there is a step $R_1 \in R_2/\sigma$, *i.e.* $R_1 \in R_0/S\sigma$, and the name of the labeled variant R_1^ℓ of R_1 is also μ, as required. □

Lemma A.96 (Any redex has an ancestor before a derivation not contributing to its name)**.** *In the LSC without* gc*, let ρ be a derivation and let R_1 be a composable step,* i.e. $\mathsf{tgt}(\rho) = \mathsf{src}(R_1)$*. Let t^ℓ be an initially reachable variant of the source of ρ, consider the labeled variant ρ^ℓ of ρ whose source is t^ℓ, and the labeled variant of R_1^ℓ of R_1 whose source if* $\mathsf{tgt}(\rho^\ell)$*. Let μ be the name of R_1^ℓ, and suppose that the names of the redexes contracted by ρ^ℓ do not contribute to μ,* i.e. *every step S^ℓ in ρ^ℓ has a name ν such that $\nu \overset{\mathsf{Name}}{\hookrightarrow} \mu$ does not hold. Then there exists a step R_0 such that $R_1 \in R_0/\rho$. Moreover, the name of its labelled variant R_0^ℓ is also μ.*

Proof. By induction on the length of ρ. If ρ is empty it is immediate by taking $R_0 := R_1$. Otherwise, ρ^ℓ is of the form $\sigma^\ell S^\ell$. The name of S^ℓ does not contribute to the name of R_0^ℓ by hypothesis.

Recall that Prop. 6.41 states that whenever a step T_1^ℓ creates a step T_2^ℓ we have that the name of T_1^ℓ contributes to the name of T_2^ℓ. By the contrapositive, whenever the name of a step T_1^ℓ does not contribute to the name of a step T_2^ℓ, the second step T_2^ℓ must have an ancestor.

In our case, given that the name of S^ℓ does not contribute to the name of R_0^ℓ, there must exist an ancestor, *i.e.* a step R_2 such that $R_0 \in R_2/S$. Moreover, by the fact that residuals have the same name as their ancestors (Lem. 6.33) the name of the labeled variant R_2^ℓ of R_2 must be μ.

Then, by applying the *i.h.*, we conclude that there is a step R_1 such that $R_2 \in R_1/\sigma$, *i.e.* $R_0 \in R_1/\sigma S$, and the name of the labeled variant R_0^ℓ of R_0 is also μ, as required. □

A.4.2 Reachable normal forms are stable — proof of Prop. 7.27

Definition A.97 (Reachable contexts)**.** Reachable contexts are defined by the following grammar:

$$\mathtt{R} ::= \Box \mid \mathtt{R}\,t \mid t\,\mathtt{R} \mid \lambda x.\mathtt{R} \mid \mathtt{R}[x\backslash t] \mid \mathtt{R}\langle\langle x\rangle\rangle[x\backslash\mathtt{R}]$$

A variable x is *reachable* in a term t if it occurs free under a reachable context, *i.e.* $t = \mathtt{R}\langle\langle x\rangle\rangle$ such that R does not bind x. We write $\mathsf{rv}(t)$ for the set of reachable variables of t. Given a term t, a *reachable step* is given by either a db redex whose application lies below a reachable context, or an ls redex contracting a variable which lies below a reachable context. A term t is a *reachable-normal form* if it has no reachable redexes. The set of reachable-normal forms is written RNF. If a context (resp. variable, redex) is not reachable we say that it is *unreachable*.

Our aim is to prove that the set of reachable normal forms is a stable set. The proof depends on a number of technical definitions and lemmas. We omit the long proofs by case analysis of these lemmas.

Definition A.98 (Nesting)**.** We follow the definition of nesting given in [6]. Namely R *immediately nests* S (written $R \prec^1_{\mathtt{B}} S$) if the *anchor* of S lies inside the *box* of R. Moreover, $R \prec_{\mathtt{B}} S$ is defined as the transitive closure of $\prec^1_{\mathtt{B}}$, and then we say that R *nests* S.

Definition A.99 (Strongly reachable redex)**.** A step $R : t \to_{\mathtt{db} \cup \mathtt{ls}} s$ is *strongly reachable* if and only if R is reachable and it is not nested by any other redex, *i.e.* R is minimal with respect to $\prec_{\mathtt{B}}$.

Lemma A.100 (Characterization of reachability)**.** *The following properties hold:*

1. *A variable x is reachable in t if and only if $x \in \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(t))$.*
2. *A term t is a reachable normal form if and only if $\mathsf{nf}_{\mathtt{gc}}(t)$ is in $\to_{\mathtt{db} \cup \mathtt{ls}}$-normal form.*

Proof. The proof of the first item is by induction on t. The interesting case is when $t = s[x\backslash u]$. Then:

$$\mathsf{nf}_{\mathtt{gc}}(t) = \begin{cases} \mathsf{nf}_{\mathtt{gc}}(s)[y\backslash \mathsf{nf}_{\mathtt{gc}}(u)] & \text{if } y \in \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(s)) \\ \mathsf{nf}_{\mathtt{gc}}(s) & \text{otherwise} \end{cases}$$

so there are two cases:

1. *If $y \in \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(s))$.* Then by *i.h.*, $y \in \mathsf{rv}(s)$, so $x \in \mathsf{rv}(t) \iff x \in (\mathsf{rv}(s)\backslash\{y\}) \cup \mathsf{rv}(u) \iff x \in (\mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(s))\backslash\{y\}) \cup \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(u)) \iff x \in \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(t))$.
2. *If $y \notin \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(s))$.* Then by *i.h.*, $y \notin \mathsf{rv}(s)$, so $x \in \mathsf{rv}(t) \iff x \in \mathsf{rv}(s) \iff x \in \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(s)) \iff x \in \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(s[y\backslash u])) \iff x \in \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(t))$.

The proof of the second item is similar, by induction on t. As before, the interesting case is when $t = s[x\backslash u]$ and there are two cases.

1. *If $x \in \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(s))$.* By item 1. of this lemma, we have that $x \in \mathsf{rv}(s)$, so $s = \mathtt{R}\langle\langle x \rangle\rangle$. Then the term t is not a RNF, since $t = \mathtt{R}\langle\langle x \rangle\rangle[x\backslash u]$ so it has a reachable $\mathtt{ls}$ step. On the other hand, the term $\mathsf{nf}_{\mathtt{gc}}(t)$ is not a $\to_{\mathtt{db} \cup \mathtt{ls}}$-normal form, since $\mathsf{nf}_{\mathtt{gc}}(t) = \mathsf{nf}_{\mathtt{gc}}(s)[x\backslash \mathsf{nf}_{\mathtt{gc}}(u)]$ and x occurs free in $\mathsf{nf}_{\mathtt{gc}}(s)$, so there is a $\mathtt{ls}$ step.
2. *If $x \notin \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(s))$.* By item 1. of this lemma, we have that $x \notin \mathsf{rv}(s)$, so t is a RNF $\iff$ s, u are RNFs $\iff$ $\mathsf{nf}_{\mathtt{gc}}(s), \mathsf{nf}_{\mathtt{gc}}(u)$ are $\to_{\mathtt{db} \cup \mathtt{ls}}$-normal forms $\iff$ $\mathsf{nf}_{\mathtt{gc}}(s)[x\backslash \mathsf{nf}_{\mathtt{gc}}(u)]$ is a $\to_{\mathtt{db} \cup \mathtt{ls}}$-normal form.

□

Definition A.101 (Free occurrence, descendant of a free occurrence)**.** We say that $(t, \mathtt{C}, x)$ is a *free occurrence* if $t = \mathtt{C}\langle\langle x \rangle\rangle$ where $\mathtt{C}$ does not bind x. A free occurrence $(t, \mathtt{C}, x)$ is *reachable* if $\mathtt{C}$ is a reachable context, and *unreachable* otherwise. If $\rho : t \twoheadrightarrow s$ is a reduction sequence, we say that a free occurrence $(s, \mathtt{C}_2, x)$ is a *descendant* of a free occurrence $(t, \mathtt{C}_1, x)$ after ρ if given an initially labelled variant $\rho^\ell : t^\ell \twoheadrightarrow_\ell s^\ell$ of the reduction ρ we have that $t^\ell = \mathtt{C}_1^\ell\langle\langle x^{\downarrow(\alpha)} \rangle\rangle$ and $s^\ell = \mathtt{C}_2^\ell\langle\langle x^\alpha \rangle\rangle$, where $\mathtt{C}_1^\ell$ and $\mathtt{C}_2^\ell$ are labelled variants of $\mathtt{C}_1$ and $\mathtt{C}_2$ respectively.

Remark A.102. Since the labeled calculus LLSC is an orthogonal axiomatic rewriting system (Prop. 6.32), descendants after a derivation ρ coincide with descendants after a derivation σ whenever ρ and σ are permutation equivalent.

Lemma A.103 (Unreachable occurrences and steps can be erased)**.** *Consider a reduction to* gc*-normal form* $\sigma : t \twoheadrightarrow_{\mathsf{gc}} \mathsf{nf}_{\mathsf{gc}}(t)$*. Then:*

1. *If* $(t, \mathtt{C}, x)$ *is an unreachable free occurrence, then* $(t, \mathtt{C}, x)$ *has no descendants after* σ*.*
2. *If* $R : t \to_{\mathtt{db} \cup \mathtt{ls}} s$ *is an unreachable step, then* R/σ *is empty.*

Proof. We only give the proof of item 1., by induction on $\mathtt{C}$. The proof of item 2. is similar, by induction on the context $\mathtt{C}$, under which the step R takes place, and depends on item 1.

1. *Empty,* $\mathtt{C} = \Box$. Impossible, since then $\mathtt{C}$ is a reachable context.
2. *Left of an application,* $\mathtt{C} = \mathtt{C}'\, s$. Note that $\mathtt{C}'$ must be an unreachable context, otherwise $\mathtt{C}$ would be reachable. So $(\mathtt{C}'\langle\!\langle x \rangle\!\rangle, \mathtt{C}', x)$ is an unreachable free occurrence. Consider a derivation to gc normal form $\sigma : \mathtt{C}'\langle\!\langle x \rangle\!\rangle\, s \twoheadrightarrow_{\mathsf{gc}} \mathsf{nf}_{\mathsf{gc}}(\mathtt{C}'\langle\!\langle x \rangle\!\rangle)\, \mathsf{nf}_{\mathsf{gc}}(s)$. By algebraic confluence, we know that σ is permutation equivalent to the composition of two derivations, *i.e.* $\sigma \equiv \sigma_1\, \sigma_2$, where σ_1 carries $\mathtt{C}'\langle\!\langle x \rangle\!\rangle$ to gc-normal form and σ_2 carries s to gc-normal form. More precisely, σ_1 is defined as the embedding of the reduction sequence τ_1 under the context $\Box\, s$, and σ_2 is defined as the embedding of the reduction sequence τ_2 under the context $\mathsf{nf}_{\mathsf{gc}}(\mathtt{C}'\langle\!\langle x \rangle\!\rangle)\, \Box$, where $\tau_1 : \mathtt{C}'\langle\!\langle x \rangle\!\rangle \twoheadrightarrow_{\mathsf{gc}} \mathsf{nf}_{\mathsf{gc}}(\mathtt{C}'\langle\!\langle x \rangle\!\rangle)$ and $\tau_2 : s \twoheadrightarrow_{\mathsf{gc}} \mathsf{nf}_{\mathsf{gc}}(s)$. By *i.h.* $(\mathtt{C}'\langle\!\langle x \rangle\!\rangle, \mathtt{C}', x)$ has no descendants after τ_1. This implies that $(\mathtt{C}\langle\!\langle x \rangle\!\rangle, \mathtt{C}, x)$ has no descendants after σ_1. Hence $(\mathtt{C}\langle\!\langle x \rangle\!\rangle, \mathtt{C}, x)$ has no descendants after $\sigma_1\sigma_2 \equiv \sigma$, as required.
3. *Right of an application,* $\mathtt{C} = s\, \mathtt{C}'$. By *i.h.*, similar to the previous case.
4. *Under an abstraction,* $\mathtt{C} = \lambda x.\mathtt{C}'$. By *i.h.*, similar to the previous case.
5. *Left of a substitution,* $\mathtt{C} = \mathtt{C}'[x\backslash s]$. Note that $\mathtt{C}'$ must be an unreachable context, otherwise $\mathtt{C}$ would be reachable. So $(\mathtt{C}'\langle\!\langle x \rangle\!\rangle, \mathtt{C}', x)$ is an unreachable free occurrence. Consider a derivation to gc normal form $\sigma : \mathtt{C}'\langle\!\langle x \rangle\!\rangle[x\backslash s] \twoheadrightarrow_{\mathsf{gc}} \mathsf{nf}_{\mathsf{gc}}(\mathtt{C}'\langle\!\langle x \rangle\!\rangle)[x\backslash \mathsf{nf}_{\mathsf{gc}}(s)]^\star$. By algebraic confluence, we know that σ is permutation equivalent to the composition of three derivations, *i.e.* $\sigma \equiv \sigma_1\, \sigma_2\, \sigma_3$, where σ_1 carries $\mathtt{C}'\langle\!\langle x \rangle\!\rangle$ to gc-normal form, σ_2 carries s to gc-normal form, and σ_3 performs the garbage collection of the outermost substitution, if possible. More precisely, σ_1 is defined as the embedding of the reduction sequence τ_1 under the context $\Box[x\backslash s]$, and σ_2 is defined as the embedding of the reduction sequence τ_2 under the context $\mathsf{nf}_{\mathsf{gc}}(\mathtt{C}'\langle\!\langle x \rangle\!\rangle)[x\backslash \Box]$, where $\tau_1 : \mathtt{C}'\langle\!\langle x \rangle\!\rangle \twoheadrightarrow_{\mathsf{gc}} \mathsf{nf}_{\mathsf{gc}}(\mathtt{C}'\langle\!\langle x \rangle\!\rangle)$ and $\tau_2 : s \twoheadrightarrow_{\mathsf{gc}} \mathsf{nf}_{\mathsf{gc}}(s)$. Moreover, σ_3 is defined either an empty derivation or a single gc step, in such a way that $\sigma_3 : \mathsf{nf}_{\mathsf{gc}}(\mathtt{C}'\langle\!\langle x \rangle\!\rangle)[x\backslash \mathsf{nf}_{\mathsf{gc}}(s)] \twoheadrightarrow_{\mathsf{gc}} \mathsf{nf}_{\mathsf{gc}}(\mathtt{C}'\langle\!\langle x \rangle\!\rangle)[x\backslash \mathsf{nf}_{\mathsf{gc}}(s)]^\star$. By *i.h.* $(\mathtt{C}'\langle\!\langle x \rangle\!\rangle, \mathtt{C}', x)$ has no descendants after τ_1. This implies that $(\mathtt{C}\langle\!\langle x \rangle\!\rangle, \mathtt{C}, x)$ has no descendants after σ_1. Hence $(\mathtt{C}\langle\!\langle x \rangle\!\rangle, \mathtt{C}, x)$ has no descendants after $\sigma_1\sigma_2\sigma_3 \equiv \sigma$, as required.

