Foundations of Strong Call by Need*

THIBAUT BALABONSKI, LRI, Université Paris-Sud, CNRS, Université Paris-Saclay
PABLO BARENBAUM, Universidad Buenos Aires and IRIF, CNRS and Université Paris-Diderot
EDUARDO BONELLI, CONICET, Universidad Nacional de Quilmes, and Stevens Institute of Technology
DELIA KESNER, IRIF, CNRS and Université Paris-Diderot

We present a *call-by-need strategy* for computing *strong* normal forms of open terms (reduction is admitted inside the body of abstractions and substitutions, and the terms may contain free variables), which guarantees that arguments are only evaluated when needed and at most once. The strategy is shown to be *complete* with respect to β-reduction to strong normal form. The proof of completeness relies on two key tools: (1) the definition of a strong call-by-need *calculus* where reduction may be performed inside *any* context, and (2) the use of non-idempotent *intersection types*. More precisely, terms admitting a β-normal form in pure lambda calculus are typable, typability implies (weak) normalisation in the strong call-by-need calculus, and weak normalisation in the strong call-by-need calculus implies normalisation in the strong call-by-need strategy. Our (strong) call-by-need strategy is also shown to be *conservative* over the standard (weak) call-by-need.

CCS Concepts: • **Theory of computation → Program semantics**; **Operational semantics**;

Additional Key Words and Phrases: Evaluation Strategies, Call-by-Need, Completeness

ACM Reference format:
Thibaut Balabonski, Pablo Barenbaum, Eduardo Bonelli, and Delia Kesner. 2017. Foundations of Strong Call by Need. *Proc. ACM Program. Lang.* 1, ICFP, Article 20 (September 2017), 29 pages.
https://doi.org/10.1145/3110264

1 INTRODUCTION

Weak reduction, the standard model of computation in functional programming languages like ML, OCaml and Haskell, does not reduce inside abstractions because they are considered *values*. Several evaluation strategies have been introduced to compute values. In particular, (weak) *call-by-need* strategies, originally introduced by Wadsworth [1971], focus on two main points: evaluate the argument of a function application only when that argument is found to be needed (which happens during the evaluation of the body of the applied function), and memoize the value of the argument so that further accesses to the argument may simply look up the value.

In contrast to weak reduction, *strong reduction* evaluates inside the bodies of abstractions (*i.e.* evaluates open terms containing free variables), and reaches the normal forms of λ-calculus, which we call *strong normal forms* in this paper for clarity. Strong reduction is required when implementing proof assistants, especially those based on type theory, such as Agda or Coq, which decide the definitional equality of functions by carrying out symbolic computations. Grégoire and Leroy

*This work was partially funded by the French-Argentinian International Laboratory INFINIS.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
2475-1421/2017/9-ART20
https://doi.org/10.1145/3110264

[2002] have proposed a strong normalisation function $\mathcal{N}$ which they have proved to be correct, in the sense that $\mathcal{N}$ computes the normal form of strongly normalising, closed λ-terms. This function essentially consists in iterating the standard (weak) *call-by-value* (CBV) strategy on terms possibly containing free variables, Grégoire and Leroy [2002] refers to this variant of CBV as *symbolic* CBV.

The starting point of this work is the observation that rather than iterating CBV, one should consider an appropriate notion of *call-by-need* (CBNd) that computes strong normal forms (of open terms). In this paper we replace *symbolic* CBV by *symbolic* CBNd, consisting in iterating the standard CBNd strategy on terms possibly containing free variables. Our strategy computes strong normal forms in which, in contrast to $\mathcal{N}$, arguments are evaluated only if they are needed and, moreover, are evaluated at most once thus avoiding duplication of work. For example, the function $\mathcal{N}$ in Grégoire and Leroy [2002] computes the value of the argument $(\lambda z.z)(\lambda z.z)$ in the λ-term $(\lambda x.\lambda y.(\lambda z.z)y)((\lambda z.z)(\lambda z.z))$ even though this value is not required for the strong normal form of the whole term, whereas our strategy will not. Also, our strategy is normalising, *i.e.* it computes the normal form of *weakly* normalising terms, that is, of terms that admit a normal form but whose evaluation may diverge along some other strategies.

Defining Strong Call-by-Need. Some of the subtleties involved in developing a theory of CBNd to strong normal form are illustrated next. In what follows, we write $\rightsquigarrow$ for denoting the CBNd strategy to strong normal form devised in this paper and motivated below.

Consider a term $(\lambda x.t)(\mathsf{id}\ \mathsf{id})$, where id abbreviates the identity term $\lambda z.z$ and t is an arbitrary subterm (in Fig. 1, t is chosen to be $\lambda y.yxx$). The first reduction step for a term of this shape is a common (weak) call-by-need step: the β-redex $(\lambda x.t)(\mathsf{id}\ \mathsf{id})$ is turned into an explicit binding between the variable x and the argument $\mathsf{id}\ \mathsf{id}$ in the expression t, which is often written $\mathtt{let}\ x = (\mathsf{id}\ \mathsf{id})\ \mathtt{in}\ t$, or here $t[x\backslash \mathsf{id}\ \mathsf{id}]$, where $[x\backslash \mathsf{id}\ \mathsf{id}]$ is called an *explicit substitution*. The content $\mathsf{id}\ \mathsf{id}$ of this explicit binding will be substituted later for some occurrences of the variable x in the expression t. If there are several occurrences of the variable x in t, the substitutions will be performed one by one, and can be performed in any order. This substitution mechanism, together with the reduction of β-redexes, results in a non-deterministic reduction mechanism: a *calculus*. However, we want to build here a more precise, deterministic way of applying the reduction rules: a *strategy*. Our goal then consists in choosing an order to perform the different reduction steps, by guaranteeing at the same time that all these steps are really *needed*.

Fig. 1 A reduction example

$$
\begin{array}{lll}
(\lambda x.\lambda y.yxx)(\mathsf{id}\ \mathsf{id}) & & \\
\rightsquigarrow & (\lambda y.yxx)[x\backslash \mathsf{id}\ \mathsf{id}] & (1)\\
\rightsquigarrow & (\lambda y.yxx)[x\backslash z[z\backslash \mathsf{id}]] & (2)\\
\rightsquigarrow & (\lambda y.yxx)[x\backslash \mathsf{id}[z\backslash \mathsf{id}]] & (3)\\
\rightsquigarrow & (\lambda y.y\ \mathsf{id}\ x)[x\backslash \mathsf{id}][z\backslash \mathsf{id}] & (4)\\
\rightsquigarrow & (\lambda y.y\ \mathsf{id}\ \mathsf{id})[x\backslash \mathsf{id}][z\backslash \mathsf{id}] & (5)
\end{array}
$$

As is standard, the notion of value is generalized to that of *answer*, essentially a value embraced by let-constructs (or, here, explicit substitutions). An example of an answer is $(\lambda y.yxx)[x\backslash \mathsf{id}\ \mathsf{id}]$, obtained after the first step of our example in Fig. 1, where computation on $\mathsf{id}\ \mathsf{id}$ is left pending since it is not required (yet).

Note that in *weak* CBNd, the term $(\lambda y.yxx)[x\backslash \mathsf{id}\ \mathsf{id}]$ is considered to be in normal form. In the *strong* case, however, reduction has to continue in order to complete the evaluation of the term, and an appropriate notion of evaluation context is required to allow this to happen. In this case, $\lambda y.yxx$ would be decomposed as an expression of the form $E[x]$, where E is an evaluation context that reaches an occurrence of x; this would trigger reduction inside the target $\mathsf{id}\ \mathsf{id}$ of the substitution.

More precisely, one can observe that the λ-abstraction $\lambda y.yxx$ will never be applied to an argument. This has two consequences: reduction can focus on the body yxx, and the variable y will never be substituted and may be considered a *constant* for all practical reasons. Therefore, yxx may

be seen as a *structure* headed by a constant y, whose two arguments have to be evaluated (in some order deterministically chosen by the strategy). Here, the notion of CBNd reduction developed in Sec. 2 will choose the evaluation context E to be of the form $\lambda y.y\Box x$, thus focusing on the first occurrence of x.

Then, we can fall back on the usual principles of call-by-need: we focus on an occurrence of a variable x that is bound to the term $\mathtt{id}\ \mathtt{id}$ which is not yet an *answer* (the only kind of term that can be substituted) but can still keep reducing: this triggers evaluation in the target of the substitution $[x\backslash \mathtt{id}\ \mathtt{id}]$; which is step (2) above. Step (3) follows a similar argument and, once completed, the target of the substitution, namely $\mathtt{id}[z\backslash \mathtt{id}]$ is an *answer*. The leftmost occurrence of x in yxx can now be substituted with this answer; as indicated in step (4) above. At this point, evaluation of the first argument of the structure yxx is complete: the strategy will now focus on the second one, which is an occurrence of x that can be substituted immediately (as indicated in step (5)) since it is bound to a term that has already been evaluated.

In summary, it is fundamental to devise an appropriate notion of evaluation context which allows reduction not only under abstractions but also inside *structures*. More precisely, we consider φ-structures, informally, structures headed by variables belonging to a set φ. An example is xt where x is a variable considered as a *constant* in the set φ, and t is in normal form. However, there are other forms of (φ-) structures that involve explicit substitutions (*cf.* Sec. 2 for a fine analysis of structures). Thus, the strong call-by-need strategy that we define in this paper is somewhat subtle in that it needs to be parameterized by a set of variables φ locally considered as constants (hence written $\overset{\varphi}{\rightsquigarrow}$ rather than $\rightsquigarrow$).

Completeness. Our strategy should be complete with respect to β-reduction in the sense that whenever a term admits a β-normal form u, then 1) the strategy computes a normal form too; and 2) the normal form computed by the strategy should correspond to u in some precise sense.

A direct proof of this result is non-trivial because of the difficulties in establishing a correlation between β-steps and reduction steps in our strategy. More precisely, β-reduction may make copies of its (unevaluated!) argument and then reduce (zero, one or more of) these copies much later than they are actually needed, if at all. For example, the β-reduction $(\lambda x.\lambda y.yxx)(\mathtt{id}\ \mathtt{id}) \to_\beta \lambda y.y(\mathtt{id}\ \mathtt{id})(\mathtt{id}\ \mathtt{id}) \to_\beta \lambda y.y\ (\mathtt{id}\ \mathtt{id})\ \mathtt{id} \to_\beta \lambda y.y\ \mathtt{id}\ \mathtt{id}$ would have to be related to the example derivation in Fig. 1 given above.

A first completeness result for weak call-by-need appears in Ariola, Felleisen, Maraist, Odersky, and Wadler [1995]. The proof is quite involved because it makes use of different syntactical tools such as sharing, residual theory and standardization. A more abstract proof is developed by Kesner [2016]; it is shown that every lambda-term that can be reduced to a weak normal-form is typable in an appropriate typing system with intersection types, and that every typable term is normalising in the weak call-by-need calculus. In this paper we adopt similar ideas in order to develop a completeness proof for strong call-by-need.