6. *Inside a substitution,* $\mathtt{C} = s[y\backslash \mathtt{C}']$. We know that $\mathtt{C}$ is an unreachable context. Hence there are two possibilities, depending on whether y is a reachable variable in s:

   6.1 *If y is a reachable variable in s.* Then $\mathtt{C}'$ must be an unreachable context, otherwise $\mathtt{C}$ would be reachable. The result follows by applying the *i.h.*, similarly as in the previous case.

   6.2 *If y is not an reachable variable in s.* That is, $y \notin \mathsf{rv}(s)$. Then $y \notin \mathtt{fv}(\mathsf{nf}_{\mathtt{gc}}(s))$ by the characterization of reachable variables (Lem. A.100). This means that $\mathsf{nf}_{\mathtt{gc}}(s[y\backslash \mathtt{C}'\langle\langle x \rangle\rangle]) = \mathsf{nf}_{\mathtt{gc}}(s)$ so, by algebraic confluence, σ is permutation equivalent to the composition of a derivation and a step, *i.e.* $\sigma \equiv \sigma_1\, S$, such that σ_1 normalizes s and S performs the $\mathtt{gc}$ step that erases the outermost substitution. More precisely, σ_1 is defined as the embedding of a derivation τ_1 under the context $\Box[y\backslash \mathtt{C}'\langle\langle x \rangle\rangle]$, where $\tau_1 : s \twoheadrightarrow_{\mathtt{gc}} \mathsf{nf}_{\mathtt{gc}}(s)$ and S is $\mathtt{gc}$ step $S : \mathsf{nf}_{\mathtt{gc}}(s)[y\backslash \mathtt{C}'\langle\langle x \rangle\rangle] \to_{\mathtt{gc}} \mathsf{nf}_{\mathtt{gc}}(s)$. Then the free occurrence $(t, \mathtt{C}, x)$ has no descendants after $\sigma_1 S \equiv \sigma$, since S erases the subterm that contains the descendant of said free occurrence.

□

Lemma A.104 (Composition of reachable contexts)**.** *The composition $\mathtt{C}_1\langle \mathtt{C}_2 \rangle$ is a reachable context if and only if $\mathtt{C}_1$ and $\mathtt{C}_2$ are reachable contexts.*

Proof. Straightforward by induction on $\mathtt{C}_1$. □

Lemma A.105 (Strongly reachable redexes are not nested)**.** *Let R be a strongly reachable redex. Then for any other redex S we have that $\neg(S \prec_{\mathtt{B}} R)$.*

Proof. Suppose that $S \prec_{\mathtt{B}} R$. Then since $\prec_{\mathtt{B}}$ is defined as the transitive closure of $\prec^1_{\mathtt{B}}$, we have that $S \preceq_{\mathtt{B}} S' \prec^1_{\mathtt{B}} R$. Note that S' cannot be a reachable redex, since R is strongly reachable, and hence minimal among the reachable redexes. So S' is unreachable. Let us show that this is impossible, by case analysis on the kind of redex S':

1. *If S' is a* $\mathtt{db}$ *redex.* Then S' is of the form $S' : \mathtt{C}\langle(\lambda x.t)\mathtt{L}\, s\rangle \to_{\mathtt{db}} \mathtt{C}\langle t[x\backslash s]\mathtt{L}\rangle$ and since $S' \prec_{\mathtt{B}} R$, the anchor of R is inside s, *i.e.* $s = \mathtt{C}'\langle u\rangle$ where the root of u is the anchor of R. Then since R is reachable we have that the context $\mathtt{C}\langle(\lambda x.t)\mathtt{L}\, \mathtt{C}'\rangle$ is a reachable context. By Lem. A.104 we conclude that $\mathtt{C}$ must also be a reachable context. Hence S' is reachable, which is a contradiction.

2. *If S' is a* $\mathtt{ls}$ *redex.* Then S' is of the form: $S' : \mathtt{C}_1\langle \mathtt{C}_2\langle\langle x \rangle\rangle[x\backslash t]\rangle \to_{\mathtt{ls}} \mathtt{C}_1\langle \mathtt{C}_2\langle t\rangle[x\backslash t]\rangle$ and since $S' \prec_{\mathtt{B}} R$, the anchor of R is inside t, *i.e.* $t = \mathtt{C}'\langle s\rangle$ where the root of s is the anchor of R. Then since R is reachable we have that the context $\mathtt{C}_1\langle \mathtt{C}_2\langle\langle x \rangle\rangle[x\backslash \mathtt{C}']\rangle$ is a reachable context. By Lem. A.104 we conclude that $\mathtt{C}_1$ and $\mathtt{C}_2\langle\langle x \rangle\rangle[x\backslash \mathtt{C}']$ must also be reachable contexts. Now observe that the only production that can be used to derive that $\mathtt{C}_2\langle\langle x \rangle\rangle[x\backslash \mathtt{C}']$ is a reachable context, is "$\mathtt{R} ::= \mathtt{R}\langle\langle x \rangle\rangle[x\backslash \mathtt{R}]$". So we must have that $\mathtt{C}_2\langle\langle x \rangle\rangle = \mathtt{C}_3\langle\langle x \rangle\rangle$ where $\mathtt{C}_3$ is a reachable context. By Lem. A.104 we have that $\mathtt{C}_1\langle \mathtt{C}_3[x\backslash t]\rangle$ is a reachable context. Hence the $\mathtt{ls}$ step $T : \mathtt{C}_1\langle \mathtt{C}_3\langle\langle x \rangle\rangle[x\backslash t]\rangle \to_{\mathtt{ls}}$

$\mathtt{C}_1\langle \mathtt{C}_3\langle t\rangle[x\backslash t]\rangle$ is reachable. Moreover $T \prec^1_{\mathtt{B}} R$, since the anchor of R is inside the argument of the substitution $[x\backslash t]$. This contradicts the fact that R is strongly reachable, and we obtain a contradiction, as required.

□

Lemma A.106 (Context-freeness)**.** *Let R, S, and T be coinitial redexes such that $R' \in R/S$ and $T' \in T/S$. If $\neg(S \prec_{\mathtt{B}} R)$ then $(T \prec_{\mathtt{B}} R \iff T' \prec_{\mathtt{B}} R')$.*

Proof. This is the context-freeness property. See [6, Proposition 4] for a proof. □

Definition A.107 (Chained substitution context)**.** A substitution context $\mathtt{L}$ is said to be (x, z)-*chained* according to the following inductive definition:

$$\frac{x \notin \mathrm{dom}(\mathtt{L})}{\mathtt{L} \text{ is } (x,x)\text{-chained}} \qquad \frac{x \neq y \quad y \neq z \quad \mathtt{C} \text{ is a reachable context} \quad \mathtt{L} \text{ is } (x,y)\text{-chained}}{\mathtt{L}[y\backslash \mathtt{C}\langle\!\langle z\rangle\!\rangle] \text{ is } (x,z)\text{-chained}}$$

$$\frac{x \neq y \quad y \neq z \quad \mathtt{L} \text{ is } (x,z)\text{-chained}}{\mathtt{L}[y\backslash t] \text{ is } (x,z)\text{-chained}}$$

Remark A.108. In general a x-chained substitution context is of the form:

$$\mathtt{L}_0[x_0\backslash \mathtt{C}_0\langle\!\langle x_1\rangle\!\rangle]\mathtt{L}_1[x_1\backslash \mathtt{C}_1\langle\!\langle x_2\rangle\!\rangle]\dots\mathtt{L}_{n-1}[x_{n-1}\backslash \mathtt{C}_{n-1}\langle\!\langle x_n\rangle\!\rangle]\mathtt{L}_n$$

for some integer $n \geqslant 0$, where $x_0 = x$, $x_{n+1} = z$, and $\mathtt{C}_i$ is a reachable context for every $0 \leqslant i \leqslant n$.

Remark A.109. If $\mathtt{L}_1$ is (x, y)-chained and $\mathtt{L}_2$ is (y, z)-chained, then $\mathtt{L}_1\mathtt{L}_2$ is (x, z)-chained.

Lemma A.110 (Reachability inside a substitution)**.** *A context $t\mathtt{L}[z\backslash \mathtt{C}]$ is reachable if and only if there exists a variable x such that the following three conditions hold:*

1. *$\mathtt{C}$ is a reachable context,*
2. *t is of the form $t = \mathtt{C}'\langle\!\langle x\rangle\!\rangle$ where $\mathtt{C}'$ is a reachable context, and*
3. *$\mathtt{L}$ is a (x, z)-chained substitution context.*

Proof. Let us check each side of the implication:

($\Rightarrow$) It is immediate to check that $\mathtt{C}$ must be a reachable context by Lem. A.104. Let us check the second and third conditions by induction on the length of $\mathtt{L}$. If $\mathtt{L}$ is empty, then t must be of the form $\mathtt{C}'\langle\!\langle z\rangle\!\rangle$ and indeed $\mathtt{L} = \Box$ is (z, z)-chained. If $\mathtt{L} = \mathtt{L}'[y\backslash s]$ there are two possibilities:

1. *If z is reached via s.* That is $s = \mathtt{C}''\langle\!\langle z\rangle\!\rangle$ where $t\mathtt{L}'[y\backslash \mathtt{C}'']$ is a reachable context. By *i.h.* $t = \mathtt{C}'\langle\!\langle x\rangle\!\rangle$ and $\mathtt{L}'$ is a (x, y)-chained substitution context. So $\mathtt{L} = \mathtt{L}'[y\backslash \mathtt{C}''\langle\!\langle z\rangle\!\rangle]$ is (x, z)-chained.

2. *Otherwise.* Then $t\mathtt{L}'$ is of the form $\mathtt{C}''\langle\langle z\rangle\rangle$ where $\mathtt{C}''$ is a reachable context, so $t\mathtt{L}'[z\backslash\mathtt{C}]$ is a reachable context. By *i.h.* this means that $t = \mathtt{C}'\langle\langle x\rangle\rangle$ and $\mathtt{L}'$ is a (x, z)-chained substitution context. So $\mathtt{L} = \mathtt{L}'[y\backslash s]$ is also (x, z)-chained.