Potential Application to Proof Assistants. The results developed in this work may find applicability in proof assistants based on dependent types. Theories of dependent types allow types to depend on terms. The paradigmatic example is the case of vectors. The type expression `Vector::*→Nat→*` denotes a type family of vectors. In particular, `Vector Bool n` denotes vectors of booleans of length `n`. Consider the following function for appending vectors:

```
append:: Πα::*.Πn::Nat.Πm::Nat.Vector α m → Vector α n → Vector α (m+n)
append nil w = w
append (cons x v) w = cons x (append v w)
```

Type checking this definition requires checking, for each clause, that the left and right hand sides have the same type. In the case of the first clause, its LHS has type `Vector α (0+n)` whereas its RHS has type `Vector α n`. Determining that these types are equal involves identifying the *terms* `0+n` and `n`. This may be achieved by evaluating `0+n` on the one hand and `n` on the other to determine whether the same normal form is reached. This example is captured by means of a general 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, such as when equivalence is taken to be definitional equality, this is implemented by evaluating all the terms on which types depend to normal form and then comparing them. Since we are dealing with a notion of equality, this comparison takes into account any subterm inside λ-abstractions, and all the terms must be evaluated to *strong normal form*, whether they contain free variables (*i.e.* they are open) or not (*i.e.* they are closed). In fact, in algorithmic presentations of dependent types, it turns out to be more reasonable, for reasons of efficiency, to first compute weak head normal forms, which exposes the head construct, and then iterate the process over the arguments if these constructs agree. One such proof assistant that adopts this notion of type conversion is Coq, whose current implementation by Barras [2017] uses a call-by-need like strategy with finely tuned heuristics that are important in practice for the type-checker performance. The understanding of such current implementation could benefit from our foundational work.

Contributions and Structure of the Paper. The main contributions of the paper are:

- The definition of the *strong call-by-need strategy* for open terms (Sec. 2).
- A proof of *completeness* of the strong call-by-need strategy with respect to β-reduction (Sec. 6). In particular, this proof introduces:
  - A *strong call-by-need calculus* (Sec. 3) for which the strong call-by-need strategy is proved to be normalising (Sec. 5).
  - The use of *non-idempotent intersection types* to relate weak normalisation in the λ-calculus with weak normalisation in the strong call-by-need calculus (Sec. 4).
- A proof that our strong call-by-need strategy is *conservative* over weak call-by-need strategies (Sec. 7).

We discuss related works in Sec. 8 and conclude in Sec. 9.

2 THE STRONG CALL-BY-NEED STRATEGY

We first recall some general notions of λ-calculus and rewriting and extend the syntax of the λ-calculus with explicit substitutions. We then provide an explanation and a formal definition of the *strong call-by-need strategy*.

The λ-Calculus. Given a countable infinite set $\mathcal{V}$ of symbols $x, y, z, \ldots$, the **set of λ-terms** (Λ_β) and the **set of λ-contexts** is given by the following grammars:

$$\begin{array}{rcl} t, u, s & ::= & x\ (x \in \mathcal{V}) \mid tu \mid \lambda x.t \\ \mathtt{C} & ::= & \Box \mid \mathtt{C}u \mid t\mathtt{C} \mid \lambda x.\mathtt{C} \end{array}$$

Recall that we write $\mathtt{id}$ as shorthand for the term $\lambda z.z$. As usual, application is left-associative, so that $(\ldots((t_1 t_2) t_3)\ldots t_n)$ abbreviates $t_1 t_2 t_3 \ldots t_n$. **Free** and **bound variables** are defined as usual. The **β-reduction relation** $\to_\beta$ is the closure by all contexts $\mathtt{C}$ of the rewrite rule $(\lambda x.t)\, u \mapsto_\beta t\{x \backslash u\}$, where $\{\_\backslash\_\}$ denotes the standard capture free meta-substitution operator.

We use the following notions of rewriting over an appropriate set of terms. A term t is in $\mathcal{R}$-**normal form** ($\mathcal{R}$-nf) if there is no u such that $t \to_{\mathcal{R}} u$. We write $\mathtt{n}_{\mathcal{R}}$ for any term in $\mathcal{R}$-normal form. A term t is **weakly $\mathcal{R}$-normalising**, if there exists u in $\mathcal{R}$-normal form s.t. $t \twoheadrightarrow_{\mathcal{R}} u$. In what follows $\mathtt{NF}(\to_{\mathcal{R}})$ denotes the set of $\mathcal{R}$-normal forms and $\mathtt{WN}(\to_{\mathcal{R}})$ the set of weakly $\mathcal{R}$-normalising terms. We write $\twoheadrightarrow_{\mathcal{R}}$ for the reflexive and transitive closure of any reduction relation $\to_{\mathcal{R}}$.

An Extended Syntax with Explicit Substitutions. The syntax in which the strong call-by-need strategy is to be defined extends the syntax of λ-calculus with explicit substitutions, which will allow us to represent sharing and memoisation. This and other required categories of syntactical objects are presented below:

$$\begin{array}{rrcl}
\textbf{(Terms)} & t, u, s & ::= & x\ (x \in \mathcal{V}) \mid tu \mid \lambda x.t \mid t[x\backslash u] \\
\textbf{(Values)} & v & ::= & \lambda x.t \\
\textbf{(Answers)} & a & ::= & \mathtt{L}[v] \\
\textbf{(Full Contexts)} & \mathtt{C} & ::= & \Box \mid \mathtt{C}t \mid t\mathtt{C} \mid \lambda x.\mathtt{C} \mid \mathtt{C}[x\backslash t] \mid t[x\backslash \mathtt{C}] \\
\textbf{(Substitution Contexts)} & \mathtt{L} & ::= & \Box \mid \mathtt{L}[x\backslash t]
\end{array}$$

The set of all terms is denoted by Λ. A term $t[x\backslash u]$ is called a **closure**, and $[x\backslash u]$ is called an **explicit substitution (ES)**. Terms without explicit substitutions are called **pure terms**. Terms of the form $t[x\backslash u]$ are written as $\mathtt{let}\ x\ \mathtt{be}\ u\ \mathtt{in}\ t$ in [Ariola, Felleisen, Maraist, Odersky, and Wadler 1995]; our use of explicit substitutions is closer to this construction than to traditional calculi with explicit substitutions [Kesner 2009]. Indeed, our strategy builds on the weak call-by-need calculus introduced by Accattoli, Barenbaum, and Mazza [2014], which in turn is based on the *Linear Substitution Calculus*, a variation of the λ-calculus with Explicit Substitutions inspired from Milner [2007] and further developed by Accattoli and Kesner [2010]. Thus, in this setting, explicit substitutions have a behaviour similar to the let-constructs commonly used in defining call-by-need.

The notions of **free** and **bound variables** of terms are defined as usual, in particular,

$$\begin{array}{rclcrcl}
\mathrm{fv}(t[x\backslash u]) & := & (\mathrm{fv}(t) \setminus \{x\}) \cup \mathrm{fv}(u) & \qquad & \mathrm{fv}(\lambda x.t) & := & \mathrm{fv}(t) \setminus \{x\} \\
\mathrm{bv}(t[x\backslash u]) & := & \mathrm{bv}(t) \cup \{x\} \cup \mathrm{bv}(u) & & \mathrm{bv}(\lambda x.t) & := & \mathrm{bv}(t) \cup \{x\}
\end{array}$$

For example $\mathrm{fv}((xy)[x\backslash z]) = \{y, z\}$ and $\mathrm{bv}(\lambda x.x[z\backslash y]) = \{x, z\}$. This notion is extended to contexts as expected, in particular $\mathrm{fv}(\Box) := \emptyset$.

We write $\mathtt{C}[t]$ (resp. $\mathtt{L}[t]$ or simply $t\mathtt{L}$) for the term obtained by replacing the hole of $\mathtt{C}$ (resp. $\mathtt{L}$) by the term t. We write $\mathtt{C}[\![t]\!]$ when the free variables of t are not captured by the context, *i.e.* there are no abstractions or explicit substitutions in the context that bind the free variables of t. For example, given $\mathtt{C} = (\Box x)[x\backslash z]$, we have $(yx)[x\backslash z] = \mathtt{C}[y] = \mathtt{C}[\![y]\!]$, but $(xx)[x\backslash z] = \mathtt{C}[x]$ cannot be written as $\mathtt{C}[\![x]\!]$.

We work with the standard notion of α-conversion on terms *i.e.* renaming of bound variables for abstractions and explicit substitutions. **Substitutions** are (finite) functions from variables to terms denoted by $\{x_1\backslash u_1, ..., x_n\backslash u_n\}$ ($n \geq 0$). The **domain** of a substitution σ is given $\mathtt{dom}(\sigma) := \{x \mid \sigma(x) \neq x\}$. We extend this notion to substitution contexts by $\mathtt{dom}(\Box) := \emptyset$ and $\mathtt{dom}(\mathtt{L}[x\backslash u]) := \mathtt{dom}(\mathtt{L}) \cup \{x\}$. Application of the substitution σ to the term t, written $t\sigma$, is defined on α-equivalence classes, as usual, so that capture of free variables cannot hold. Thus *eg.* $(\lambda x.z)\{z\backslash x\} = \lambda x'.x$. Terms (with explicit substitutions) and λ-terms may be related by the **unfolding** function $\_^\diamond$:

$$x^\diamond := x \qquad (\lambda x.t)^\diamond := \lambda x.t^\diamond \qquad (tu)^\diamond := t^\diamond u^\diamond \qquad t[x\backslash u]^\diamond := t^\diamond\{x\backslash u^\diamond\}$$

An example is $(\lambda y.x[x\backslash yz[z\backslash \mathsf{id}]])^\diamond = \lambda y.y\ \mathsf{id}$. Note that $\mathrm{fv}(t^\diamond) \subseteq \mathrm{fv}(t)$, *eg.* $\mathrm{fv}(x[y\backslash z]^\diamond) = \mathrm{fv}(x) = \{x\} \subset \{x, z\} = \mathrm{fv}(x[y\backslash z])$.

Garbage Collection. Once all the occurrences of the variable x bound by an explicit substitution $[x\backslash t]$ have been substituted, the explicit substitution itself plays no role anymore and thus could be erased without altering the meaning of the term. While our strategy will not perform any such erasure, reasoning on these erasable substitutions turns out to be useful. The explicit substitution $[x\backslash u]$ in $t[x\backslash u]$ is called a **garbage substitution** if $x \notin \mathsf{fv}(t)$. We provide a **garbage collection function** $\downarrow_{\mathsf{gc}}(t)$ defined as follows:

$$\begin{array}{rcl} \downarrow_{\mathsf{gc}}(x) & := & x \\ \downarrow_{\mathsf{gc}}(\lambda x.t) & := & \lambda x.\downarrow_{\mathsf{gc}}(t) \\ \downarrow_{\mathsf{gc}}(tu) & := & \downarrow_{\mathsf{gc}}(t)\downarrow_{\mathsf{gc}}(u) \\ \downarrow_{\mathsf{gc}}(t[x\backslash u]) & := & \begin{cases} \downarrow_{\mathsf{gc}}(t)[x\backslash\downarrow_{\mathsf{gc}}(u)] & \text{if } x \in \mathsf{fv}(\downarrow_{\mathsf{gc}}(t)) \\ \downarrow_{\mathsf{gc}}(t) & \text{otherwise} \end{cases} \end{array}$$

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 $\downarrow_{\mathsf{gc}}(.)$, even if an occurrence of the variable z temporarily appears in the subterm $x[y\backslash z]$. Formally, the explicit substitution $[x\backslash u]$ in $t[x\backslash u]$ is called an ultimately-garbage substitution if it is a garbage substitution or if every free occurrence of x in t is itself inside an ultimately-garbage substitution. The $\downarrow_{\mathsf{gc}}(.)$ function then erases all ultimately-garbage substitutions.

Non-garbage variables of a term t, denoted $\mathsf{ngv}(t)$, are the free variables of t that are not erased by garbage collection, *i.e.*

$$\mathsf{ngv}(t) \quad = \quad \mathsf{fv}(\downarrow_{\mathsf{gc}}(t))$$

Equivalently, non-garbage variables can be characterized by the following inductive equations:

$$\begin{array}{rcl} \mathsf{ngv}(x) & := & \{x\} \\ \mathsf{ngv}(\lambda x.t) & := & \mathsf{ngv}(t) \setminus \{x\} \\ \mathsf{ngv}(tu) & := & \mathsf{ngv}(t) \cup \mathsf{ngv}(u) \\ \mathsf{ngv}(t[x\backslash u]) & := & (\mathsf{ngv}(t) \setminus x) \cup \begin{cases} \mathsf{ngv}(u) & \text{if } x \in \mathsf{ngv}(t) \\ \emptyset & \text{otherwise} \end{cases} \end{array}$$

Conversely, a free variable is a **garbage variable** if it is not non-garbage.

Structures and Normal Terms. The *normal terms*, the terms that are intended to be the target of the strong call-by-need strategy, depend on two key notions of *structures* and *frozen variables*. The latter play a central role in the definition of our system and hence we outline the three principles that have guided their conception. Formal definitions shall duly follow.

Principle 1: frozen variables define the shape of the structures. Consider a term such as xt. Since the variable x will never be substituted, it can be considered a constant, and we coin it *frozen*. Frozen variables provide terms with a certain rigidity regarding their form: reduction, if possible at all, will not change the overall form of these terms. Indeed, reduction in xt must necessarily take place in t and hence the variable x persists in the reduct. This motivates our calling a term such as xt, with t in normal form, a *structure*. Normal forms will include structures.

Principle 1 leaves the following question open: Is $y[y\backslash xt]$ a structure? The answer depends crucially on whether xt should be substituted for y.

Principle 2: structures should not be duplicated. Weak call-by-need only substitutes values, abstractions being the only possible values. The fact that in such calculi values coincide with weak-head normal forms raises the question of whether structures, weak normal forms in our strong reduction context, should be substituted too. In contrast to abstractions, structures cannot contribute in any way to creating new redexes. They represent an incomplete computation whose evaluation is

Fig. 2 Normal Terms and Structures

φ-Structures

$$\frac{x \in \varphi}{x \in \mathsf{S}_\varphi}\ \text{N-VAR}$$

$$\frac{t \in \mathsf{S}_\varphi \quad u \in \mathsf{N}_\varphi}{tu \in \mathsf{S}_\varphi}\ \text{N-APP}$$

φ-Normal Terms

$$\frac{t \in \mathsf{S}_\varphi}{t \in \mathsf{N}_\varphi}\ \text{N-INCL}$$

$$\frac{t \in \mathsf{N}_{\varphi \cup \{x\}}}{\lambda x.t \in \mathsf{N}_\varphi}\ \text{N-LAM}$$

Common Rules

$$\frac{t \in \mathbb{X}^{\varphi \cup \{x\}} \quad u \in \mathsf{S}_\varphi \quad x \in \mathsf{ngv}(t)}{t[x\backslash u] \in \mathbb{X}^\varphi}\ \text{N-SUB}^\bullet$$

$$\frac{t \in \mathbb{X}^\varphi \quad x \notin \mathsf{ngv}(t)}{t[x\backslash u] \in \mathbb{X}^\varphi}\ \text{N-SUB}^\circ$$

In the last two rules (labeled as "common rules"), the symbol $\mathbb{X}$ represents either S or N.

blocked by a head variable. Since we do not want to duplicate incomplete computations, we do not substitute structures: structures will not be considered as values. As a consequence, $y[y\backslash xt]$ is a structure.

To complete the picture on structures some questions remain: is the occurrence of the subterm $y[y\backslash xt]$ in $y[y\backslash xt][x\backslash \mathsf{id}]$ a structure? Similarly, is the subterm xy in $(xy)[x\backslash \mathsf{id}]$ a structure? The answer depends on the context in which the subterm occurs.

Principle 3: structures are context dependent. The notion of structure depends on the context in which it is considered. Indeed, xy is a structure under the context $\Box[x\backslash zz]$ where x itself is bound to a structure and thus is frozen, but it is not a structure under the context $\Box[x\backslash \mathsf{id}]$ where x is bound to a value and thus is not frozen. In the forthcoming definitions, we abstract this surrounding context by only remembering which variables are frozen or not: xy is a structure under the set of frozen variables $\{x, y\}$ (which corresponds, for instance, to the context $\Box[x\backslash zz]$), while it is not under the set of frozen variables $\{y\}$ (which corresponds, for instance, to the context $\Box[x\backslash \mathsf{id}]$).

Following our discussion above, *structures* and *normal terms* are to be understood under a set of frozen variables φ. The sets of **normal terms under the set of frozen variables** φ, called also **φ-normal terms** (N_φ) and of **structures under the set of frozen variables** φ, called also **φ-structures** (S_φ) are defined mutually recursively by means of the rules in Fig. 2.

Note that every φ-structure is *headed* by a frozen variable in the set φ: indeed both xyy and $z[z\backslash xy]$ are $\{x\}$-structures headed by x. Note also that the set φ of frozen variables in the premises and the conclusion of a rule might differ depending on how the terms are combined. For example, a variable bound by an abstractions $\lambda x.t$ is considered as frozen in its body t. Note also that normal terms may contain explicit substitutions, which play two very different roles:

- either they contain a term whose particular shape should be shared, as in $\lambda y.(xx)[x\backslash yt]$.
- or they are ultimately-garbage substitutions, as in $\lambda y.y[x\backslash t]$.

In the first case the variable x bound by the explicit substitution is frozen, while in the second one it is not. As a further example, $\lambda z.x[x\backslash yz][y\backslash z]$ is a normal form since x is frozen under $\lambda z.\Box[x\backslash yz][y\backslash z]$ given that yz is a structure in the context $\lambda z.\Box[y\backslash z]$ in which y and z are frozen. However, $\lambda z.x[x\backslash yz][y\backslash \lambda w.t]$ is not a normal form since x is not frozen under $\lambda z.\Box[x\backslash yz][y\backslash \lambda w.t]$ given that yz is not a structure in $\lambda z.\Box[y\backslash \lambda w.t]$, in which only z is frozen.

Evaluation Contexts. The strong call-by-need strategy is based on two reduction rules that are applied at specific subterms in a term, as specified by *evaluation contexts*. The two principles of reduction are:

(1) perform function application as soon as β-redexes are found; and

(2) evaluate and substitute the terms to which variables are bound *on demand*.

Let us see how these principles translate for the reduction of an application tu. Evaluation should not focus on the argument u by default, since we do not know yet whether this argument is actually needed. Hence the first step is to reduce t (*cf.* rule E-APP$_L$) 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 $(\lambda x.t')\mathsf{L}$, then the evaluation focus does not go deeper in this subterm, and we create an explicit substitution $t'[x\backslash u]\mathsf{L}$.
- 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 (examples of such would-be structures are x (id id), $x[x\backslash y$ (id id)](id id) or $x\ \Omega$ with Ω the usual non-terminating term). In this case both t and u have to be independently evaluated to full normal form. According to our strategy, the evaluation focus will stay in t until it becomes a proper structure (*i.e.* a normal form), and then continue in u (*cf.* rule E-APP$_R$).

Hence, the choice of reducing in t or u 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 φ of frozen variables, and the rule E-APP$_R$ specifies that $t\,\mathsf{C}$ is an evaluation context under the set of frozen variables φ whenever C is an evaluation context and t is a structure under the same set of frozen variables φ.

Now consider a term of the form $t[x\backslash u]$. Following the second reduction principle, reduction of the term u 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 u (*cf.* rule E-SUB$_R$) 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 $(zy)[z\backslash x$ id]. In contrast, x is not the focused variable in $(yx)[y\backslash$ id], since y is not frozen under $\Box[y\backslash$ id] so, in this particular case, evaluation should proceed to perform the substitution of id 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. We implement this condition by distinguishing a particular subset of the evaluation contexts, containing all the evaluation context 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 β-redex (*cf.* rule E-APP$_L$), and in a substitution to avoid reduction in a value that should be substituted (*cf.* rule E-SUB$_R$). 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 when exploring t (*cf.* rule E-LAM).

Following these principles, we define the sets of **evaluation contexts under the set of frozen variables** φ, also called **φ-evaluation contexts** (E_φ) and of **inert evaluation contexts under the set of frozen variables** φ, also called **inert φ-evaluation contexts** ($\mathsf{E}^{@}_\varphi$) mutually recursively in Fig. 3. The notion of frozen variable is (as before) encoded in the parameter φ, that is extended with a new frozen variable when this is required.

Observe that $\mathsf{E}^{@}_\varphi$ is a subset of E_φ by the rule E-INCL. Intuitively, inert evaluation contexts are those evaluation contexts that can be plugged into arbitrary contexts without creating a redex. In particular, the inclusion $\mathsf{E}^{@}_\varphi \subset \mathsf{E}_\varphi$ is proper. For instance, $\lambda x.\Box$ is an evaluation context, *i.e.*

Fig. 3 Evaluation Contexts

Inert φ-evaluation contexts

$$\frac{}{\Box \in \mathsf{E}^{@}_{\varphi}}\ \text{E-EMPTY}$$

$$\frac{\mathsf{C} \in \mathsf{E}^{@}_{\varphi}}{\mathsf{C}\,t \in \mathsf{E}^{@}_{\varphi}}\ \text{E-APP}_L$$

$$\frac{t \in \mathsf{S}_{\varphi} \quad \mathsf{C} \in \mathsf{E}_{\varphi}}{t\,\mathsf{C} \in \mathsf{E}^{@}_{\varphi}}\ \text{E-APP}_R$$

φ-Evaluation contexts

$$\frac{\mathsf{C} \in \mathsf{E}^{@}_{\varphi}}{\mathsf{C} \in \mathsf{E}_{\varphi}}\ \text{E-INCL}$$

$$\frac{\mathsf{C} \in \mathsf{E}_{\varphi \cup \{x\}}}{\lambda x.\mathsf{C} \in \mathsf{E}_{\varphi}}\ \text{E-LAM}$$

Common rules

$$\frac{\mathsf{C} \in \mathbb{X}^{\varphi} \quad t \notin \mathsf{S}_{\varphi} \quad x \notin \varphi}{\mathsf{C}[x\backslash t] \in \mathbb{X}^{\varphi}}\ \text{E-SUB}^{\notin S}_{L}$$

$$\frac{\mathsf{C} \in \mathbb{X}^{\varphi \cup \{x\}} \quad t \in \mathsf{S}_{\varphi}}{\mathsf{C}[x\backslash t] \in \mathbb{X}^{\varphi}}\ \text{E-SUB}^{\in S}_{L}$$

$$\frac{\mathsf{C}_1 \in \mathbb{X}^{\varphi} \quad \mathsf{C}_2 \in \mathsf{E}^{@}_{\varphi}}{\mathsf{C}_1[\![x]\!][x\backslash \mathsf{C}_2] \in \mathbb{X}^{\varphi}}\ \text{E-SUB}_{R}$$

In the last three rules (labeled as "common rules"), the symbol $\mathbb{X}$ represents either E or $\mathsf{E}^{@}$.