($\Leftarrow$) Let $t = \mathtt{C}'\langle\langle x\rangle\rangle$. By induction on the derivation that $\mathtt{L}$ is (x, z)-chained:

1. *If* $\mathtt{L}$ *is* (z, z)*-chained with* $z \notin \mathsf{dom}(\mathtt{L})$. Then $x = z$ and $\mathtt{C}'\langle\langle z\rangle\rangle[z\backslash\mathtt{C}]$ is reachable since $\mathtt{C}'$ and $\mathtt{C}$ are reachable.
2. *If* $\mathtt{L} = \mathtt{L}'[y\backslash\mathtt{C}''\langle\langle z\rangle\rangle]$ *where* $\mathtt{C}''$ *is reachable and* $\mathtt{L}'$ *is* (x, y)*-chained.* By *i.h.* we have that $\mathtt{C}'\langle\langle x\rangle\rangle\mathtt{L}'[y\backslash\mathtt{C}'']$ is a reachable context. Hence $\mathtt{C}'\langle\langle x\rangle\rangle\mathtt{L}'[y\backslash\mathtt{C}''\langle\langle z\rangle\rangle][z\backslash\mathtt{C}]$ is reachable.
3. *If* $\mathtt{L} = \mathtt{L}'[y\backslash s]$ *where* $\mathtt{L}'$ *is* (x, z)*-chained.* By *i.h.* we have that $\mathtt{C}'\langle\langle x\rangle\rangle\mathtt{L}'[z\backslash\mathtt{C}]$ is a reachable context. Hence by Lem. A.104 $\mathtt{C}'\langle\langle x\rangle\rangle\mathtt{L}'$ must be of the form $\mathtt{C}_1\langle\langle z\rangle\rangle$ where $\mathtt{C}_1$ is a reachable context. Then $\mathtt{C}_1[y\backslash s]$ is also reachable, which in turn implies that $\mathtt{C}_1\langle\langle z\rangle\rangle[y\backslash s][z\backslash\mathtt{C}] = \mathtt{C}'\langle\langle x\rangle\rangle\mathtt{L}'[y\backslash s][z\backslash\mathtt{C}]$ is reachable, as required.

□

Lemma A.111 (Strongly reachable redexes have reachable residuals)**.** *Let R be a strongly reachable redex and let $S \neq R$ be any other redex coinitial to R. Then:*

- *The set of residuals R/S is a singleton and it is reachable.*
- *If* $\mathsf{tgt}(R)$ *is in* RNF*, then R/S is strongly reachable.*

Proof. During the proof we shall implicity use the fact that these two conditions are equivalent for a reachable redex R:

(1) For every redex S we have that $\neg(S \prec_{\mathtt{B}} R)$.

(2) For every reachable redex S we have that $\neg(S \prec_{\mathtt{B}} R)$, *i.e.* R is strongly reachable.

The implication $(1 \implies 2)$ is immediate. The reverse implication is Lem. A.105. The proof proceeds by induction on the context $\mathtt{C}$ under which the S step takes place. We study only the base case, when $\mathtt{C} = \Box$. The inductive cases are not all straightforward, but they can be proved using the same ideas. By case analysis on the kind of redex S:

1. $\mathtt{db}$ *redex.* Then the step S is of the form $(\lambda x.t)\mathtt{L}\, s \xrightarrow{S} t[x\backslash s]\mathtt{L}$. Note that S is a reachable redex so, since R is minimal, the anchor of R cannot be internal to s. We consider two subcases, depending on whether the anchor of R is internal to t or internal to $\mathtt{L}$.

   1.1 *If the anchor of R is internal to t.* Then $t = \mathtt{C}'\langle u\rangle$ where the anchor of R is at the root of u and the situation is:

$$\begin{array}{ccc} (\lambda x.\mathtt{C}'\langle u\rangle)\mathtt{L}\, s & \xrightarrow{S} & \mathtt{C}'\langle u\rangle[x\backslash s]\mathtt{L} \\ {\scriptstyle R}\downarrow & & {\scriptstyle R/S}\downarrow \\ (\lambda x.\mathtt{C}'\langle u'\rangle)\mathtt{L}\, s & & \mathtt{C}'\langle u'\rangle[x\backslash s]\mathtt{L} \end{array}$$

Note that R is reachable, so $(\lambda x.\mathtt{C}')\mathtt{L}\ s$ is a reachable context. Then, by Lem. A.104, we have that $\mathtt{C}'[x\backslash s]\mathtt{L}$ is also a reachable context. Hence R/S is reachable.

Moreover, let us show that R/S is strongly reachable. By contradiction, suppose that $T \prec^1_{\mathtt{B}} R/S$ for some reachable redex T. We consider three cases, depending on whether T is a `db` redex created by S, a `ls` redex created by S, or it has has an ancestor before S:

1.1.1 *T is a* `db` *redex created by* S. This case is not possible, since there should be an outermost application in order to create a `db` redex in this way.

1.1.2 *T is a* `ls` *redex created by* S. This case is not possible as we would have $\neg(T \prec^1_{\mathtt{B}} R)$, since the box of T is the argument of the substitution $[x\backslash s]$ and the anchor of R/S is inside u.

1.1.3 *T has an ancestor before* S. Let T_0 be an ancestor of T, *i.e.* $T \in T_0/S$. Then since $\neg(S \prec_{\mathtt{B}} R)$ and $T \prec_{\mathtt{B}} R/S$, by Context-freeness (Lem. A.106) we have that $T_0 \prec_{\mathtt{B}} R$, contradicting the fact that R is minimal with respect to the box order.

1.2 *If the anchor of R is internal to* `L`. Then $\mathtt{L} = \mathtt{L}_1[y\backslash \mathtt{C}'u]\mathtt{L}_2$, where the anchor of R is at the root of u, and the situation is:

$$\begin{array}{ccc} (\lambda x.t)\mathtt{L}_1[y\backslash \mathtt{C}'\langle u\rangle]\mathtt{L}_2\ s & \xrightarrow{S} & t[x\backslash s]\mathtt{L}_1[y\backslash \mathtt{C}'\langle u\rangle]\mathtt{L}_2 \\ {\scriptstyle R}\downarrow & & {\scriptstyle R/S}\downarrow \\ (\lambda x.t)\mathtt{L}_1[y\backslash \mathtt{C}'\langle u'\rangle]\mathtt{L}_2\ s & & t[x\backslash s]\mathtt{L}_1[y\backslash \mathtt{C}'\langle u'\rangle]\mathtt{L}_2 \end{array}$$

Then since R is a reachable redex, the context $(\lambda x.t)\mathtt{L}_1[y\backslash \mathtt{C}']\mathtt{L}_2\ s$ must be reachable. By Lem. A.110 there must exist a variable z such that $\lambda x.t$ is of the form $\mathtt{C}''\langle\!\langle z\rangle\!\rangle$ where $\mathtt{C}''$ is a reachable context, and moreover $\mathtt{L}_1$ is a (z, y)-chained substitution context. Note that $\lambda x.t = \mathtt{C}''\langle\!\langle z\rangle\!\rangle$ so $\mathtt{C}'' = \lambda x.\mathtt{C}_1$ and $x \neq z$. In particular, $t = \mathtt{C}_1\langle\!\langle z\rangle\!\rangle$ and $\mathtt{C}_1$ is a reachable context. By applying Lem. A.110 now in the opposite direction, we have that the context $t[x\backslash s]\mathtt{L}_1[y\backslash \mathtt{C}']\mathtt{L}_2$ is reachable, so R/S is a reachable redex. Moreover, let us show that R/S is strongly reachable. By contradiction, suppose that $T \prec^1_{\mathtt{B}} R/S$ for some reachable redex T. We consider three cases, depending on whether T is a `db` redex created by S, a `ls` redex created by S, or it has has an ancestor before S:

1.2.1 *T is a* `db` *redex created by* S. This case is not possible, since there should be an outermost application in order to create a `db` redex in this way.

1.2.2 *T is a* `ls` *redex created by* S. Then T contracts an occurrence of x in t, and the box of T is the argument of the substitution $[x\backslash s]$. The anchor of R/S is not inside that substitution, so we have $\neg(T \prec^1_{\mathtt{B}} R/S)$.

1.2.3 *T has an ancestor before* S. Let T_0 be an ancestor of T, *i.e.* $T \in T_0/S$. Then since $\neg(S \prec_{\mathtt{B}} R)$ and $T \prec_{\mathtt{B}} R/S$, by Context-freeness (Lem. A.106) we have that $T_0 \prec_{\mathtt{B}} R$, contradicting the fact that R is minimal with respect to the box order.

2. $\mathtt{ls}$ *redex.* Then the step S is of the form: $\mathtt{C}_1\langle\!\langle x \rangle\!\rangle[x\backslash t] \xrightarrow{S} \mathtt{C}_1\langle t\rangle[x\backslash t]$. Let us consider two cases, depending on whether the anchor of R is to the left of the substitution, or inside the substitution:

   2.1 *If the anchor of R is to the left of the substitution,* i.e. *internal to* $\mathtt{C}_1\langle\!\langle x \rangle\!\rangle$. Let $\mathtt{C}_1\langle\!\langle x \rangle\!\rangle[x\backslash t] = \mathtt{C}'\langle u\rangle[x\backslash t]$ such that u is the anchor of R. We consider three further subcases, depending on the relative positions of the hole of $\mathtt{C}_1$ and the hole of $\mathtt{C}'$:

      2.1.1 *If $\mathtt{C}_1$ is a prefix of $\mathtt{C}'$.* That is, $\mathtt{C}' = \mathtt{C}_1\langle \mathtt{C}_2\rangle$. Then $\mathtt{C}'\langle\!\langle x \rangle\!\rangle = \mathtt{C}_1\langle \mathtt{C}_2\langle u\rangle\rangle$, so $\mathtt{C}_2$ must be empty and $u = x$ must be the anchor of R, so R is an $\mathtt{ls}$ redex. Furthermore R contracts the same variable occurrence as S, so $R = S$, which is impossible.

      2.1.2 *If $\mathtt{C}'$ is a prefix of $\mathtt{C}_1$.* That is, $\mathtt{C}_1 = \mathtt{C}'\langle \mathtt{C}_2\rangle$. Then the source of R and S is of the form $\mathtt{C}'\langle \mathtt{C}_2\langle\!\langle x \rangle\!\rangle\rangle[x\backslash t]$, where the subterm $\mathtt{C}_2\langle\!\langle x \rangle\!\rangle$ is the anchor of R. Note that R cannot be a $\mathtt{ls}$ step, since then we would have that $\mathtt{C}_2 = \Box$. This would in turn mean that R and S contract the same variable occurrence, so $R = S$, which is impossible. Hence R must be a $\mathtt{db}$ step, that is, $\mathtt{C}_2\langle\!\langle x \rangle\!\rangle = (\lambda y.s)\mathtt{L}\, u$. Let $(\lambda y.\hat{s})\hat{\mathtt{L}}\, \hat{u}$ be the term that results from $(\lambda y.s)\mathtt{L}\, u$ by replacing the occurrence of x under the context $\mathtt{C}_2$ by t, *i.e.* $(\lambda y.\hat{s})\hat{\mathtt{L}}\, \hat{u} = \mathtt{C}_2\langle t\rangle$. The situation is:

$$\begin{array}{ccc} \mathtt{C}'\langle(\lambda y.s)\mathtt{L}\, u\rangle[x\backslash t] & \xrightarrow{S} & \mathtt{C}'\langle(\lambda y.\hat{s})\hat{\mathtt{L}}\, \hat{u}\rangle[x\backslash t] \\ \Big\downarrow{\scriptstyle R} & & \Big\downarrow{\scriptstyle R/S} \\ \mathtt{C}'\langle s[y\backslash u]\mathtt{L}\rangle[x\backslash t] & & \mathtt{C}'\langle \hat{s}[y\backslash \hat{u}]\hat{\mathtt{L}}\rangle[x\backslash t] \end{array}$$

      By hypothesis, R is a reachable redex, so the context $\mathtt{C}'[x\backslash t]$ is reachable. Then R/S is also a reachable redex. Moreover, let us show that R/S is strongly reachable. By contradiction, suppose that $T \prec^1_{\mathtt{B}} R/S$ for some reachable redex T. We consider two cases, depending on whether T is a $\mathtt{db}$ redex created by S, or it has has an ancestor before S:

         2.1.2.1 *T is a $\mathtt{db}$ redex created by S.* In order to create a $\mathtt{db}$ redex, we know that two conditions must hold. On the first hand, the argument of the substitution contracted by S must be an answer, more precisely, we know that t is of the form $t = \mathtt{v}\mathtt{L}_1$, where $\mathtt{v}$ is an abstraction. On the other hand, $\mathtt{C}_2$ must be an applicative context, more precisely we know that $\mathtt{C}_2 = \mathtt{C}'_2\langle\Box\mathtt{L}_2\, r\rangle$.

         To conclude, note that the application node involved in the pattern of T is strictly contained either in $\hat{s}$, or in $\hat{\mathtt{L}}$, or in $\hat{u}$, so it cannot possibly nest R/S. To state it more precisely, we consider three similar subcases, depending on whether the hole of $\mathtt{C}_2$ lies inside s, inside $\mathtt{L}$, or inside u:

         - *The hole of $\mathtt{C}_2$ lies inside s.* Then $s = \mathtt{C}''_2\langle\Box\mathtt{L}_2\, r\rangle$ and $\mathtt{C}'_2 = (\lambda y.\mathtt{C}''_2)\mathtt{L}\, u$.