$\lambda x.\Box \in \mathsf{E}_{\emptyset}$, but it is not inert, *i.e.* $\lambda x.\Box \notin \mathsf{E}^{@}_{\emptyset}$, since plugging it into a context whose hole is applied to an argument, such as $\Box[y\backslash t]u$ creates a redex $(\lambda x.\Box)[y\backslash t]u$. Note that plugging $\lambda x.\Box$ into a context might also create an $\mathtt{lsv}$-redex, for instance by plugging it in $y[y\backslash\Box[z\backslash u]]$ we obtain the $\mathtt{lsv}$-redex $y[y\backslash(\lambda x.\Box)[z\backslash u]]$. For a different example, $(\lambda x.y)[y\backslash\Box]$ is an evaluation context but it is not inert. In fact, it is not difficult to show that inert evaluation contexts $\mathsf{E}^{@}_{\varphi}$ are exactly those evaluation contexts $\mathsf{C} \in \mathsf{E}_{\varphi}$ such that $\mathsf{C}[\![x]\!]$ is not an answer[1].

Reduction. The strong call-by-need strategy is defined by two reduction rules, whose application is guided by the evaluation contexts:

(1) if an evaluation context focuses on an application of the form $(\lambda x.t)\mathsf{L}\,u$, then two things happen: the substitution context L is pushed outside the application and a closure $t[x\backslash u]\mathsf{L}$ is created (*cf.* rule $\mathtt{dB}$)[2];
(2) if an evaluation context focuses on an occurrence of a variable x, and this variable is bound by a substitution to an answer $v\mathsf{L}$, then two things happen: the occurrence of x is substituted by the value v, and the substitution $[x\backslash v\mathsf{L}]$ is turned into the sequence of explicit substitutions $[x\backslash v]\mathsf{L}$ (*cf.* rule $\mathtt{lsv}$)[3].

From these reduction principles we deduce two reduction rules $\overset{\varphi}{\rightsquigarrow}_{\mathtt{dB}}$ and $\overset{\varphi}{\rightsquigarrow}_{\mathtt{lsv}}$ defined in Fig. 4. A term t_1 is said to reduce in the **strong call-by-need strategy** to t_2 under a set of frozen variables φ, written $t_1 \overset{\varphi}{\rightsquigarrow} t_2$, if either $t_1 \overset{\varphi}{\rightsquigarrow}_{\mathtt{dB}} t_2$ or $t_1 \overset{\varphi}{\rightsquigarrow}_{\mathtt{lsv}} t_2$.

Fig. 4 Reduction Rules of the Strong Call-by-Need Strategy

$$\mathsf{C}[(\lambda x.t)\mathsf{L}\,u] \ \overset{\varphi}{\rightsquigarrow}_{\mathtt{dB}}\ \mathsf{C}[t[x\backslash u]\mathsf{L}] \qquad \text{if } \mathsf{C} \in \mathsf{E}_{\varphi}$$

$$\mathsf{C}_1[\mathsf{C}_2[\![x]\!][x\backslash v\mathsf{L}]] \ \overset{\varphi}{\rightsquigarrow}_{\mathtt{lsv}}\ \mathsf{C}_1[\mathsf{C}_2[v][x\backslash v]\mathsf{L}] \qquad \text{if } \mathsf{C}_1[\mathsf{C}_2[\Box][x\backslash v\mathsf{L}]] \in \mathsf{E}_{\varphi}$$

[1] In $\mathsf{C}[\![x]\!]$ the variable x does not play any significant role—it is just a placeholder to regard the context C as a term.
[2] This corresponds in [Ariola, Felleisen, Maraist, Odersky, and Wadler 1995] to multiple applications of the rule $\mathtt{C}$ followed by a single application of the rule $\mathtt{I}$.
[3] This corresponds in [Ariola, Felleisen, Maraist, Odersky, and Wadler 1995] to multiple applications of the rule $\mathtt{A}$ followed by a single application of the rule $\mathtt{V}$.

We write $\overset{\varphi}{\rightsquigarrow\!\!\!\!\rightarrow}$ to denote the reflexive-transitive closure of $\overset{\varphi}{\rightsquigarrow}$. Expressions of the form $(\lambda x.t)\mathsf{L}\ u$ and $\mathsf{C}[[x]][x\backslash v\mathsf{L}]$ (with C an evaluation context) are called **redexes** ($r, r_1, \ldots$). In particular, the second one is called an $\mathtt{lsv}$-redex on the variable x. We write $t_1 \rightsquigarrow t_2$ if $\varphi = \emptyset$.

Note the requirement $\mathsf{C}_1[\mathsf{C}_2[\Box][x\backslash v\mathsf{L}]] \in \mathsf{E}_\varphi$ in the definition of the $\mathtt{lsv}$-redex; it avoids $\mathtt{lsv}$-reducing $(\lambda x.y)[y\backslash \mathtt{id}]$ in $(\lambda x.y)[y\backslash \mathtt{id}]t$ so that the outermost dB-reduction step takes precedence instead. The condition also avoids to $\mathtt{lsv}$-reduce $x[x\backslash(\lambda y.yz)[z\backslash \mathtt{id}]]$ on the variable z, so that the $\mathtt{lsv}$-reduction step on the variable x takes precedence over it.

Two examples of $\overset{\varphi}{\rightsquigarrow}$-reduction sequences follow. In the example on the left, we consider a term $x\ (\mathtt{id}\ \mathtt{id})$ headed by a frozen variable: reduction proceed in the argument $\mathtt{id}\ \mathtt{id}$ until its normal form $\mathtt{id}[z\backslash \mathtt{id}]$ is attained, and thus we obtain a structure (*i.e.* a normal term). In the example on the right, we consider a variable x in a substitution context, and all but the last reduction steps are here to compute the answer to which x is bound.

$$
\begin{array}{ll}
x(\mathtt{id}\ \mathtt{id}) & \\
\overset{\{x\}}{\rightsquigarrow} & x(z[z\backslash \mathtt{id}]) \\
\overset{\{x\}}{\rightsquigarrow} & x(\mathtt{id}[z\backslash \mathtt{id}])
\end{array}
\qquad\qquad
\begin{array}{ll}
x[x\backslash y\ \mathtt{id}][y\backslash \mathtt{id}] & \\
\rightsquigarrow & x[x\backslash \mathtt{id}\ \mathtt{id}][y\backslash \mathtt{id}] \\
\rightsquigarrow & x[x\backslash z[z\backslash \mathtt{id}]][y\backslash \mathtt{id}] \\
\rightsquigarrow & x[x\backslash \mathtt{id}[z\backslash \mathtt{id}]][y\backslash \mathtt{id}] \\
\rightsquigarrow & \mathtt{id}[x\backslash \mathtt{id}][z\backslash \mathtt{id}][y\backslash \mathtt{id}]
\end{array}
$$

The following result entails that the relation $\rightsquigarrow$ is indeed a *deterministic strategy*.

Lemma 2.1 (Determinism). *If $\mathsf{C}[r]$ 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 $\mathsf{C}_1[r_1]$ and $\mathsf{C}_2[r_2]$, where $\mathsf{C}_1, \mathsf{C}_2 \in \mathsf{E}_\varphi$ are evaluation contexts and r_1, r_2 are anchors. Then $\mathsf{C}_1 = \mathsf{C}_2$ and $r_1 = r_2$.*

Furthermore, the inductive set N_φ turns out to characterize the notion of $\overset{\varphi}{\rightsquigarrow}$-normal form so from now on we write n_φ for any term in N_φ.

Lemma 2.2 (Normal forms). $\mathsf{N}_\varphi = \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$.

Note that, in contrast to Maraist, Odersky, and Wadler [1998], variables are not considered to be values. Doing so would render our strategy non-deterministic. Eg. $x[x\backslash y][y\backslash \mathtt{id}]$ would reduce (with $\varphi = \emptyset$) to both $x[x\backslash \mathtt{id}][y\backslash \mathtt{id}]$ and also $y[x\backslash y][y\backslash \mathtt{id}]$.

Discussion. It is natural to imagine other possible call-by-need strategies reducing terms to strong normal forms. The rationale behind our design choice is twofold: obtaining a strategy that is not only complete with respect to β-reduction (see Sec. 6), but also conservative over the weak call-by-need strategy from the literature (see Sec. 7). Naive extensions of the notion of evaluation context in [Ariola et al. 1995] may yield strong call-by-need strategies that do not meet both of these constraints.

First of all, any reduction inside an abstraction that is applied or that is going to be applied in the future breaks conservativity. For instance, one could choose to immediately normalize $\mathtt{id}\ x$ in the term $(\lambda x.\mathtt{id}\ x)\ \mathtt{id}$, or in the term $(y\ \mathtt{id})[y\backslash \lambda x.\mathtt{id}\ x]$, but this would not coincide with the weak call-by-need strategy from the literature.

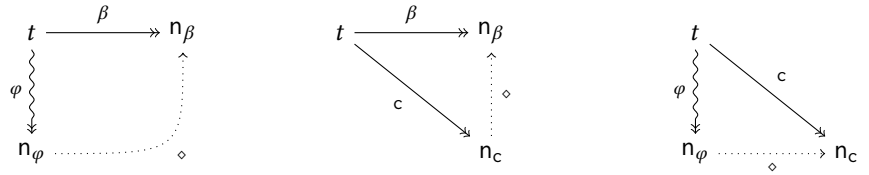
Still, in order to improve the efficiency of the strong strategy, one could want to relax this constraint and perform some steps that are not consistent with the weak call-by-need strategy. For instance, immediately normalizing $\mathtt{id}\ x$ in the term $(y\ y)[y\backslash \lambda x.\mathtt{id}\ x]$ would avoid doing it twice

after $\lambda x.\mathsf{id}\ x$ has been substituted for each occurrence of y. This could however very easily break completeness. Informally, performing any reduction step that is not actually necessary in finding the normal form is a threat for completeness. Moreover, the fact that a λ-abstraction is applied does not imply that its body needs to be fully normalized: there is no point in normalizing $\mathsf{id}\ x$ in the term $(\lambda z.z\ (\mathsf{id}\ x))(\lambda y.\mathsf{id})$ and more importantly, if Ω is a diverging term, we absolutely do not want to try to normalize it in the term $(\lambda z.z\ \Omega)(\lambda y.\mathsf{id})$. Thus a strategy reducing inside abstractions before they are applied should be clever enough to perform it only when the reduced redexes are needed, or at least when they do not compromise termination. Our strategy can be understood as one of the simplest strong call-by-need strategies, since it completely avoids these kinds of analyses, exactly as the weak call-by-need strategy does by selecting only obviously necessary redexes.

Finally, it is also possible to define other natural strong call-by-need strategies that turn out to be equivalent to ours, as for example, evaluating the arguments of an applied term from right to left. Among these equivalent possibilities, our choice is an arbitrary way of getting determinism.

3 THE STRONG CALL-BY-NEED CALCULUS

Fig. 5 Decomposition of the Completeness Proof



(a) Completeness of the strategy

(b) Completeness of the calculus

(c) Relation between the calculus and the strategy

The goal of Secs. 3 to 6 is to prove completeness of the strong call-by-need strategy introduced in Sec. 2: if a pure lambda term t β-reduces to a β-normal form n_β, then *essentially* the same normal form can be attained starting from t but using the strong call-by-need strategy to $\overset{\varphi}{\rightsquigarrow}$*-normal form.* Since $\overset{\varphi}{\rightsquigarrow}$-reduction is over terms with explicit substitutions, the $\overset{\varphi}{\rightsquigarrow}$-normal form, say n_φ, won't be exactly n_β, but rather a term such that, after unfolding of substitutions, results in n_β (*cf.* Fig. 5(a)). More precisely, $\mathsf{n}_\beta = \mathsf{n}_\varphi^\diamond$, where $\_^\diamond$ is the unfolding functioned defined in Sec. 2. As mentioned in the introduction, a direct proof of this result is non-trivial because of the difficulties in establishing a correlation between β-steps and $\overset{\varphi}{\rightsquigarrow}$-steps.

Decomposition of the Completeness Proof. The completeness result is slightly simpler to tackle if β-reduction is first correlated with a different reduction notion, that of the *strong call-by-need calculus* (written $\to_{\mathsf{c}}$). Steps of $\to_{\mathsf{c}}$ can take place under *any* context and are thus non-deterministic. This simplifies the task because $\to_{\mathsf{c}}$ operates over terms with explicit substitutions and has the same axioms as $\overset{\varphi}{\rightsquigarrow}$, even though $\overset{\varphi}{\rightsquigarrow}$ may only take place under evaluation contexts, rather than arbitrary contexts.

To tackle this correlation, we make use of a type-theoretic argument which was recently introduced in Kesner [2016] in the setting of weak reduction; we adapt it here to the strong case. Given the key role it plays in this paper, this technique is discussed in a dedicated section (*cf.* Sec. 4). For now we mention its main result: every λ-term t which is weakly β-normalising is also weakly normalising in the strong call-by-need calculus c (see Fig. 5(b)).

Even though we can come up with a way of relating β-steps and $\to_{\mathsf{c}}$-steps, *i.e.* steps in the strong call-by-need *calculus*, we are still left with the non-trivial issue of the difference in behavior between $\to_{\mathsf{c}}$ and $\overset{\varphi}{\rightsquigarrow}$, *i.e.* the strong call-by-need *strategy*. There are situations where the calculus $\to_{\mathsf{c}}$ will reduce subterms appearing in contexts that are not (strong call-by-need) evaluation contexts, such as $(xx)[x\backslash \mathsf{id}] \to_{\mathsf{c}} (x\ \mathsf{id})[x\backslash \mathsf{id}]$ and $y[x\backslash \mathsf{id}\ \mathsf{id}] \to_{\mathsf{c}} y[x\backslash z[z\backslash \mathsf{id}]]$. Appropriate postponement results introduced in Sec. 5 will show that these steps can always be permuted with subsequent $\overset{\varphi}{\rightsquigarrow}$-steps and ultimately boil down to reduction inside garbage substitutions, which are removed by the unfolding function $\_^{\diamond}$ (*cf.* Fig. 5(c)).

The Strong Call-by-Need Calculus. As mentioned before, the strong call-by-need strategy $\overset{\varphi}{\rightsquigarrow}$ is in fact part of a larger picture, the strong call-by-need calculus $\to_{\mathsf{c}}$: while $\overset{\varphi}{\rightsquigarrow}$ only represents one possible, albeit deterministic, way of computing normal forms, the calculus $\to_{\mathsf{c}}$ is a liberal reduction system where different reduction sequences are able to obtain normal forms. We next present the non-deterministic theory which our strategy relates to. It is obtained from the equational theory of (weak) call-by-need, but reduction is now closed by arbitrary contexts, and we additionally have a rule that discards useless explicit substitutions. Sec. 5 addresses completeness of $\overset{\varphi}{\rightsquigarrow}$ with respect to $\to_{\mathsf{c}}$.

The **strong call-by-need calculus**, or **c-calculus** for short, is given by the set of terms Λ, and the reduction relation $\to_{\mathsf{c}} := \to_{\mathsf{dB}} \cup \to_{\mathtt{lsv}} \cup \to_{\mathsf{gc}}$, where for each $\mathcal{R} \in \{\mathsf{dB}, \mathtt{lsv}, \mathsf{gc}\}$, $\to_{\mathcal{R}}$ is the closure by *full* contexts of the corresponding rewrite rules in Fig. 6.

Fig. 6 Reduction Rules of the Strong Call-by-Need Calculus

$$
\begin{array}{rcll}
(\lambda x.t)\mathsf{L}\ u & \mapsto_{\mathsf{dB}} & t[x\backslash u]\mathsf{L} & \\
\mathsf{C}[[x]][x\backslash v\mathsf{L}] & \mapsto_{\mathtt{lsv}} & \mathsf{C}[v][x\backslash v]\mathsf{L} & \\
t[x\backslash u] & \mapsto_{\mathsf{gc}} & t & \text{if } x \notin \mathsf{fv}(t)
\end{array}
$$

This new notion of reduction is a generalization of the strong call-by-need *strategy* $\overset{\varphi}{\rightsquigarrow}$ where $\{\mathsf{dB}, \mathtt{lsv}\}$-steps are allowed in *any* context and, moreover, also gc-steps are allowed.

Examples of reduction in the c-calculus are given in Fig. 7. In reduction sequence (a), no $\to_{\mathsf{c}}$-

Fig. 7 Examples of c-Reduction Sequences

(a)		(b)			(c)	
$x[y\backslash \mathsf{id}\ \mathsf{id}]$		$(xx)[x\backslash \mathsf{id}\ \mathsf{id}]$			$((\lambda z.y)x)[x\backslash \mathsf{id}\ \mathsf{id}]$	
$\to_{\mathsf{dB}}$	$x[y\backslash w[w\backslash \mathsf{id}]]$	$\to_{\mathsf{dB}}$	$(xx)[x\backslash w[w\backslash \mathsf{id}]]$	(1)	$\to_{\mathsf{dB}}$	$((\lambda z.y)x)[x\backslash w[w\backslash \mathsf{id}]]$
$\to_{\mathtt{lsv}}$	$x[y\backslash \mathsf{id}[w\backslash \mathsf{id}]]$	$\to_{\mathtt{lsv}}$	$(xx)[x\backslash \mathsf{id}[w\backslash \mathsf{id}]]$	(2)	$\to_{\mathtt{lsv}}$	$((\lambda z.y)x)[x\backslash \mathsf{id}[w\backslash \mathsf{id}]]$
$\to_{\mathsf{gc}}$	$x[y\backslash \mathsf{id}]$	$\to_{\mathtt{lsv}}$	$(x\ \mathsf{id})[x\backslash \mathsf{id}][w\backslash \mathsf{id}]$	(3)	$\to_{\mathtt{lsv}}$	$((\lambda z.y)\,\mathsf{id})[x\backslash \mathsf{id}][w\backslash \mathsf{id}]$
$\to_{\mathsf{gc}}$	x	$\to_{\mathtt{lsv}}$	$(\mathsf{id}\ \mathsf{id})[x\backslash \mathsf{id}][w\backslash \mathsf{id}]$	(4)	$\twoheadrightarrow_{\mathsf{gc}}$	$(\lambda z.y)\,\mathsf{id}$
		$\twoheadrightarrow_{\mathsf{gc}}$	$\mathsf{id}\ \mathsf{id}$	(5)		

reduction step belongs to strategy $\overset{\varphi}{\rightsquigarrow}$, the term $x[y\backslash \mathsf{id}\ \mathsf{id}]$ already being in $\overset{\varphi}{\rightsquigarrow}$-normal form. In reduction sequence (b), steps (1), (2) and (4) belong to the strategy $\overset{\varphi}{\rightsquigarrow}$, while (3) and (5) do not; and the final term $\mathsf{id}\ \mathsf{id}$ is not already in c-normal form. Finally, in the reduction sequence (c), no

$\to_{\mathsf{c}}$-reduction step belongs to the strategy $\overset{\varphi}{\rightsquigarrow}$, and the final term $(\lambda z.y)\mathsf{id}$ is not already in c-normal form.

We write $t \in \mathsf{NF}(\to_{\mathsf{c}})$ iff t is a c-normal form, *i.e.* if t is not reducible by $\to_{\mathsf{c}}$, and $t \in \mathsf{SNF}(\to_{\mathsf{c}})$ iff $t \in \mathsf{NF}(\to_{\mathsf{c}})$ and t is not an answer (*i.e.* is not a λ-abstraction inside a list context). We usually write n_{c} for a term in $\mathsf{NF}(\to_{\mathsf{c}})$. Terms in $\mathsf{NF}(\to_{\mathsf{c}})$ and $\mathsf{SNF}(\to_{\mathsf{c}})$ can be characterized, respectively, by the grammars in Fig. 8.

Fig. 8 Normal Forms for the Strong Call-by-Need Calculus $\to_{\mathsf{c}}$

Structures	Normal forms	Common Rule
$\dfrac{}{x \in \overline{\mathsf{S}}}$	$\dfrac{t \in \overline{\mathsf{S}}}{t \in \overline{\mathsf{N}}}$	$\dfrac{t \in \mathbb{X} \quad u \in \overline{\mathsf{S}} \quad x \in \mathsf{fv}(t)}{t[x\backslash u] \in \mathbb{X}}$
$\dfrac{t \in \overline{\mathsf{S}} \quad u \in \overline{\mathsf{N}}}{tu \in \overline{\mathsf{S}}}$	$\dfrac{t \in \overline{\mathsf{N}}}{\lambda x.t \in \overline{\mathsf{N}}}$	

In the last rule, the symbol $\mathbb{X}$ represents either $\overline{\mathsf{S}}$ or $\overline{\mathsf{N}}$.

Lemma 3.1. *Let $t \in \Lambda$. Then, $t \in \mathsf{SNF}(\to_{\mathsf{c}})$ iff $t \in \overline{\mathsf{S}}$, and $t \in \mathsf{NF}(\to_{\mathsf{c}})$ iff $t \in \overline{\mathsf{N}}$.*

Proof. The right-to-left implication is by simultaneous induction on $\overline{\mathsf{N}}$ and $\overline{\mathsf{S}}$ and the left-to-right implication is by induction on t. □

4 FROM β-REDUCTION TO THE c-CALCULUS

This section proves the diagram in Fig. 5(b), as discussed in Sec. 3: if t is a λ-term s.t. $t \twoheadrightarrow_{\beta} u \in \mathsf{NF}(\to_{\beta})$, then there is a c-reduction sequence from t to v', such that v' and u are related by the unfolding function. To do so, we use a type system which we dub $\mathcal{HW}$ (from $\mathcal{H}$ead $\mathcal{W}$eak normalisation). The system $\mathcal{HW}$ is known to be complete with respect to weakly β-normalising terms in the sense that if a λ-term t is weakly β-normalising, then it is typable in system $\mathcal{HW}$. In this section we are going to develop a soundness proof for the system $\mathcal{HW}$ by showing that t typable implies t is weakly normalising in the strong $\to_{\mathsf{c}}$-calculus (Thm. 4.5). Thus, $\to_{\mathsf{c}}$-reduction turns out to be complete with respect to β-reduction. Diagrammatically,

$$\beta\text{-normalisability} \Longrightarrow \mathcal{HW}\text{-typability} \Longrightarrow \mathsf{c}\text{-normalisability}$$

The completeness proof that we give in this section is a non-trivial extension to the *strong* call-by-need calculus of that developed byKesner [2016] for the *weak* call-by-need calculus. The main novelty in the proof of the strong case is that typability is considered under special conditions specified by means of positive and negative occurrences of types. Indeed, for a term t to be weakly β- and c-normalising, the type judgment of t must verify some particular additional syntactical restrictions that we describe in this section.