The situation is:

$$\begin{array}{ccc} \mathtt{C}'\langle(\lambda y.\mathtt{C}''_2\langle x\mathtt{L}_2\, r\rangle)\mathtt{L}\, u\rangle[x\backslash \mathtt{v}\mathtt{L}_1] & \xrightarrow{S} & \mathtt{C}'\langle(\lambda y.\mathtt{C}''_2\langle \underline{\mathtt{v}\mathtt{L}_1\mathtt{L}_2\, r}\rangle)\mathtt{L}\, u\rangle[x\backslash \mathtt{v}\mathtt{L}_1] \\ {\scriptstyle R}\downarrow & & {\scriptstyle R/S}\downarrow \\ \mathtt{C}'\langle\mathtt{C}''_2\langle x\mathtt{L}_2\, r\rangle[y\backslash u]\mathtt{L}\rangle[x\backslash \mathtt{v}\mathtt{L}_1] & & \mathtt{C}'\langle\mathtt{C}''_2\langle \mathtt{v}\mathtt{L}_1\mathtt{L}_2\, r\rangle[y\backslash u]\mathtt{L}\rangle[x\backslash \mathtt{v}\mathtt{L}_1] \end{array}$$

The redex T is underlined and clearly $\neg(T \prec_{\mathtt{B}} R/S)$ since the anchor of R/S is to the left of the box of T.

- *The hole of* $\mathtt{C}_2$ *lies inside* $\mathtt{L}$. Similar to the previous case.
- *The hole of* $\mathtt{C}_2$ *lies inside* u. Similar to the previous case.

2.1.2.2 T *has an ancestor before* S. Let T_0 be an ancestor of T, *i.e.* $T \in T_0/S$. Then since $\neg(S \prec_{\mathtt{B}} R)$ and $T \prec_{\mathtt{B}} R/S$, by Context-freeness (Lem. A.106) we have that $T_0 \prec_{\mathtt{B}} R$, contradicting the fact that R is minimal with respect to the box order.

2.1.3 *If* $\mathtt{C}'$ *and* $\mathtt{C}_1$ *are disjoint.* Then there is a two hole context $\widehat{\mathtt{C}}$ such that: $\mathtt{C}' = \widehat{\mathtt{C}}\langle\Box, x\rangle$ and $\mathtt{C}_1 = \widehat{\mathtt{C}}\langle s, \Box\rangle$, and the situation is:

$$\begin{array}{ccc} \widehat{\mathtt{C}}\langle s, x\rangle[x\backslash t] & \xrightarrow{S} & \widehat{\mathtt{C}}\langle s, t\rangle[x\backslash t] \\ {\scriptstyle R}\downarrow & & {\scriptstyle R/S}\downarrow \\ \widehat{\mathtt{C}}\langle s', x\rangle[x\backslash t] & & \widehat{\mathtt{C}}\langle s'', t\rangle[x\backslash t] \end{array}$$

Since R is a reachable redex, the context $\widehat{\mathtt{C}}\langle\Box, x\rangle$ is reachable. Hence $\widehat{\mathtt{C}}\langle\Box, t\rangle$ is also reachable, which implies that R/S is also a reachable redex. Moreover, let us show that R/S is strongly reachable. By contradiction, suppose that $T \prec^1_{\mathtt{B}} R/S$ for some reachable redex T. We consider two cases, depending on whether T is a $\mathtt{db}$ redex created by S, or it has has an ancestor before S:

- T *is a* $\mathtt{db}$ *redex created by* S. In order to create a $\mathtt{db}$ redex, we know that two conditions must hold. On the first hand, the argument of the substitution contracted by S must be an answer, more precisely, we know that t is of the form $t = \mathtt{v}\mathtt{L}_1$ where $\mathtt{v}$ is an abstraction. On the other hand, $\mathtt{C}_1$ must be an applicative context, more precisely we know that $\mathtt{C}_1 = \mathtt{C}'_1\langle\Box\mathtt{L}_2\, u\rangle$. Moreover, since the anchor of R/S is inside the box of T, we have that $u = \mathtt{C}_3\langle s\rangle$, and $\widehat{\mathtt{C}} = \mathtt{C}'_1\langle\Box_2\mathtt{L}_2\, \mathtt{C}_3\langle\Box_1\rangle\rangle$ where $\Box_1$ and $\Box_2$ are the first and second parameters of the two-hole context $\widehat{\mathtt{C}}$. Then the situation is:

$$\begin{array}{ccc} \mathtt{C}'_1\langle x\mathtt{L}_2\, \mathtt{C}_3\langle s\rangle\rangle[x\backslash \mathtt{v}\mathtt{L}_1] & \xrightarrow{S} & \mathtt{C}'_1\langle \mathtt{v}\mathtt{L}_1\mathtt{L}_2\, \mathtt{C}_3\langle s\rangle\rangle[x\backslash \mathtt{v}\mathtt{L}_1] \\ {\scriptstyle R}\downarrow & & {\scriptstyle R/S}\downarrow \\ \mathtt{C}'_1\langle x\mathtt{L}_2\, \mathtt{C}_3\langle s'\rangle\rangle[x\backslash \mathtt{v}\mathtt{L}_1] & \xrightarrow{S/R} & \mathtt{C}'_1\langle \mathtt{v}\mathtt{L}_1\mathtt{L}_2\, \mathtt{C}_3\langle s'\rangle\rangle[x\backslash \mathtt{v}\mathtt{L}_1] \end{array}$$

Since R is a reachable redex by hypothesis, the context $\mathtt{C}'_1\langle x\mathtt{L}_2\, \mathtt{C}_3\rangle[x\backslash \mathtt{v}\mathtt{L}_1]$ must be reachable. By Lem. A.104 this implies that $\mathtt{C}'_1\langle\Box\mathtt{L}_2\, \mathtt{C}_3\langle s'\rangle\rangle[x\backslash \mathtt{v}\mathtt{L}_1]$ must also be a reachable context. Hence the step at the bottom of the diagram S/R is a reachable redex. This contradicts the hypothesis that the target of R is in RNF.

- T *has an ancestor before* S. Recall that R is strongly reachable so it is minimal with respect to the box order and in particular $\neg(S \prec_{\mathtt{B}} R)$.
  Let T_0 be an ancestor of T, *i.e.* $T \in T_0/S$. Then since $\neg(S \prec_{\mathtt{B}} R)$ and $T \prec_{\mathtt{B}} R/S$, by Context-freeness (Lem. A.106) we have that $T_0 \prec_{\mathtt{B}} R$, contradicting the fact that R is minimal with respect to the box order.

2.2 *If the anchor of* R *is inside the substitution,* i.e. *internal to* t. This case is impossible, as we would have $S \prec^1_{\mathtt{B}} R$, but R is strongly reachable, and in particular minimal with respect to the box order.

□

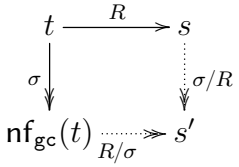
With these tools, we can prove the main result of this section.

Proposition A.112 (Full proof of Prop. 7.27—The set RNF is stable)**.**

Proof. The proof goes by checking items 1. and 2. in the definition of stable set:

1. **RNF is closed under parallel moves.** It suffices to check that RNF is closed under reduction. Let $R : t \to_{\mathtt{db} \cup \mathtt{ls}} s$ with $t \in \mathsf{RNF}$, and let us check that $s \in \mathsf{RNF}$. It can be proved as a lemma that RNFs given in Lem. A.100, that $t \in \mathsf{RNF}$ if and only if $\mathsf{nf}_{\mathsf{gc}}(t)$ is in $\to_{\mathtt{db} \cup \mathtt{ls}}$-normal form, and similarly for s.

   Let $\sigma : t \twoheadrightarrow_{\mathsf{gc}} \mathsf{nf}_{\mathsf{gc}}(t)$ be a sequence of gc steps to normal form. Since $t \in \mathsf{RNF}$, by Lem. A.100, we have that $\mathsf{nf}_{\mathsf{gc}}(t)$ is in $\to_{\mathtt{db} \cup \mathtt{ls}}$-normal form. Consider the relative projections σ/R and R/σ. Since σ/R is the projection of a sequence of gc steps, it is also sequence of gc steps. Let $\sigma/R : s \twoheadrightarrow_{\mathsf{gc}} s'$. The situation is:

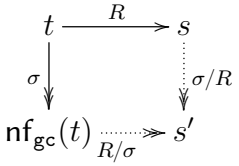
   

   Since $\mathsf{nf}_{\mathsf{gc}}(t)$ is in $\to_{\mathtt{db} \cup \mathtt{ls}}$-normal form, R/σ must be empty, so $s' = \mathsf{nf}_{\mathsf{gc}}(t)$. In particular, s' is a gc normal form, so by confluence s' is the gc normal form of s, *i.e.* $\mathsf{nf}_{\mathsf{gc}}(s) = s' = \mathsf{nf}_{\mathsf{gc}}(t)$. Therefore $\mathsf{nf}_{\mathsf{gc}}(s)$ is in $\to_{\mathtt{db} \cup \mathtt{ls}}$-normal form which means, by Lem. A.100, that $s \in \mathsf{RNF}$ as required.

2. **RNF is closed under unneeded expansion.**. Let $R : t \to_{\mathtt{db} \cup \mathtt{ls}} s$ with $t \notin \mathsf{RNF}$ and $s \in \mathsf{RNF}$, and let us show that R is RNF-needed. In fact, it suffices to show that R is a strongly reachable redex. First we prove that R is reachable.

   Claim: R *is a reachable redex.* By contradiction, suppose that R is unreachable, consider a reduction from t to gc-normal form $\sigma : t \twoheadrightarrow_{\mathsf{gc}} \mathsf{nf}_{\mathsf{gc}}(t)$, and the relative projections $R/\sigma : \mathsf{nf}_{\mathsf{gc}}(t) \twoheadrightarrow_{\mathtt{db} \cup \mathtt{ls}} s'$ and $\sigma/R : s \twoheadrightarrow_{\mathtt{db} \cup \mathtt{ls}} s'$. By the fact that unreachable redexes have no residual after going to gc-normal form (Lem. A.103) we know that R has no residual after σ, so R/σ is empty. Hence $\mathsf{nf}_{\mathsf{gc}}(t) = s'$, so

s' is in gc-normal form and by confluence we obtain that $\mathsf{nf}_{\mathtt{gc}}(s) = s' = \mathsf{nf}_{\mathtt{gc}}(t)$. The situation is:



Since $t \in \mathsf{RNF}$, by the characterization of RNFs given in Lem. A.100, we have that $\mathsf{nf}_{\mathtt{gc}}(t)$ is not a $\rightarrow_{\mathtt{db} \cup \mathtt{ls}}$-normal form. On the other hand, since $s \in \mathsf{RNF}$, by Lem. A.100, we have that $\mathsf{nf}_{\mathtt{gc}}(s) = \mathsf{nf}_{\mathtt{gc}}(t)$ is a $\rightarrow_{\mathtt{db} \cup \mathtt{ls}}$-normal form. This is a contradiction, which concludes the proof of the claim.

To see that R is a strongly reachable redex, we are left to check that R is minimal, among the reachable redexes, with respect to the nesting order $\prec_{\mathtt{B}}$. Indeed, by contradiction, suppose that R is not minimal. Then since the order $\prec_{\mathtt{B}}$ is well-founded (as there are finitely many redexes in any given term) there is a reachable redex such that $S \prec_{\mathtt{B}} R$ and such that S is minimal among the reachable redexes. That is, S is a strongly reachable redex. Then by the fact that strongly reachable redexes have reachable residuals (Lem. A.111) the redex S/R is reachable. This contradicts the fact that s is in RNF. So R must be minimal with respect to the nesting order $\prec_{\mathtt{B}}$, as required.

□

A.4.3 Head linear reduction is normalizing — proof of Coro. 7.56

Corollary A.113 (Full proof of Coro. 7.56—Head-linear reduction is HLNF-normalizing)**.** *The strategy $\mathbb{S}_{HL}$ associated to the sub-ARS $\mathbb{HL}$ is* HLNF*-normalizing.*

Proof. To show that $\mathbb{S}_{HL}$ is HLNF-normalizing, using Prop. 7.54 we must show that:

1. The set $\mathsf{NF}(\mathbb{HL})$ coincides with the set HLNF, so being $\mathsf{NF}(\mathbb{HL})$-normalizing is equivalent to being HLNF-normalizing. For this we will show the two inclusions:
   1.1 $\mathsf{NF}(\mathbb{HL}) \subseteq \mathsf{HLNF}$
   1.2 $\mathsf{HLNF} \subseteq \mathsf{NF}(\mathbb{HL})$
2. The sub-ARS $\mathbb{HL}$ is closed and residual-invariant, to be able to apply Prop. 7.54. For this we will show that:
   2.1 The set $\mathsf{NF}(\mathbb{HL})$ is closed by reduction.
   2.2 The sub-ARS $\mathbb{HL}$ is residual-invariant.