4.1 Non-idempotent Intersection Types

In contrast to simple types, intersection types Coppo and Dezani-Ciancaglini [1980] are powerful enough to characterize termination properties of lambda-calculi: a λ-term is head-normalising *if and only if* it is typable in a suitable intersection type system. That means, in particular, that β-head normalising terms like $\lambda x.xx$, which cannot be simply typed although it is in normal form, becomes now typable. This is done by introducing a new constructor of types ($\wedge$) together

with a corresponding set of typing rules. For instance, the previous term $\lambda x.xx$ is typable with $((\sigma \to \sigma) \wedge \sigma) \to \sigma$ so that the first (resp. second) occurrence of the variable x is typed with $\sigma \to \sigma$ (resp. σ). Typically, intersection types are commutative (*i.e.* $\sigma \wedge \tau = \tau \wedge \sigma$), associative (*i.e.* $(\sigma \wedge \tau) \wedge \rho = \sigma \wedge (\tau \wedge \rho)$) and *idempotent*, (*i.e.* $\sigma \wedge \sigma = \sigma$). An important variant is *non-idempotent* intersection types Gardner [1994], where $\sigma \wedge \sigma \neq \sigma$. These non-idempotent types give rise to refined resource aware semantics and provide a simple formal framework to reason about termination properties: in particular, correctness results can now be obtained by simple combinatorial arguments without any need of reducibility arguments Tait [1967].

From a formal point of view, commutative, associative and non-idempotent intersection can be denoted by *multisets*, which provide a very convenient notation to manipulate them. In this paper, finite multisets are denoted 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 $\sigma \wedge \sigma \wedge \tau$. We use $+$ for multiset union and $\sqsubset$ for multiset inclusion.

We next recall the intersection type system $\mathcal{HW}$ introduced in Kesner and Ventura [2014], which extends that in de Carvalho [2007]; Gardner [1994] from λ-terms to Λ-terms. Given a countable infinite set $\mathcal{B}$ of *base types* $\alpha, \beta, \gamma, \ldots$ we consider the following grammar:

(Types)	τ, σ, ρ	::=	$\alpha\ (\alpha \in \mathcal{B}) \mid \mathcal{M} \supset \tau$	
(Multiset types)	$\mathcal{M}$	::=	$\{\tau_i\}_{i \in I}$	I a finite set

The type $\{\ \}$ plays the rôle of the universal ω type in Coppo, Dezani-Ciancaglini, and Venneri [1981]. The types are *strict* Coppo and Dezani-Ciancaglini [1980]; van Bakel [1992], *i.e.* the right-hand sides of functional types are never multisets.

Type assignments, written Γ, Δ, are functions from variables to multiset types, assigning the empty multiset to all but a finite set of the variables. The domain of Γ is $\mathsf{dom}(\Gamma) := \{x \mid \Gamma(x) \neq \{\ \}\}$. The **union of type assignments**, written $\Gamma + \Delta$, is a type assignment defined by $(\Gamma + \Delta)(x) := \Gamma(x) + \Delta(x)$, where the symbol $+$ denotes multiset union; thus, $\mathsf{dom}(\Gamma + \Delta) = \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Delta)$. For example $(x : \{\sigma\}, y : \{\tau\}) + (x : \{\sigma\}, z : \{\sigma\}) = x : \{\sigma, \sigma\}, y : \{\tau\}, z : \{\sigma\}$. When $\mathsf{dom}(\Gamma)$ and $\mathsf{dom}(\Gamma)$ are disjoint we use $\Gamma; \Delta$ instead of $\Gamma + \Delta$. We write $+_{i \in I} \Delta_i$ for a finite union of type assignments (so that $I = \emptyset$ gives an empty function), and $\Gamma \setminus\!\setminus x$ for the assignement $(\Gamma \setminus\!\setminus x)(x) = \{\ \}$ and $(\Gamma \setminus\!\setminus x)(y) = \Gamma(y)$ if $y \neq x$. The **inclusion between type assignments** is defined by $\Gamma \sqsubseteq \Delta$ iff for every variable x we have $\Gamma(x) \sqsubseteq \Delta(x)$, *eg.* $x : \{\sigma\} \sqsubseteq x : \{\sigma, \sigma\}, y : \rho$.

Type judgments have the form $\Gamma \vdash t : \tau$, where Γ is a type assignment, t is a term and τ is a type. The **$\mathcal{HW}$-type system** for Λ-terms is given by the typing rules in Fig. 9. Notice that the axiom

Fig. 9 The Non-Idempotent Intersection Type System $\mathcal{HW}$

$$\frac{}{x{:}\{\tau\} \vdash x : \tau}\ (\mathsf{ax}) \qquad \frac{\Gamma \vdash t : \{\sigma_i\}_{i \in I} \supset \tau \qquad (\Delta_i \vdash u : \sigma_i)_{i \in I}}{\Gamma +_{i \in I} \Delta_i \vdash tu : \tau}\ (\supset\ \mathsf{e})$$

$$\frac{\Gamma \vdash t : \tau}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : \Gamma(x) \supset \tau}\ (\supset\ \mathsf{i}) \qquad \frac{x{:}\{\sigma_i\}_{i \in I}; \Gamma \vdash t : \tau \qquad (\Delta_i \vdash u : \sigma_i)_{i \in I}}{\Gamma +_{i \in I} \Delta_i \vdash t[x \backslash u] : \tau}\ (\mathsf{cut})$$

typing rule is *relevant* (there is no weakening) and the rules for application and substitution are *multiplicative* (the type assignment of the conclusion is obtained from the union of the premises type assignments); both characteristics being related to the *resource aware* semantics of the underlying calculus. As a consequence of this resource aware semantics, one can think of the type assignment

Γ in a judgment $\Gamma \vdash t : \tau$ as follows: given a variable x, each element in the multiset $\Gamma(x)$ concerns one potential use of this variable in the computation of t. This informal description helps in understanding the rules ($\supset$ e) and (cut), in which several typing judgments are required in the premises for the term u: each of these typing judgments concerns one of the potential uses of u in the computation of the whole term. A particular case of the rules ($\supset$ e) and (cut) is when $I = \emptyset$, *i.e.* there is no potential use of u: the subterm u occurring in the *typed* term tu (resp. $t[x\backslash u]$) *does not need to be typed.*

By restricting the $\mathcal{HW}$-system to λ-terms, so that it only contains the rules (ax), ($\supset$ i), and ($\supset$ e), we obtain the system presented in de Carvalho [2007]; Gardner [1994], which we call here **λ-type system**.

A **(type) derivation** is a tree obtained by applying the (inductive) rules of the type system. We write $\Phi \rhd \Gamma \vdash t : \tau$ if there is a derivation Φ typing t, *i.e.* ending in the type judgment $\Gamma \vdash t : \tau$ and we write $\Phi \rhd_\lambda \Gamma \vdash t : \tau$ when the derivation particularly holds in the λ-type system. A derivation Φ' is an **immediate subderivation** of Φ if Φ' ends in a type judgment which is a premiss of the last rule applied in Φ. A term t is **typable** iff there is a derivation typing t. The **size** of a type derivation Φ is a natural number $\mathsf{sz}(\Phi)$ denoting the number of nodes of the tree Φ. Remark that the typing rules of our system are syntax oriented, so that generation lemmas are not needed to distinguish particular syntactical forms of derivations.

An example of a derivation follows, where $\tau = \{\sigma\} \supset \sigma$, and Ω denotes the non-terminating term $(\lambda z.zz)(\lambda z.zz)$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : \{\tau\} \vdash x : \{\sigma\} \supset \sigma} \quad \dfrac{}{x : \{\sigma\} \vdash x : \sigma}}{x : \{\tau, \sigma\} \vdash xx : \sigma}(\supset \text{e})}{x : \{\tau, \sigma\} \vdash \lambda y.xx : \{\,\} \supset \sigma}(\supset \text{i})}{x : \{\tau, \sigma\} \vdash (\lambda y.xx)\Omega : \sigma}(\supset \text{e}) \quad \dfrac{}{z : \{\tau\} \vdash z : \tau} \quad \dfrac{}{z : \{\sigma\} \vdash z : \sigma}}{z : \{\tau, \sigma\} \vdash ((\lambda y.xx)\Omega)[x\backslash z] : \sigma}(\text{cut})$$

Given $\Phi \rhd \Gamma \vdash t : \sigma$, not every free variable in t necessarily appears in the domain of Γ, *eg.* $x : \{\sigma\} \vdash (\lambda y.x)z : \sigma$. However, $\mathcal{HW}$ does enjoy the following property, that can be easily shown by induction on derivations.

Lemma 4.1 (Relevance). *If* $\Phi \rhd \Gamma \vdash t : \sigma$ *then* $\mathsf{dom}(\Gamma) \subseteq \mathsf{fv}(t)$.

It is 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 we have a type derivation ending with $x : \{\{\,\} \supset \alpha\} \vdash x(\Delta\Delta) : \alpha$. In order to characterize weak β-normalisation by means of typability we need to restrict the types and the type assignments to those that do not have positive occurrences of the constant $\{\,\}$. Formally, **positive** (resp. **negative**) types of a type (resp. multiset type, assignment and pair of assignment and type) is the smallest set of types satisfying the following conditions, where T denotes a strict type or a multiset type:

$$\frac{}{\sigma \in \mathcal{P}(\sigma)} \quad \frac{}{\mathcal{M} \in \mathcal{P}(\mathcal{M})} \quad \frac{\mathsf{T} \in \mathcal{P}(\sigma_i) \;\&\; I \neq \emptyset}{\mathsf{T} \in \mathcal{P}(\{\sigma_i\}_{i\in I})} \quad \frac{\mathsf{T} \in \mathcal{N}(\mathcal{M})}{\mathsf{T} \in \mathcal{P}(\mathcal{M} \supset \tau)} \quad \frac{\mathsf{T} \in \mathcal{P}(\tau)}{\mathsf{T} \in \mathcal{P}(\mathcal{M} \supset \tau)}$$

$$\frac{\mathsf{T} \in \mathcal{N}(\sigma_i)\; I \neq \emptyset}{\mathsf{T} \in \mathcal{N}(\{\sigma_i\}_{i\in I})} \quad \frac{\mathsf{T} \in \mathcal{P}(\mathcal{M})}{\mathsf{T} \in \mathcal{N}(\mathcal{M} \supset \tau)} \quad \frac{\mathsf{T} \in \mathcal{N}(\tau)}{\mathsf{T} \in \mathcal{N}(\mathcal{M} \supset \tau)} \quad \frac{\mathsf{T} \in \mathcal{P}(\Gamma)}{\mathsf{T} \in \mathcal{P}(\Gamma \vdash \tau)} \quad \frac{\mathsf{T} \in \mathcal{P}(\tau)}{\mathsf{T} \in \mathcal{P}(\Gamma \vdash \tau)}$$

$$\frac{y \in \mathsf{dom}(\Gamma) \;\&\; \mathsf{T} \in \mathcal{N}(\Gamma(y))}{\mathsf{T} \in \mathcal{P}(\Gamma)} \quad \frac{y \in \mathsf{dom}(\Gamma) \;\&\; \mathsf{T} \in \mathcal{P}(\Gamma(y))}{\mathsf{T} \in \mathcal{N}(\Gamma)} \quad \frac{\mathsf{T} \in \mathcal{N}(\Gamma)}{\mathsf{T} \in \mathcal{N}(\Gamma \vdash \tau)} \quad \frac{\mathsf{T} \in \mathcal{N}(\tau)}{\mathsf{T} \in \mathcal{N}(\Gamma \vdash \tau)}$$

As an example, $\{\ \} \in \mathcal{P}(\{\ \})$, so that $\{\ \} \in \mathcal{N}(\{\ \} \supset \sigma)$, $\{\ \} \in \mathcal{P}(x : \{\{\ \} \supset \sigma\})$ and $\{\ \} \in \mathcal{P}(x : \{\{\ \} \supset \sigma\} \vdash \sigma)$.