Part 1a: every $\mathbb{HL}$-normal form is a HLNF.
By induction on t it is straightforward to check that if $t \in \mathsf{NF}(\mathbb{HL})$ then $t \in \mathsf{HLNF}$.
Part 1b: every HLNF is a $\mathbb{HL}$-normal form.
It is immediate to check that $(\lambda x.t)\mathtt{L}$ and $H\langle\langle x \rangle\rangle$ have no `db` or `ls` redexes under a head

context.

Part 2a: the set $\mathsf{NF}(\mathbb{HL})$ is closed by reduction.

We have already seen that $\mathsf{NF}(\mathbb{HL}) = \mathsf{HLNF}$. Let $t \in \mathsf{HLNF}$ and $t \to s$, and let us check that $s \in \mathsf{HLNF}$. More precisely, wee show that if t is an answer, s is still an answer, and if t is a structure whose head variable is x, s is still a structure with the same head variable:

1. *If t is an answer,* $t = (\lambda x.t')\mathtt{L}$. There are three cases depending on the kind of the step $t \to s$:

   1.1 `db` *step.* A `db` step can be internal to t' or internal to one of the substitutions in $\mathtt{L}$.

   1.2 `ls` *step.* The variable contracted by an `ls` step can be internal to t' or internal to one of the substitutions in $\mathtt{L}$; the substitution affected by the `ls` step might be or not be one of the substitutions of $\mathtt{L}$.

   In any case, the contractum s is still an answer.

2. *If t is a structure,* $t = H\langle\!\langle x \rangle\!\rangle$. By induction on H:

   2.1 *Empty,* $H = \Box$. Trivial, as there are no steps from x.

   2.2 *Left of an application,* $H = H'\,u$. Three cases, depending on the position where the step $t \to s$ takes place:

      2.2.1 *At the root.* This case is impossible, as the step should be a `db` step, but $H'\langle\!\langle x \rangle\!\rangle$ is not of the form $(\lambda y.t')\mathtt{L}$.

      2.2.2 *Left of the application.* That is, the step is internal to $H'\langle\!\langle x \rangle\!\rangle$. Then we have that $t = H'\langle\!\langle x \rangle\!\rangle\, u \to r\, u = s$, and r is a structure by *i.h.*, so $s = r\,u$ is also a structure.

      2.2.3 *Right of the application.* That is, the step is internal to u. Then we have that $t = H'\langle\!\langle x \rangle\!\rangle\, u \to H'\langle\!\langle x \rangle\!\rangle\, r = s$, and $s = H'\langle\!\langle x \rangle\!\rangle\, r$ is still a structure.

   2.3 *Left of a substitution,* $H = H'[y\backslash u]$.

      2.3.1 *At the root.* The step must contract an occurrence of y in $H'\langle\!\langle x \rangle\!\rangle$. Note that, since t is a structure, the head context $H = H'[y\backslash u]$ must not bind x, and in particular $x \neq y$. So there is a two-hole context $\widehat{\mathtt{C}}$ such that $H' = \widehat{\mathtt{C}}\langle\Box, y\rangle$, and the step is of the form:

      $$t = \widehat{\mathtt{C}}\langle x, y\rangle[y\backslash u] \to \widehat{\mathtt{C}}\langle x, u\rangle[y\backslash u] = s$$

      Note that $\widehat{\mathtt{C}}\langle\Box, u\rangle$ is still a head context as a consequence of (A.11), so s is a structure with head variable x.

      2.3.2 *Left of the substitution.* Then the step is $t = H'\langle\!\langle x \rangle\!\rangle[y\backslash u] \to r[y\backslash u] = s$. By *i.h.* r is a structure with the same head variable, *i.e.* $r = H''\langle\!\langle x \rangle\!\rangle$. Note that $x \neq y$, so $s = H''\langle\!\langle x \rangle\!\rangle[y\backslash u]$ is a structure with head variable x.

      2.3.3 *Inside the substitution.* Then the step is $t = H'\langle\!\langle x \rangle\!\rangle[y\backslash u] \to H'\langle\!\langle x \rangle\!\rangle[y\backslash r] = s$. So $s = H'\langle\!\langle x \rangle\!\rangle[y\backslash r]$ is still a structure with head variable x.

Part 2b: the sub-ARS $\mathbb{HL}$ is residual-invariant.

Let $R \in \mathbb{HL}$ and consider $R \neq S$. Let us show that there is a residual $R' \in \mathbb{HL} \cap R/S$.

Before, let us state two simple facts:

$$\widehat{\mathtt{C}}\langle \Box, t\rangle \text{ is a head context } \implies \widehat{\mathtt{C}}\langle \Box, t'\rangle \text{ is a head context where } \widehat{\mathtt{C}} \text{ is any two-hole context} \tag{A.11}$$

$$\mathtt{C}_1\langle \mathtt{C}_2[x\backslash t]\rangle \text{ is a head context } \implies \mathtt{C}_1\langle \mathtt{C}_2\rangle \text{ is a head context} \tag{A.12}$$

These properties can be shown by induction on $\mathtt{C}$.

Now we proceed by induction on the head context H under which the step R takes place:

1. *Empty, $H = \Box$.* Two cases, depending on the kind of redex of R:

   1.1 *If R is a* `db` *step.* Then R is of the form:

   $$(\lambda x.t)\mathtt{L}\, s \to t[x\backslash s]\mathtt{L}$$

   There are three cases, depending on the position where S takes place. Note that S cannot be at the root:

   1.1.1 *If S is internal to t.* By this we mean that S is a `db` redex completely internal to t, or an `ls` redex whose anchor is a variable y that lies inside t (the substitution binding y might be also inside t, or it might be one of the substitutions in $\mathtt{L}$). Let $\widehat{t}$ denote the result of applying S on t. Then:

   $$\begin{array}{ccc} (\lambda x.t)\mathtt{L}\, s & \xrightarrow{R} & t[x\backslash s]\mathtt{L} \\ {\scriptstyle S}\downarrow & & \\ (\lambda x.\widehat{t})\mathtt{L}\, s & \xrightarrow{R/S} & (\lambda x.\widehat{t})\mathtt{L}\, s \end{array}$$

   and R/S is also in $\mathbb{HL}$.

   1.1.2 *If S is internal to* $\mathtt{L}$. By this we mean that $\mathtt{L} = \mathtt{L}_1[y\backslash u]\mathtt{L}_3$, and S is either a `db` redex completely internal to u, or an `ls` redex whose anchor is a variable z that lies inside u (the substitution binding z might be also inside u, or it might be one of the substitutions in $\mathtt{L}_2$). Let $\widehat{\mathtt{L}}$ denote the result of applying S on $\mathtt{L}$. Then:

   $$\begin{array}{ccc} (\lambda x.t)\mathtt{L}\, s & \xrightarrow{R} & t[x\backslash s]\mathtt{L} \\ {\scriptstyle S}\downarrow & & \\ (\lambda x.t)\widehat{\mathtt{L}}\, s & \xrightarrow{R/S} & (\lambda x.t)\widehat{\mathtt{L}}\, s \end{array}$$

   and R/S is also in $\mathbb{HL}$.

   1.1.3 *If S is internal to s.* Let $\widehat{s}$ denote the result of applying S on s. Then:

   $$\begin{array}{ccc} (\lambda x.t)\mathtt{L}\, s & \xrightarrow{R} & t[x\backslash s]\mathtt{L} \\ {\scriptstyle S}\downarrow & & \\ (\lambda x.t)\mathtt{L}\, \widehat{s} & \xrightarrow{R/S} & (\lambda x.t)\mathtt{L}\, \widehat{s} \end{array}$$

   and R/S is also in $\mathbb{HL}$.

1.2 *If R is a* `ls` *step.* Then R is of the form:

$$H\langle\!\langle x \rangle\!\rangle[x\backslash t] \to H\langle t\rangle[x\backslash t]$$

There are four cases, depending on the position where S takes place.

1.2.1 *At the root.* Note that $S \neq R$ and that S cannot be a `db` step, since there is not an application at the root. The remaining possibility is that S is a `ls` step contracting a different occurrence of x. That is, that there is a two-hole context $\widehat{\mathtt{C}}$ such that:

$$\widehat{\mathtt{C}}\langle \Box, x\rangle = H$$

Then:

$$\begin{array}{ccc} \widehat{\mathtt{C}}\langle x, x\rangle[x\backslash t] & \xrightarrow{R} & \widehat{\mathtt{C}}\langle t, x\rangle[x\backslash t] \\ {\scriptstyle S}\downarrow & & \\ \widehat{\mathtt{C}}\langle x, t\rangle[x\backslash t] & \xrightarrow{R/S} & \widehat{\mathtt{C}}\langle t, t\rangle[x\backslash t] \end{array}$$

To conclude that $R/S \in \mathbb{HL}$ it suffices to observe that $\widehat{\mathtt{C}}\langle \Box, t\rangle$ is a head context as a consequence of (A.11) and the fact that $\widehat{\mathtt{C}}\langle \Box, x\rangle$ is a head context.

1.2.2 *Internal to $H\langle\!\langle x \rangle\!\rangle$, disjoint from the hole of H.* Then there is a two-hole context $\widehat{\mathtt{C}}$ such that

$$\widehat{\mathtt{C}}\langle \Box, s\rangle = H$$

and $\widehat{\mathtt{C}}\langle x, \Box\rangle$ is the context under which the step S takes place. Then:

$$\begin{array}{ccc} \widehat{\mathtt{C}}\langle x, s\rangle[x\backslash t] & \xrightarrow{R} & \widehat{\mathtt{C}}\langle t, s\rangle[x\backslash t] \\ {\scriptstyle S}\downarrow & & \\ \widehat{\mathtt{C}}\langle x, s'\rangle[x\backslash t] & \xrightarrow{R/S} & \widehat{\mathtt{C}}\langle t, s'\rangle[x\backslash t] \end{array}$$

To conclude that $R/S \in \mathbb{HL}$ it suffices to observe that $\widehat{\mathtt{C}}\langle \Box, s'\rangle$ is a head context as a consequence of (A.11) and the fact that $\widehat{\mathtt{C}}\langle \Box, s\rangle$ is a head context.

1.2.3 *Internal to $H\langle\!\langle x \rangle\!\rangle$, above the hole of H.* Two cases, depending on the kind of redex of S:

1.2.3.1 *If S is a* `db` *redex.* Then the step S is of the form:

$$\mathtt{C}_1\langle(\lambda y.s)\mathtt{L}\, u\rangle[x\backslash t] \to \mathtt{C}_1\langle s[y\backslash u]\mathtt{L}\rangle[x\backslash t]$$

such that $H\langle\!\langle x \rangle\!\rangle = \mathtt{C}_1\langle(\lambda y.s)\mathtt{L}\, u\rangle$ and $\mathtt{C}_1$ is a prefix of H (*i.e.* $H = \mathtt{C}_1\langle \mathtt{C}_2\rangle$). So the hole of H can be either:

- *Internal to s.* This is impossible as head contexts do not go under abstractions.
- *Internal to one of the substitutions in* `L`. This is impossible as head contexts do not go inside substitutions.
- *Internal to u.* This is impossible as head contexts do not go to the right of applications.

1.2.3.2 *If S is a* `ls` *redex.* Then the step S is of the form:

$$\mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle y \rangle\!\rangle [y \backslash s] \rangle [x \backslash t] \to \mathtt{C}_1\langle \mathtt{C}_2\langle s \rangle [y \backslash s] \rangle [x \backslash t]$$

such that $H\langle\!\langle x \rangle\!\rangle = \mathtt{C}_1\langle \mathtt{C}_2\langle y \rangle [y \backslash s] \rangle$ and $\mathtt{C}_1$ is a prefix of H (*i.e.* $H = \mathtt{C}_1\langle \mathtt{C}_3 \rangle$). Note that the hole of H cannot be internal to s since head contexts cannot go inside substitutions, so the hole of H must be internal to $\mathtt{C}_2\langle\!\langle y \rangle\!\rangle$ The occurrences of x and y must be disjoint, since they are different variables and R and S are different `ls` redexes, so there must exist a two-hole context $\widehat{\mathtt{C}}$ such that:

$$\mathtt{C}_1\langle \widehat{\mathtt{C}}\langle \Box, y \rangle [y \backslash s] \rangle = H \quad \widehat{\mathtt{C}}\langle x, \Box \rangle = \mathtt{C}_2$$

Then:

$$\begin{array}{ccc} \mathtt{C}_1\langle \widehat{\mathtt{C}}\langle x, y \rangle [y \backslash s] \rangle [x \backslash t] & \xrightarrow{R} & \mathtt{C}_1\langle \widehat{\mathtt{C}}\langle t, y \rangle [y \backslash s] \rangle [x \backslash t] \\ \Big\downarrow{\scriptstyle S} & & \\ \mathtt{C}_1\langle \widehat{\mathtt{C}}\langle x, s \rangle [y \backslash s] \rangle [x \backslash t] & \xrightarrow{R/S} & \mathtt{C}_1\langle \widehat{\mathtt{C}}\langle t, s \rangle [y \backslash s] \rangle [x \backslash t] \end{array}$$

To conclude that $R/S \in \mathbb{HL}$ it suffices to observe that $\mathtt{C}_1\langle \widehat{\mathtt{C}}\langle \Box, s \rangle [y \backslash s] \rangle$ is a head context as a consequence of (A.11) and the fact that $\mathtt{C}_1\langle \widehat{\mathtt{C}}\langle \Box, y \rangle [y \backslash s] \rangle$ is a head context.

1.2.4 *Internal to t.* Then:

$$\begin{array}{ccc} H\langle\!\langle x \rangle\!\rangle [x \backslash t] & \xrightarrow{R} & H\langle t \rangle [x \backslash t] \\ \Big\downarrow{\scriptstyle S} & & \\ H\langle\!\langle x \rangle\!\rangle [x \backslash t'] & \xrightarrow{R/S} & H\langle t' \rangle [x \backslash t'] \end{array}$$

2. *Left of an application, $H = H'\, t$.* We argue that the step S cannot take place at the root. Suppose that S takes place at the root. Then it is a `db` step. Then $(\lambda x.s)\mathtt{L}$ must have a `db` or `ls` redex under the head context H'. Two cases:

   2.1 *If R is a* `db` *redex.* Then there must be an application node in $(\lambda x.s)\mathtt{L}$ under a head context. But head contexts do not go below abstractions or inside substitutions, so this is impossible.

   2.2 *If R is a* `ls` *redex.* Then there must be a variable in $(\lambda x.s)\mathtt{L}$ under a head context. But head contexts do not go below abstractions or inside substitutions, so this is impossible.

   Then the step S must take place either to the left of the application (and we conclude by *i.h.*) or to the right of the application (and then R and S are disjoint, so it is trivial).

3. *Left of a substitution, $H = H'[x \backslash t]$.* Three cases:

   3.1 *If S takes place at the root.* Then S must be a `ls` redex. Depending on the kind of redex of R:

3.1.1 *If R is a* `db` *redex.* Then R is of the form:

$$H'\langle(\lambda y.s)\mathtt{L}\,u\rangle[x\backslash t] \to H'\langle s[y\backslash u]\mathtt{L}\rangle[x\backslash t]$$

Four cases depending on the position of the contracted occurrence of x:

3.1.1.1 *The contracted occurrence of x is in H'.* Then there is a two-hole context $\widehat{\mathtt{C}}$ such that:

$$\widehat{\mathtt{C}}\langle\Box, x\rangle = H$$

and:

$$\begin{array}{ccc}\widehat{\mathtt{C}}\langle(\lambda y.s)\mathtt{L}\,u, x\rangle[x\backslash t] & \xrightarrow{R} & \widehat{\mathtt{C}}\langle s[y\backslash u]\mathtt{L}, x\rangle[x\backslash t] \\ \Big\downarrow{\scriptstyle S} & & \\ \widehat{\mathtt{C}}\langle(\lambda y.s)\mathtt{L}\,u, t\rangle[x\backslash t] & \xrightarrow{R/S} & \widehat{\mathtt{C}}\langle s[y\backslash u]\mathtt{L}, t\rangle[x\backslash t]\end{array}$$

To conclude that $R/S \in \mathbb{HL}$ it suffices to observe that $\widehat{\mathtt{C}}\langle\Box, t\rangle$ is a head context as a consequence of (A.11) and the fact that $\widehat{\mathtt{C}}\langle\Box, x\rangle$ is a head context.

3.1.1.2 *The contracted occurrence of x is in s.* Let $\widehat{s}$ denote the result of replacing the corresponding occurrence of x in s by t. Then:

$$\begin{array}{ccc}H'\langle(\lambda y.s)\mathtt{L}\,u\rangle[x\backslash t] & \xrightarrow{R} & H'\langle s[y\backslash u]\mathtt{L}\rangle[x\backslash t] \\ \Big\downarrow{\scriptstyle S} & & \\ H'\langle(\lambda y.\widehat{s})\mathtt{L}\,u\rangle[x\backslash t] & \xrightarrow{R/S} & H'\langle\widehat{s}[y\backslash u]\mathtt{L}\rangle[x\backslash t]\end{array}$$

3.1.1.3 *The contracted occurrence of x is in* `L`. Analogous to the previous case.

3.1.1.4 *The contracted occurrence of x is in u.* Analogous to the previous case.

3.1.2 *If R is a* `ls` *redex.* Then R is of the form:

$$H_1\langle H_2\langle\!\langle y\rangle\!\rangle[y\backslash s]\rangle[x\backslash t] \to H_1\langle H_2\langle s\rangle[y\backslash s]\rangle[x\backslash t]$$

with $H' = H_1\langle H_2[y\backslash s]\rangle$.

Three cases depending on the position of the contracted occurrence of x:

3.1.1 *The contracted occurrence of x is in H_1.* Then there is a two-hole context $\widehat{\mathtt{C}}$ such that:

$$\widehat{\mathtt{C}}\langle\Box, x\rangle = H_1$$

and:

$$\begin{array}{ccc}\widehat{\mathtt{C}}\langle H_2\langle\!\langle y\rangle\!\rangle[y\backslash s], x\rangle[x\backslash t] & \xrightarrow{R} & \widehat{\mathtt{C}}\langle H_2\langle s\rangle[y\backslash s], x\rangle[x\backslash t] \\ \Big\downarrow{\scriptstyle S} & & \\ \widehat{\mathtt{C}}\langle H_2\langle\!\langle y\rangle\!\rangle[y\backslash s], t\rangle[x\backslash t] & \xrightarrow{R/S} & \widehat{\mathtt{C}}\langle H_2\langle s\rangle[y\backslash s], t\rangle[x\backslash t]\end{array}$$

To conclude that $R/S \in \mathbb{HL}$ it suffices to observe that $\widehat{\mathtt{C}}\langle\Box, t\rangle$ is a head context as a consequence of (A.11) and the fact that $\widehat{\mathtt{C}}\langle\Box, x\rangle$ is a head context.

3.1.2 *The contracted occurrence of x is in H_2.* Analogous to the previous case.

3.1.3 *The contracted occurrence of x is in s.* Then $s = \mathtt{C}_1\langle\!\langle x \rangle\!\rangle$ and:

$$\begin{array}{ccc} H_1\langle H_2\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{C}_1\langle\!\langle x \rangle\!\rangle]\rangle[x\backslash t] & \xrightarrow{R} & H_1\langle H_2\langle \mathtt{C}_1\langle\!\langle x \rangle\!\rangle\rangle[y\backslash \mathtt{C}_1\langle\!\langle x \rangle\!\rangle]\rangle[x\backslash t] \\ \Big\downarrow{\scriptstyle S} & & \\ H_1\langle H_2\langle\!\langle y \rangle\!\rangle[y\backslash \mathtt{C}_1\langle t \rangle]\rangle[x\backslash t] & \xrightarrow{R/S} & H_1\langle H_2\langle \mathtt{C}_1\langle t \rangle\rangle[y\backslash \mathtt{C}_1\langle t \rangle]\rangle[x\backslash t] \end{array}$$

To conclude that $R/S \in \mathbb{HL}$ it suffices to observe that $H_1\langle H_2[y\backslash \mathtt{C}_1\langle t \rangle]\rangle$ is a head context as a consequence of (A.11) and the fact that $H_1\langle H_2[y\backslash \mathtt{C}_1\langle x \rangle]\rangle$ is a head context.

3.2 *If S takes place to the left of the substitution.* Then we conclude by *i.h.*.

3.3 *If S takes place to the right of the substitution.* Then R and S are disjoint, so it is trivial.

□

A.4.4 Need linear reduction is normalizing — proof of Coro. 7.59

Lemma A.114 (Properties of needed contexts)**.** *The following hold:*

1. **Answers have no redexes or variables under need contexts.**
   If $(\lambda x.s)\mathtt{L} = \mathtt{N}\langle\Delta\rangle$ then Δ is not a redex nor a free occurrence of a variable.

2. **Unique needed variable.**
   If $\mathtt{N}_1\langle\!\langle x \rangle\!\rangle = \mathtt{N}_2\langle\!\langle y \rangle\!\rangle$ then $\mathtt{N}_1 = \mathtt{N}_2$.

3. **Erasing a substitution in a need context.**
   If $\mathtt{N}_1\langle \mathtt{N}_2[x\backslash t]\rangle$ is a need context, then $\mathtt{N}_1\langle \mathtt{N}_2\rangle$ is also a need context.

4. **Replacing a term in a need context.**
   If $\widehat{\mathtt{C}}$ is a two-hole context, $\widehat{\mathtt{C}}\langle\Box, t\rangle$ is a need context, and t has no variables bound by $\widehat{\mathtt{C}}$, then $\widehat{\mathtt{C}}\langle\Box, s\rangle$ is also a need context (where s is an arbitrary term).

Proof. Item 1 is by induction on $\mathtt{L}$. Items 2 and 3 are by induction on $\mathtt{N}_1$. Item 4 is by induction on the formation of the need context $\widehat{\mathtt{C}}\langle\Box, t\rangle$. □

Corollary A.115 (Full proof of Coro. 7.59—Needed linear reduction is NLNF-normalizing)**.** *The strategy $\mathbb{S}_{\mathbb{NL}}$ associated to the sub-ARS $\mathbb{NL}$ is* NLNF*-normalizing.*

Proof. To show that $\mathbb{S}_{\mathbb{NL}}$ is NLNF-normalizing, we will apply Prop. 7.54 to conclude that $\mathbb{S}_{\mathbb{NL}}$ is $\mathsf{NF}(\mathbb{NL})$-normalizing. We must show that:

1. The set $\mathsf{NF}(\mathbb{NL})$ coincides with the set NLNF, so being $\mathsf{NF}(\mathbb{NL})$-normalizing is equivalent to being NLNF-normalizing. For this we will show the two inclusions, **(1a)** $\mathsf{NF}(\mathbb{NL}) \subseteq$ NLNF and **(1b)** NLNF $\subseteq \mathsf{NF}(\mathbb{NL})$.

2. The sub-ARS $\mathbb{NL}$ is closed residual-invariant, to be able to apply Prop. 7.54. For this we will show that **(2a)** the set $\mathsf{NF}(\mathbb{NL})$ is closed by reduction, and **(2b)** the sub-ARS $\mathbb{NL}$ is residual-invariant.

Part 1a: every $\mathbb{NL}$-normal form is a NLNF.
By induction on t it is straightforward to check that if $t \in \mathsf{NF}(\mathbb{NL})$ then $t \in \mathsf{NLNF}$.
Part 1b: every NLNF is a $\mathbb{NL}$-normal form.
Given $t \in \mathsf{NLNF}$ it can be shown that it is a $\mathbb{NL}$-normal form. There are two cases, depending on the shape of t. If t is an answer it is a direct consequence of Lem. A.114. If it is a structure, $t = \mathtt{N}\langle\!\langle x \rangle\!\rangle$, then it is straightforward by induction on $\mathtt{N}$.
Part 2a: the set $\mathsf{NF}(\mathbb{NL})$ is closed by reduction.
By items **(1a)** and **(1b)**, we know that $\mathsf{NF}(\mathbb{NL}) = \mathsf{NLNF}$. Let $t_1 \in \mathsf{NLNF}$ and let $t_1 \to t_2$ be an arbitrary step (not necessarily in the strategy). We claim that $t_2 \in \mathsf{NLNF}$. There are two cases, depending on the shape of t_1: if t_1 is an answer $(\lambda x.t)\mathtt{L}$, then by induction on $\mathtt{L}$ it can be seen that t_2 is also an answer. If t_1 is a structure $\mathtt{N}\langle\!\langle x \rangle\!\rangle$, then by induction on $\mathtt{N}$ it can be seen that t_2 is also of the form $\mathtt{N}'\langle\!\langle x \rangle\!\rangle$.
Part 2b: the sub-ARS $\mathbb{NL}$ is residual-invariant.
Let $R \in \mathbb{NL}$ and consider $R \neq S$. Let us show that there is a residual $R' \in \mathbb{NL} \cap R/S$. By induction on the need context $\mathtt{N}$ under which the step R takes place. Most overlappings between redexes R and S are uninteresting, and it is immediate to show that there is a residual $R' \in R/S$ in the strategy, resorting to Lem. A.114 when required. Below we deal with the interesting cases:

- `lsnl` **vs.** `ls` **at the root:** let $\widehat{\mathtt{C}}$ be a two-hole context such that $\widehat{\mathtt{C}}\langle\Box, x\rangle$ is a need context. Then:

$$\begin{array}{ccc} \mathtt{N}\langle\widehat{\mathtt{C}}\langle x, x\rangle[x\backslash \mathtt{vL}]\rangle & \xrightarrow{R} & \mathtt{N}\langle\widehat{\mathtt{C}}\langle \mathtt{vL}, x\rangle[x\backslash \mathtt{vL}]\rangle \\ \Big\downarrow{\scriptstyle S} & & \\ \mathtt{N}\langle\widehat{\mathtt{C}}\langle x, \mathtt{vL}\rangle[x\backslash \mathtt{vL}]\rangle & \xrightarrow{R/S} & \mathtt{N}\langle\widehat{\mathtt{C}}\langle \mathtt{vL}, \mathtt{vL}\rangle[x\backslash \mathtt{vL}]\rangle \end{array}$$

  To conclude that $R/S \in \mathbb{NL}$ it suffices to observe that $\widehat{\mathtt{C}}\langle\Box, t\rangle$ is a need context as a consequence of Lem. A.114.