Characterizations of weakly-normalising terms in the λ-calculus can be stated by means of the type system $\mathcal{HW}$ restricted to non-positive assignments and types.

Theorem 4.2. *Let t be a λ-term. Then, $t \in \mathtt{WN}(\to_\beta)$ iff $\Phi \rhd_\lambda \Gamma \vdash t : \tau$ and $\{\ \} \notin \mathcal{P}(\Gamma \vdash \tau)$.*

Proof. A straightforward adaptation of Krivine [1993] to the non-idempotent case. See Bucciarelli, Kesner, and Ventura [2017] for details. □

4.2 Typability implies Normalisation

In order to show that typable terms are weakly c-normalising we first need to establish a subject reduction property. For that, we need to introduce the notion of *positions* of terms that are *typed occurrences* (or T-occurrences), which helps to understand which are the redex occurrences actually constrained by the type system. Positions of terms can be seen as paths leading from the top level to some particular subterm of a term. Formally, the set of **positions** of a term t, written $\mathsf{pos}(t)$, is the set of finite words over the alphabet $\{0, 1\}$, inductively defined as follows:

$$\frac{}{\epsilon \in \mathsf{pos}(t)} \qquad \frac{p \in \mathsf{pos}(t)}{0p \in \mathsf{pos}(\lambda x.t)} \qquad \frac{p \in \mathsf{pos}(t)}{0p \in \mathsf{pos}(tu)} \qquad \frac{p \in \mathsf{pos}(t)}{1p \in \mathsf{pos}(ut)} \qquad \frac{p \in \mathsf{pos}(t)}{0p \in \mathsf{pos}(t[x\backslash u])} \qquad \frac{p \in \mathsf{pos}(t)}{1p \in \mathsf{pos}(u[x\backslash t])}$$

where ϵ denotes the empty word.

The set of positions of a context $\mathtt{C}$ is defined similarly. Thus for example, given $t = x[z\backslash z'](\lambda y.y)$ and $\mathtt{C} = (\lambda x.\Box)yz$, we have $\mathsf{pos}(t) = \{\epsilon, 0, 00, 01, 1, 10\}$ and $\mathsf{pos}(\mathtt{C}) = \{\epsilon, 0, 00, 000, 01, 1\}$. The subterm of t (resp $\mathtt{C}$) at position p is written $t|_p$ (resp. $\mathtt{C}|_p$) and defined as expected. Following the previous example, we have $t|_1 = \lambda y.y$ and $\mathtt{C}|_{000} = \Box$.

Let us consider a derivation $\Phi \rhd \Gamma \vdash t : \tau$. A position $p \in \mathsf{pos}(t)$ is a **T-occurrence** of t in Φ if either $p = \epsilon$, or $p = ip'$ $(i = 0, 1)$ and $p' \in \mathsf{pos}(t|_i)$ 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. Thus 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'.

$$\Phi' \rhd \dfrac{\dfrac{}{x{:}\{\{\tau,\tau\} \supset \tau\} \vdash x : \{\tau,\tau\} \supset \tau} \quad \dfrac{y{:}\{\{\ \} \supset \tau\} \vdash y : \{\ \} \supset \tau}{y{:}\{\{\ \} \supset \tau\} \vdash yz : \tau} \quad \dfrac{y{:}\{\{\ \} \supset \tau\} \vdash y : \{\ \} \supset \tau}{y{:}\{\{\ \} \supset \tau\} \vdash yz : \tau}}{x{:}\{\{\tau,\tau\} \supset \tau\}, y{:}\{\{\ \} \supset \tau, \{\ \} \supset \tau\} \vdash x(yz) : \tau}$$

Notice that whenever a variable (say x) position has a T-occurrence of t in Φ, then $x \in \mathsf{fv}(t)$. Given $\Phi \rhd \Gamma \vdash t : \tau$, the **no-redex-occurrences** predicate $A(t, \Phi)$ holds if and only if t has no c-redex T-occurrences in Φ. The notion of T-occurrence plays a key role in the forthcoming Lemma 4.3.

Lemma 4.3 (Weighted Subject Reduction for c). *Let $\Phi \rhd \Gamma \vdash t : \tau$. If $t \to_{\mathtt{c}} t'$ reduces a c-redex T-occurrence of t in Φ, then there exists Φ' such that $\Phi' \rhd \Gamma \vdash t' : \tau$ and $\mathsf{sz}(\Phi) > \mathsf{sz}(\Phi')$.*

Proof. By induction on Φ using a Partial Substitution Lemma (see *eg.* [Kesner 2016]). The $\to_{\mathtt{gc}}$ case is the only one that eventually changes the type assignment Γ and the type τ. □

We now relate the notions of T-occurrence and c-normal form, before concluding with the main result of this section.

Lemma 4.4. *Let $\Phi \rhd \Gamma \vdash t : \tau$ s.t. $\{\ \} \notin \mathcal{P}(\Gamma \vdash \tau)$. Then $A(t, \Phi)$ implies $t \in \mathtt{NF}(\to_{\mathtt{c}})$.*

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 $\{\ \} \notin \mathcal{P}(\Gamma)$, and t is not an answer, then $t \in \overline{\mathsf{S}}$.
(2) If $\{\ \} \notin \mathcal{P}(\Gamma \vdash \tau)$, and t is an answer, then $t \in \overline{\mathsf{N}}$.

Moreover, $x \in \mathsf{fv}(t)$ implies x has some T-occurrence in Φ.

Now, suppose $\{\ \} \notin \mathcal{P}(\Gamma \vdash \tau)$. Thus in particular $\{\ \} \notin \mathcal{P}(\Gamma)$. If t is not an answer, then one easily shows that $t \in \overline{\mathsf{S}}$, which gives $t \in \overline{\mathsf{N}}$ since $\overline{\mathsf{S}} \subseteq \overline{\mathsf{N}}$; If t is an answer, then one easily shows $t \in \overline{\mathsf{N}}$. We conclude that $t \in \mathsf{NF}(\to_{\mathsf{c}})$ by Lem. 3.1. □

Theorem 4.5 (Typability implies c-normalisation). *Let $\Phi \rhd \Gamma \vdash t : \tau$ s.t. $\{\ \} \notin \mathcal{P}(\Gamma \vdash \tau)$. Then t is weakly normalising in the strong* c*-calculus.*

Proof. Let $\Phi \rhd \Gamma \vdash t : \tau$ s.t. $\{\ \} \notin \mathcal{P}(\Gamma \vdash \tau)$. By Lem. 4.3 and Lem. 4.4 we can construct a *finite* c-reduction sequence which only reduces c-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) $t_i \to_{\mathsf{c}} t_{i+1}$ reduces a c-redex T-occurrences of t_i in Φ_i, and (5) $A(t_n, \Phi_n)$ holds. This together with $\{\ \} \notin \mathcal{P}(\Gamma \vdash \tau)$ gives $t_n \in \mathsf{NF}(\to_{\mathsf{c}})$ by Lem. 4.4. We thus conclude $t \in \mathsf{WN}(\to_{\mathsf{c}})$. □

4.3 Completeness for the c-Calculus

This section gives the final result in Fig. 5(b), *i.e.* completeness of the strong call-by-need calculus with respect to the β-calculus. Before doing so, we present some basic properties of unfolding.

Lemma 4.6. *Let $t, u \in \Lambda$.*

(1) *If $t \to_{\mathsf{c}} u$, then $t^\circ \twoheadrightarrow_\beta u^\circ$.*
(2) *$t \in \mathsf{NF}(\to_{\mathsf{c}})$ implies $t^\circ \in \mathsf{NF}(\to_\beta)$, where $\mathsf{NF}(\to_\beta)$ stands for the set of β-normal forms.*

Indeed, to illustrate the first point we have $t = y[y\backslash(\lambda z.zz)(\mathsf{id}\ \mathsf{id})] \to_{\mathsf{dB}} y[y\backslash(zz)[z\backslash \mathsf{id}\ \mathsf{id}]] = u$ and $t^\circ = (\lambda z.zz)(\mathsf{id}\ \mathsf{id}) \to_\beta (\mathsf{id}\ \mathsf{id})\ (\mathsf{id}\ \mathsf{id}) = u^\circ$, and to illustrate the second one we have $t = x[y\backslash \mathsf{id}[w'\backslash \mathsf{id}]][z\backslash \mathsf{id}] \in \mathsf{NF}(\to_{\mathsf{c}})$ and $t^\circ = x \in \mathsf{NF}(\to_\beta)$.

We now conclude with the completeness result for the c-calculus (*cf.* Fig. 5(b)), which is the essential step in the final completeness result of the paper (*cf.* Thm 6.1).

Theorem 4.7 (Completeness for $\to_{\mathsf{c}}$). *Let t be a λ-term. If $t \in \mathsf{WN}(\to_\beta)$, then $t \in \mathsf{WN}(\to_{\mathsf{c}})$. More precisely, if $t \twoheadrightarrow_\beta \mathsf{n}_\beta$, where n_β is a β-normal form, then $t \twoheadrightarrow_{\mathsf{c}} \mathsf{n}_{\mathsf{c}}$, where n_{c} is a* c*-normal form. Moreover, $\mathsf{n}_{\mathsf{c}}^\circ = \mathsf{n}_\beta$.*

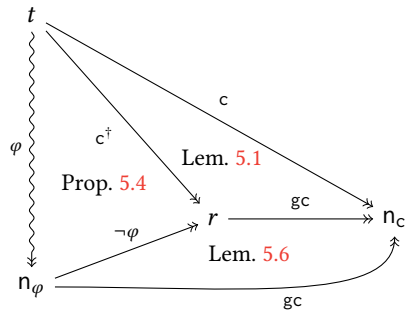
Proof. Let $t \twoheadrightarrow_\beta \mathsf{n}_\beta$, where n_β is in β-nf. Then $\Phi \rhd \Gamma \vdash t : \tau$ and $\{\ \} \notin \mathcal{P}(\Gamma \vdash \tau)$ by Thm. 4.2. But then t is weakly c-normalising by Thm. 4.5, so that $t \twoheadrightarrow_{\mathsf{c}} \mathsf{n}_{\mathsf{c}}$, where n_{c} is in c-nf. By Lem. 4.6(1) $t^\circ \twoheadrightarrow_\beta \mathsf{n}_{\mathsf{c}}^\circ$ and by Lem. 4.6(2) $\mathsf{n}_{\mathsf{c}}^\circ \in \mathsf{NF}(\to_\beta)$. Since $t^\circ = t \twoheadrightarrow_\beta \mathsf{n}_\beta$ and $t^\circ \twoheadrightarrow_\beta \mathsf{n}_{\mathsf{c}}^\circ$, then we conclude $\mathsf{n}_{\mathsf{c}}^\circ = \mathsf{n}_\beta$ because $\to_\beta$ is CR. □

With this completeness theorem, we complete the first part of our proof, corresponding to Fig. 5(b) on page 11.

5 FROM THE c-CALCULUS TO THE STRONG CALL-BY-NEED STRATEGY

Sec. 4.3 showed the relations stated in Fig. 5(b). We still, however, must show those in Fig. 5(c) by relating the $\to_{\mathsf{c}}$-normal form to the $\overset{\varphi}{\rightsquigarrow}$-normal form, that is, to the normal form of t produced by the $\overset{\varphi}{\rightsquigarrow}$ strategy and Fig. 5(c). For this, we show that $\to_{\mathsf{c}}$ reduction steps which are not $\overset{\varphi}{\rightsquigarrow}$ steps can always be postponed after $\overset{\varphi}{\rightsquigarrow}$ reduction steps, that this postponement process terminates, and that, ultimately, all remaining non-$\overset{\varphi}{\rightsquigarrow}$ steps are erasable by gc (and thus erased by unfolding $\_^\circ$). More precisely we proceed in three stages, as pictured in Fig. 10 and explained in the following paragraph.