- `lsnl` **vs.** `db` **above the variable:** that is, $R : \mathtt{N}_1\langle\mathtt{N}_2\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}']\rangle \to \mathtt{N}_1\langle\mathtt{N}_2\langle \mathtt{vL}'\rangle[x\backslash \mathtt{vL}']\rangle$ and $S : \mathtt{N}_1\langle\mathtt{N}'_2\langle(\lambda y.s)\mathtt{L}\, u\rangle[x\backslash \mathtt{vL}']\rangle \to \mathtt{N}_1\langle\mathtt{N}'_2\langle s[y\backslash u]\mathtt{L}\rangle[x\backslash \mathtt{vL}']\rangle$ such that the context $\mathtt{N}'_2$ is a prefix of the context $\mathtt{N}_2$, *i.e.* $\mathtt{N}_2 = \mathtt{N}'_2\langle\mathtt{N}''_2\rangle$. The variable x must lie somewhere inside the `db`-redex $(\lambda y.s)\mathtt{L}\, u$, below the need context $\mathtt{N}''_2$. But need contexts do not go below abstractions or to the right of applications, so this case is impossible.

- `lsnl` **vs.** `ls` **above the variable:** let $\widehat{\mathtt{C}}$ be a two-hole context such that $\widehat{\mathtt{C}}\langle\Box, y\rangle$ is a need context. Then:

$$\begin{array}{ccc} \mathtt{N}_1\langle\mathtt{N}_2\langle\widehat{\mathtt{C}}\langle x, y\rangle[y\backslash s]\rangle[x\backslash \mathtt{vL}]\rangle & \xrightarrow{R} & \mathtt{N}_1\langle\mathtt{N}_2\langle\widehat{\mathtt{C}}\langle \mathtt{vL}, y\rangle[y\backslash s]\rangle[x\backslash \mathtt{vL}]\rangle \\ \Big\downarrow{\scriptstyle S} & & \\ \mathtt{N}_1\langle\mathtt{N}_2\langle\widehat{\mathtt{C}}\langle x, s\rangle[y\backslash s]\rangle[x\backslash \mathtt{vL}]\rangle & \xrightarrow{R/S} & \mathtt{N}_1\langle\mathtt{N}_2\langle\widehat{\mathtt{C}}\langle \mathtt{vL}, s\rangle[y\backslash s]\rangle[x\backslash \mathtt{vL}]\rangle \end{array}$$

To conclude that $R/S \in \mathbb{NL}$ it suffices to observe that $\mathtt{N}_2\langle\widehat{\mathtt{C}}\langle\Box, s\rangle[y\backslash s]\rangle$ is a need context as a consequence of Lem. A.114.

- `lsnl` **vs.** `ls` **duplicating** R **on the needed position:** let x be bound to an answer $\mathtt{vL}$ either in $\mathtt{N}_1$ or in $\mathtt{N}_3$. Then:

$$\begin{array}{ccc} \mathtt{N}_1\langle\mathtt{N}_2\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{N}_3\langle x\rangle]\rangle & \xrightarrow{R} & \mathtt{N}_1\langle\mathtt{N}_2\langle\!\langle y\rangle\!\rangle[y\backslash\mathtt{N}_3\langle\mathtt{vL}\rangle]\rangle \\ \downarrow{\scriptstyle S} & & \\ \mathtt{N}_1\langle\mathtt{N}_2\langle\mathtt{N}_3\langle x\rangle\rangle[y\backslash\mathtt{N}_3\langle x\rangle]\rangle & \xrightarrow{R_1} & \mathtt{N}_1\langle\mathtt{N}_2\langle\mathtt{N}_3\langle\mathtt{vL}\rangle\rangle[y\backslash\mathtt{N}_3\langle x\rangle]\rangle \end{array}$$

  Note that R_1 is one of the two residuals of R, and $R_1 \in \mathbb{NL}$.

- `lsnl` **vs.** `ls` **duplicating** R **on a non-needed position:** let x be bound to an answer $\mathtt{vL}$ either in $\mathtt{N}_1$ or in $\mathtt{N}_2$, and let $\widehat{\mathtt{C}}$ be a two-hole context such that $\widehat{\mathtt{C}}\langle\Box, y\rangle$ is a need context. Then:

$$\begin{array}{ccc} \mathtt{N}_1\langle\widehat{\mathtt{C}}\langle y, y\rangle[y\backslash\mathtt{N}_2\langle x\rangle]\rangle & \xrightarrow{R} & \mathtt{N}_1\langle\widehat{\mathtt{C}}\langle y, y\rangle[y\backslash\mathtt{N}_2\langle\mathtt{vL}\rangle]\rangle \\ \downarrow{\scriptstyle S} & & \\ \mathtt{N}_1\langle\widehat{\mathtt{C}}\langle y, \mathtt{N}_2\langle x\rangle\rangle[y\backslash\mathtt{N}_2\langle x\rangle]\rangle & \xrightarrow{R_1} & \mathtt{N}_1\langle\widehat{\mathtt{C}}\langle y, \mathtt{N}_2\langle x\rangle\rangle[y\backslash\mathtt{N}_2\langle\mathtt{vL}\rangle]\rangle \end{array}$$

  To conclude that $R_1 \in \mathbb{NL}$ it suffices to observe that $\widehat{\mathtt{C}}\langle\Box, \mathtt{N}_2\langle x\rangle\rangle$ is a need context as a consequence of Lem. A.114.

- `lsnl` **vs. step internal to the argument:** Let $\mathtt{vL} \to t$ be a step. By Part 2a, the set of $\mathbb{S}_{NL}$-normal forms is closed by reduction and, more specifically, the set of answers is closed by reduction. So $t = \mathtt{v'L'}$. Then:

$$\begin{array}{ccc} \mathtt{N}_1\langle\mathtt{N}_2\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}]\rangle & \xrightarrow{R} & \mathtt{N}_1\langle\mathtt{N}_2\langle t\rangle[x\backslash\mathtt{vL}]\rangle \\ \downarrow{\scriptstyle S} & & \\ \mathtt{N}_1\langle\mathtt{N}_2\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{v'L'}]\rangle & \xrightarrow{R/S} & \mathtt{N}_1\langle\mathtt{N}_2\langle\mathtt{v'L'}\rangle[x\backslash\mathtt{v'L'}]\rangle \end{array}$$

□

Bibliography

[1] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.

[2] Beniamino Accattoli. *Jumping around the box: Graphical and operational studies on λ-calculus and Linear Logic*. PhD thesis, Universitá degli Studi di Roma "La Sapienza", december 2010.

[3] Beniamino Accattoli. An abstract factorization theorem for explicit substitutions. In *23rd International Conference on Rewriting Techniques and Applications (RTA'12), May 28 - June 2, 2012, Nagoya, Japan*, pages 6–21, 2012.

[4] Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In J. Jeuring and M. Chakravarty, editors, *Proceedings of ICFP*, pages 363–376. ACM, 2014.

[5] Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. *ACM SIGPLAN Notices*, 49(1):659–670, 2014.

[6] Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 659–670. ACM, 2014.

[7] Beniamino Accattoli and Giulio Guerrieri. Abstract machines for open call-by-value. *Sci. Comput. Program.*, 184, 2019.

[8] Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2010.

[9] Beniamino Accattoli and Delia Kesner. The structural λ-calculus. In *International Workshop on Computer Science Logic*, pages 381–395. Springer, 2010.

[10] Beniamino Accattoli and Delia Kesner. Preservation of Strong Normalisation modulo permutations for the structural lambda-calculus. *Logical Methods in Computer Science*, 8(1):44, March 2012.

[11] Beniamino Accattoli and Ugo Dal Lago. Beta reduction is invariant, indeed. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 8:1–8:10. ACM, 2014.

[12] Zena M Ariola and Matthias Felleisen. The call-by-need lambda calculus. *Journal of functional programming*, 7(3):265–301, 1997.

[13] Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. A call-by-need lambda calculus. In *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, pages 233–246, 1995.

[14] Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*. Cambridge Tracts in Theoretical Computer Science. CUP, 1999.

[15] Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the lambda-calculus, 1995.

[16] Andrea Asperti and Harry G. Mairson. Parallel beta reduction is not elementary recursive. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '98, pages 303–315, New York, NY, USA, 1998. ACM.

[17] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1999.

[18] John W Backus, Robert J Beeber, Sheldon Best, Richard Goldberg, L Mitchell Haibt, Harlan L Herrick, Robert A Nelson, David Sayre, Peter B Sheridan, H Stern, et al. The fortran automatic coding system. In *Papers presented at the February 26-28, 1957, western joint computer conference: Techniques for reliability*, pages 188–198. ACM, 1957.

[19] Thibaut Balabonski. Optimality for dynamic patterns. In *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, PPDP '10, pages 231–242, New York, NY, USA, 2010. ACM.

[20] Thibaut Balabonski. Weak optimality, and the meaning of sharing. *SIGPLAN Not.*, 48(9):263–274, September 2013.

[21] Hendrik Pieter Barendregt, Jan A. Bergstra, Jan Willem Klop, and Henri Volken. Some notes on lambda reduction. Technical Report 22, University of Utrecht, Department of Mathematics, 1976.

[22] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103. Elsevier, 1984.

[23] Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda calculus with types*. Cambridge University Press, 2013.

[24] Zine-El-Abidine Benaissa, Daniel Briaud, Pierre Lescanne, and Jocelyne Rouyer-Degli. $\lambda\nu$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996.

[25] Alexis Bernadet. *Types intersections non-idempotents pour rafiner la normalisation forte avec des informations quantitatives*. PhD thesis, École Polytechnique, 2014.

[26] Gérard Berry. Stable models of typed λ-calculi. In *International Colloquium on Automata, Languages, and Programming*, pages 72–89. Springer, 1978.

[27] Gérard Berry and Jean-Jacques Lévy. Minimal and optimal computations of recursive programs. *J. ACM*, 26(1):148–175, January 1979.

[28] Malgorzata Biernacka and Olivier Danvy. A concrete framework for environment machines. *ACM Trans. Comput. Log.*, 9(1), 2007.

[29] Roel Bloo and Kristoffer H Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *In CSN-95: Computer Science in the Netherlands*. Citeseer, 1995.

[30] Sander Bruggink, Dimitri Hendriks, Vincent van Oostrom, and Roel de Vrijer. Optimal implementation of higher-order rewriting, 2002.

[31] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017.

[32] Rod M Burstall, David B MacQueen, and Donald T Sannella. Hope: An experimental applicative language. In *Proceedings of the 1980 ACM conference on LISP and functional programming*, pages 136–143. ACM, 1980.

[33] Felice Cardone and J. Roger Hindley. J.r.: History of lambda-calculus and combinatory logic. In *Handbook of the History of Logic, volume 5: Logic*, 2006.

[34] Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. PhD thesis, Ecole Doctorale Physique et Sciences de la Matière (Marseille), 2007.

[35] Stephen Chang and Matthias Felleisen. The call-by-need lambda calculus, revisited. In *European Symposium on Programming*, pages 128–147. Springer, 2012.

[36] Alonzo Church. A set of postulates for the foundation of logic part i. *Annals of Mathematics*, 33(2):346–366, 1932. http://www.jstor.org/stable/1968702Electronic Edition.

[37] Alonzo Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(02):56–68, 1940.

[38] Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Arch. Math. Log.*, 19(1):139–156, 1978.

[39] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.

[40] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Mathematical Logic Quarterly*, 27(2-6):45–58, 1981.

[41] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and computation*, 76(2-3):95–120, 1988.

[42] Pierre Crégut. Strongly reducing variants of the krivine abstract machine. *Higher-Order and Symbolic Computation*, 20(3):209–230, 2007.

[43] Pierre-Louis Curien. An abstract framework for environment machines. *Theoretical Computer Science*, 82(2):389–402, 1991.

[44] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *ACM sigplan notices*, volume 35, pages 233–243. ACM, 2000.

[45] Haskell B. Curry. The combinatory foundations of mathematical logic. *The Journal of Symbolic Logic*, 7(2):49–64, 1942.

[46] H.B. Curry and R. Feys. *Combinatory Logic*. Number v. 1 in Combinatory Logic. North-Holland Publishing Company, 1958.

[47] Vincent Danos and Laurent Regnier. Head linear reduction. Technical report, 2004.

[48] Olivier Danvy. A rational deconstruction of landin's secd machine. In *IFL*, pages 52–71, 2004.

[49] Olivier Danvy and Ian Zerny. A synthetic operational account of call-by-need evaluation. In *PPDP*, pages 97–108, 2013.

[50] Nicolaas Govert De Bruijn. The mathematical language automath, its usage, and some of its extensions. In *Symposium on automatic demonstration*, pages 29–61. Springer, 1970.

[51] Nicolaas Govert de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. 1978.

[52] Flávio L. C. de Moura, Delia Kesner, and Mauricio Ayala-Rincón. Metaconfluence of Calculi with Explicit Substitutions at a Distance. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, volume 29 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 391–402, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[53] Roel De Vrijer. A direct proof of the finite developments theorem. *The Journal of symbolic logic*, 50(2):339–343, 1985.

[54] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.

[55] Matthias Felleisen. *The Calculi of Lambda-Nu-CS Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages.* PhD thesis, 1988.

[56] Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the lambda-calculus. In *3rd Working Conference on the Formal Description of Programming Concepts*, August 1986.

[57] Mattias Felleisen and Daniel P Friedman. A calculus for assignments in higher-order languages. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, page 314. ACM, 1987.

[58] Philippa Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software*, pages 555–574. Springer, 1994.

[59] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris 7, 1972.

[60] Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

[61] John R. W. Glauert and Zurab Khasidashvili. Relative normalization in deterministic residual structures. In Hélène Kirchner, editor, *Trees in Algebra and Programming - CAAP'96, 21st International Colloquium, Linköping, Sweden, April, 22-24, 1996, Proceedings*, volume 1059 of *Lecture Notes in Computer Science*, pages 180–195. Springer, 1996.

[62] Georges Gonthier. The four colour theorem: Engineering of a formal proof. In *Computer Mathematics, 8th Asian Symposium, ASCM 2007, Singapore, December 15-17, 2007. Revised and Invited Papers*, page 333, 2007.

[63] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '92, pages 15–26, New York, NY, USA, 1992. ACM.

[64] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the odd order theorem. In *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, pages 163–179, 2013.

[65] Mike Gordon. Proof, language, and interaction. chapter From LCF to HOL: A Short History, pages 169–185. MIT Press, Cambridge, MA, USA, 2000.

[66] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. *ACM SIGPLAN Notices*, 37(9):235–246, 2002.

[67] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In *ICFP '02, Pittsburgh, Pennsylvania, USA, October 4-6, 2002.*, pages 235–246, 2002.

[68] Timothy G Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–58. ACM, 1989.

[69] Stefano Guerrini. A general theory of sharing graphs. Technical report, Theoret. Comput. Sci, 1998.

[70] Stefano Guerrini and Marco Solieri. Is the optimal implementation inefficient? elementarily not. In *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84, pages 17–1, 2017.

[71] Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the kepler conjecture. *CoRR*, abs/1501.02155, 2015.

[72] Thérèse Hardin. Confluence results for the pure strong categorical logic ccl. λ-calculi as subsystems of ccl. *Theoretical computer science*, 65(3):291–342, 1989.

[73] Thérèse Hardin and Luc Maranget. Functional runtime systems within the lambda-sigma calculus. *J. Funct. Program.*, 8(2):131–176, 1998.

[74] J Roger Hindley and Jonathan P Seldin. *Lambda-calculus and combinators: an introduction*, volume 13.

[75] Roger Hindley. Reductions of residuals are finite. *Transactions of the American Mathematical Society*, 240:345–361, 1978.

[76] W. A. Howard. *The formulae-as-types notion of construction*, pages 480–490. Academic Press, London-New York, 1980.

[77] Gérard Huet. The zipper. *Journal of functional programming*, 7(5):549–554, 1997.

[78] Gérard P. Huet and Jean-Jacques Lévy. Computations in orthogonal rewriting systems, I. In *Computational Logic - Essays in Honor of Alan Robinson*, pages 395–414, 1991.

[79] Gérard P. Huet and Jean-Jacques Lévy. Computations in orthogonal rewriting systems, II. In *Computational Logic - Essays in Honor of Alan Robinson*, pages 415–443, 1991.

[80] John Hughes. Why functional programming matters. *The computer journal*, 32(2):98–107, 1989.

[81] Kenneth E Iverson. A programming language. In *Proceedings of the May 1-3, 1962, spring joint computer conference*, pages 345–351. ACM, 1962.

[82] Richard Jones, Antony Hosking, and Eliot Moss. *The garbage collection handbook: the art of automatic memory management*. Chapman and Hall/CRC, 2016.

[83] Simon Peyton Jones. *Haskell 98 language and libraries: the revised report*. Cambridge University Press, 2003.

[84] Fairouz Kamareddine and Alejandro Ríos. A λ-calculus à la de bruijn with explicit substitutions. In *International Symposium on Programming Language Implementation and Logic Programming*, pages 45–62. Springer, 1995.

[85] Delia Kesner. The theory of calculi with explicit substitutions revisited. In *International Workshop on Computer Science Logic*, pages 238–252. Springer, 2007.

[86] Delia Kesner. Reasoning about call-by-need by means of types. In *Proceedings of the 19th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 424–441. Springer-Verlag, 2016.

[87] Delia Kesner. Reasoning about call-by-need by means of types. In *International Conference on Foundations of Software Science and Computation Structures*, pages 424–441. Springer, 2016.

[88] Delia Kesner and Stéphane Lengrand. Extending the explicit substitution paradigm. In *International Conference on Rewriting Techniques and Applications*, pages 407–422. Springer, 2005.

[89] Delia Kesner and Fabien Renaud. The prismoid of resources. In *International Symposium on Mathematical Foundations of Computer Science*, pages 464–476. Springer, 2009.

[90] Delia Kesner, Alejandro Ríos, and Andrés Viso. Call-by-need, neededness and all that. In *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, pages 241–257, 2018.

[91] Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus. In *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference (TCS'14), Rome, Italy, September 1-3, 2014.*, pages 296–310, 2014.

[92] Assaf J Kfoury. A linearization of the lambda-calculus and consequences. 2000.

[93] Assaf J Kfoury and Joe B Wells. Principality and type inference for intersection types using expansion variables. *Theoretical Computer Science*, 311(1-3):1–70, 2004.

[94] Donald E Knuth. Ancient babylonian algorithms. *Communications of the ACM*, 15(7):671–677, 1972.

[95] Jean Louis Krivine. *Lambda-calculus, Types and Models*. Computers and their applications. Ellis Horwood, 1993.

[96] Jean-Louis Krivine. *Lambda-calculus, types and models*. Ellis Horwood, 1993.

[97] Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-order and symbolic computation*, 20(3):199–207, 2007.

[98] John Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 16–30. ACM Press, 1990.

[99] Peter J Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

[100] Peter J Landin. The next 700 programming languages. *Communications of the ACM*, 9(3):157–166, 1966.

[101] Cosimo Laneve. *Optimality and Concurrency in Interaction Systems*. PhD thesis, Dipartimento di Informatica, Università di Pisa, august 1993.

[102] Cosimo Laneve. Distributive evaluations of λ-calculus. *Fundamenta Informaticae*, 20(4):333–352, 1994.

[103] Frédéric Lang. Explaining the lazy krivine machine using explicit substitution and addresses. *Higher-Order and Symbolic Computation*, 20(3):257–270, 2007.

[104] Julia L. Lawall and Harry G. Mairson. Optimality and inefficiency: What isn't a cost model of the lambda calculus? In *Proceedings of the First ACM SIGPLAN International Conference on Functional Programming*, ICFP '96, pages 92–101, New York, NY, USA, 1996. ACM.

[105] Julia L. Lawall and Harry G. Mairson. On global dynamics of optimal graph reduction. In *1997 ACM International Conference on Functional Programming*, pages 188–195, 1997.

[106] Xavier Leroy. *The ZINC experiment: an economical implementation of the ML language*. PhD thesis, INRIA, 1990.

[107] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009.

[108] Pierre Lescanne and Jocelyne Rouyer-Degli. Explicit substitutions with de bruijn's levels. In *International Conference on Rewriting Techniques and Applications*, pages 294–308. Springer, 1995.

[109] Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul*. PhD thesis, Université de Paris 7, 1978.

[110] Jean-Jacques Lévy. Optimal reductions in the lambda calculus. *Essays on Combinatory Logic, Lambda Calculus and Formalism*, 1980.

[111] Jean-Jacques Levy. Redexes are stable in the λ-calculus. 27:1–13, 07 2015.

[112] Barry James Mailloux, John Edward Lancelot Peck, Cornelis HA Koster, and A Van Wijngaarden. Report on the algorithmic language algol 68. In *Report on the algorithmic language Algol 68*, pages 80–218. Springer, 1969.

[113] John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. *Journal of functional programming*, 8(3):275–317, 1998.

[114] Per Martin-Löf. A theory of types, 1971.

[115] Gianfranco Mascari and Marco Pedicini. Head linear reduction and pure proof net extraction. *Theor. Comput. Sci.*, 135(1):111–137, 1994.

[116] John McCarthy and Michael I Levin. *LISP 1.5 programmer's manual*. 1965.

[117] Paul-André Mellies. Typed λ-calculi with explicit substitutions may not terminate. In *International Conference on Typed Lambda Calculi and Applications*, pages 328–334. Springer, 1995.

[118] Paul-André Melliès. *Description Abstraite des Systèmes de Réécriture*. PhD thesis, Université Paris 7, december 1996.

[119] Robin Milner. *The definition of standard ML: revised*. 1997.

[120] Robin Milner. Local bigraphs and confluence: Two conjectures. *Electronic Notes in Theoretical Computer Science*, 175(3):65–73, 2007.

[121] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[122] Ulf Norell. Dependently typed programming in agda. In *Proceedings of the 4th international workshop on Types in language design and implementation*, pages 1–2. ACM, 2009.

[123] Chris Okasaki. *Purely functional data structures*. Cambridge University Press, 1999.

[124] Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Logic programming and automated reasoning*, pages 190–201. Springer, 1992.

[125] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical computer science*, 1(2):125–159, 1975.

[126] John C Reynolds. Towards a theory of type structure. In *Programming Symposium*, pages 408–425. Springer, 1974.

[127] Kristoffer Høgsbro Rose. Explicit cyclic substitutions. In *International Workshop on Conditional Term Rewriting Systems*, pages 36–50. Springer, 1992.

[128] Bertrand Russell. *The principles of mathematics*. WW Norton & Company, 1938.

[129] Peter Sestoft. Deriving a lazy abstract machine. *Journal of Functional Programming*, 7(3):231–264, 1997.

[130] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*, volume 149. Elsevier, 2006.

[131] Michael Sperber, R Kent Dybvig, Matthew Flatt, Anton Van Straaten, Robby Findler, and Jacob Matthews. Revised6 report on the algorithmic language scheme. *Journal of Functional Programming*, 19(S1):1–301, 2009.

[132] John Staples. Efficient combinatory reduction. *Mathematical Logic Quarterly*, 27(2530):391–402, 1981.

[133] Christopher Strachey. Fundamental concepts in programming languages. *Higher Order Symbol. Comput.*, 13(1-2):11–49, April 2000.

[134] Gerald Jay Sussman and Guy L Steele Jr. Scheme: An interpreter for extended lambda calculus. In *MEMO 349, MIT AI LAB*, 1975.

[135] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[136] Alan M. Turing. On computable numbers, with an application to the entscheidungs problem. *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.

[137] David Turner. An overview of miranda 1. In *Research Topics in Functional Programming*, pages 1–16. Addison-Wesley Longman Publishing Co., Inc., 1990.

[138] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[139] Steffen van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102(1):135–163, 1992.

[140] Peter van Emde Boas. Machine models and simulations. 1989.

[141] Vincent van Oostrom. *Higher-order families*, pages 392–407. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[142] Vincent Van Oostrom and Roel De Vrijer. Four equivalent equivalences of reductions. *Electronic Notes in Theoretical Computer Science*, 70(6):21–61, 2002.

[143] Jean Etienne Vuillemin. *Proof-techniques for Recursive Programs.* PhD thesis, Stanford University, Stanford, CA, USA, 1974. AAI7413700.

[144] Jean Etienne Vuillemin. *Syntaxe, sémantique et axiomatique d'un langage de programmation simple.* PhD thesis, Université de Paris VII, 1975.

[145] Christopher P. Wadsworth. *Semantics and Pragmatics of the Lambda Calculus*. PhD thesis, Oxford University, 1971.