Fig. 10 Decomposition of Fig. 5(c)



(1) As a preliminary step, we first get gc-steps out of the way: any $\to_{\mathsf{c}}$ reduction sequence can be factorized into a $\to_{\mathsf{c}}$ reduction sequence without gc-steps, which we call strict and write $\to_{\mathsf{c}^\dagger}$, followed by a sequence of gc-steps (*cf.* Lem. 5.1).
(2) Then we provide another more involved commutation result: $\to_{\mathsf{c}}$-reductions without gc-steps can be factored in two parts (*cf.* Prop. 5.4):
   (a) a sequence of *external* $\to_{\mathsf{c}}$-steps, which correspond to the strategy $\overset{\varphi}{\rightsquigarrow}$, followed by;
   (b) a sequence of *internal* $\to_{\mathsf{c}}$-steps, which are not related to the strategy.

   We write $\xrightarrow{\neg\varphi}$ instead of $\to_{\mathsf{c}}$ for such internal steps. Two examples of internal steps are $(xx)[x\backslash \mathsf{id}] \xrightarrow{\neg\varphi} (x\ \mathsf{id})[x\backslash \mathsf{id}]$ and $(\mathsf{id}\ x)[x\backslash \mathsf{id}\ \mathsf{id}] \xrightarrow{\neg\varphi} (\mathsf{id}\ x)[x\backslash z[z\backslash \mathsf{id}]]$, 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 $\overset{\varphi}{\rightsquigarrow}$-normal form is reached only take place inside garbage substitutions, that are removed by the unfolding function $\_^\circ$ (*cf.* Lem. 5.6).

5.1 Postponement of gc

This section states the common 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 $\to_{\mathsf{c}^\dagger}$ (resp. $\twoheadrightarrow_{\mathsf{c}^\dagger}$).

Lemma 5.1 (Postponement of gc). *If $t \twoheadrightarrow_{\mathsf{c}} u$, then there is r such that $t \twoheadrightarrow_{\mathsf{c}^\dagger} r \twoheadrightarrow_{\mathsf{gc}} u$. Diagramatically, see Fig. 10.*

In this lemma, observe that assuming $u \in \mathsf{NF}(\to_{\mathsf{c}})$ would not imply $r \in \mathsf{NF}(\to_{\mathsf{c}^\dagger})$. Indeed, if we take $t = x[y\backslash\Omega]$ and $u = x$, then $r = x[y\backslash\Omega]$, which is not even normalising for $\to_{\mathsf{c}^\dagger}$. The actual relation of postponement of gc with normal forms is stated in Lem. 5.6.

5.2 Factorization of Strict Reduction in the Strong Call-by-Need Calculus

We say that t_1 reduces in a **φ-internal step** to t_2, written $t_1 \xrightarrow{\neg\varphi} t_2$, if and only if there is a step in the strict strong call-by-need calculus that is not a step in the strong call-by-need strategy, *i.e.* $t_1\ (\to_{\mathsf{c}^\dagger} \setminus \overset{\varphi}{\rightsquigarrow})\ t_2$. We sometimes call φ-internal steps just **internal steps** if φ is clear from the context. Steps in the strategy $\overset{\varphi}{\rightsquigarrow}$ are called **φ-external steps** (or just **external steps**).

The following lemma states that an external step can be permuted 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). Recall that $\overset{\varphi}{\rightsquigarrow\!\!\!\!\rightarrow}$ denotes the reflexive-transitive closure of $\overset{\varphi}{\rightsquigarrow}$.

Lemma 5.2 (Postponement of internal steps). *Let* $\mathsf{fv}(t_0) \subseteq \varphi$. *If* $t_0 \xrightarrow{\neg\varphi} t_1 \overset{\varphi}{\rightsquigarrow} t_3$, *then there is a term* t_2 *such that* $t_0 \overset{\varphi}{\rightsquigarrow\!\!\!\!\rightarrow} t_2 \overset{\neg\varphi}{\rightarrow\!\!\!\!\rightarrow} t_3$, *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. The whole proof is tedious (due to the numerous cases in the inductive characterization of evaluation contexts) but requires some care in tracing back the evaluation context used in t_1, to t_0. We present here the most interesting cases.

- Starting from the term $(\lambda x.\mathsf{id}\ x)t$ we have the following reduction:

$$(\lambda x.\mathsf{id}\ x)t \xrightarrow{\neg\varphi} (\lambda x.z[z\backslash x])t \overset{\varphi}{\rightsquigarrow} z[z\backslash x][x\backslash t]$$

  where the first step is internal since it reduces inside an applied abstraction. This is a case where the internal step does not modify anything outside of the abstraction, hence it does not affect what defines the external step, which could have been performed first, yielding the reduction

$$(\lambda x.\mathsf{id}\ x)t \overset{\varphi}{\rightsquigarrow} (\mathsf{id}\ x)[x\backslash t] \overset{\varphi}{\rightsquigarrow} z[z\backslash x][x\backslash t]$$

  Remark that in this particular example, the once-internal step becomes an external step after permutation, since the redex $\mathsf{id}\ x$ arrives to head position.
- Starting from the term $x[x\backslash\lambda y.z][z\backslash\mathsf{id}]$ we have the following reduction:

$$x[x\backslash\lambda y.z][z\backslash\mathsf{id}] \xrightarrow{\neg\varphi} x[x\backslash\lambda y.\mathsf{id}][z\backslash\mathsf{id}] \overset{\varphi}{\rightsquigarrow} (\lambda y.\mathsf{id})[x\backslash\lambda y.\mathsf{id}][z\backslash\mathsf{id}]$$

  where the first step is internal since it substitutes a variable inside a value that has not been substituted yet. In such a case one immediately remarks that the internal step does not affect the context of the external step, and then that the external can be performed first:

$$x[x\backslash\lambda y.z][z\backslash\mathsf{id}] \overset{\varphi}{\rightsquigarrow} (\lambda y.z)[x\backslash\lambda y.z][z\backslash\mathsf{id}]$$

  What is new in this example is that the external step duplicates the subterm $\lambda y.z$, and thus duplicates the internal step: we need two more steps to close the diagram.

$$x[x\backslash\lambda y.z][z\backslash\mathsf{id}] \overset{\varphi}{\rightsquigarrow} (\lambda y.z)[x\backslash\lambda y.z][z\backslash\mathsf{id}] \overset{\varphi}{\rightsquigarrow} (\lambda y.\mathsf{id})[x\backslash\lambda y.z][z\backslash\mathsf{id}] \xrightarrow{\neg\varphi} (\lambda y.\mathsf{id})[x\backslash\lambda y.t][z\backslash t]$$

  Remark that in this particular example, the first copy of the internal step turns external, while the second copy stays internal.
- Starting from the term $(y(\mathsf{id}\ x))[y\backslash\mathsf{id}\ x]$ we have the following reduction:

$$(y(\mathsf{id}\ x))[y\backslash\mathsf{id}\ x] \xrightarrow{\neg\varphi} (y(z[z\backslash x]))[y\backslash\mathsf{id}\ x] \overset{\varphi}{\rightsquigarrow} (y(z[z\backslash x]))[y\backslash z[z\backslash x]]$$

  where the first step is internal since it reduces on the right of an application whose head variable y is not frozen. One easily remarks that these two steps can be permuted to form the sequence

$$(y(\mathsf{id}\ x))[y\backslash\mathsf{id}\ x] \overset{\varphi}{\rightsquigarrow} (y(\mathsf{id}\ x))[y\backslash z[z\backslash x]] \to_{\mathsf{c}} (y(z[z\backslash x]))[y\backslash z[z\backslash x]]$$

  However, in contrast to the previous example, the internal step affects the context of the external step. To conclude that the step $(y(\mathsf{id}\ x))[y\backslash\mathsf{id}\ x]\to_{\mathsf{c}}(y(\mathsf{id}\ x))[y\backslash z[z\backslash x]]$ is external, we need to ensure that the context $(y(\mathsf{id}\ x))[y\backslash\Box]$ is an evaluation context while by hypothesis we only know that the context $(y(z[z\backslash x]))[y\backslash\Box]$ obtained after the internal step is an evaluation

context. Here the two evaluation contexts are easily related, since their derivations are isomorphic (they are respectively built using the contexts $\Box(\mathsf{id}\ x)$ and $\Box(z[z\backslash x])$, which are straightforward evaluation contexts). By noting that $(y\Box)[y\backslash z[z\backslash x]]$ is an evaluation context, we conclude:

$$(y(\mathsf{id}\ x))[y\backslash \mathsf{id}\ x] \overset{\varphi}{\rightsquigarrow} (y(\mathsf{id}\ x))[y\backslash z[z\backslash x]] \overset{\varphi}{\rightsquigarrow} (y(z[z\backslash x]))[y\backslash z[z\backslash x]]$$

The next example shows a variant where this relation between contexts is not as straightforward.

- Starting from the term $(yx)[x\backslash vL][y\backslash \mathsf{id}\ \mathsf{id}]$ we have the following reduction:

$$(yx)[x\backslash vL][y\backslash \mathsf{id}\ \mathsf{id}] \xrightarrow{\neg\varphi} (yv)[x\backslash v]L[y\backslash \mathsf{id}\ \mathsf{id}] \overset{\varphi}{\rightsquigarrow} (yv)[x\backslash v]L[y\backslash z[z\backslash \mathsf{id}]]$$

where the first step is internal since it substitutes a variable on the right of an application whose head variable y is not frozen. As in the previous example, one easily remarks the possibility of permutation:

$$(yx)[x\backslash vL][y\backslash \mathsf{id}\ \mathsf{id}] \to_{\mathsf{c}} (yx)[x\backslash vL][y\backslash z[z\backslash \mathsf{id}]] \to_{\mathsf{c}} (yv)[x\backslash v]L[y\backslash z[z\backslash \mathsf{id}]]$$

To conclude it is sufficient to remark that $(yx)[x\backslash vL][y\backslash\Box]$ is an evaluation context while $(y\Box)[x\backslash vL][y\backslash z[z\backslash \mathsf{id}]]$ is not, so that the reduction sequence is indeed of the form

$$(yx)[x\backslash vL][y\backslash \mathsf{id}\ \mathsf{id}] \overset{\varphi}{\rightsquigarrow} (yx)[x\backslash vL][y\backslash z[z\backslash \mathsf{id}]] \xrightarrow{\neg\varphi} (yv)[x\backslash v]L[y\backslash z[z\backslash \mathsf{id}]]$$

To conclude in the general case we need to ensure that the context of the first term ($(yx)[x\backslash vL][y\backslash\Box]$ in our example) is an evaluation context, using only the hypothesis that the context in the second term ($(yv)[x\backslash v]L[y\backslash\Box]$ in our example) is an evaluation context. While the two evaluation contexts are still related, the derivations are not isomorphic anymore. To solve this (frequent) case, we rely on the following lemma stating that the various kinds of objects that are relevant for the strong call-by-need strategy (evaluation contexts, normal forms, etc.) cannot spring into existence by the action of an internal step.

Lemma 5.3 (Backward stability by internal steps). *Let $t_0 \xrightarrow{\neg\varphi} t$ be a φ-internal step. Then:*

(1) *If t is an answer (resp. a dB-redex) then t_0 is an answer (resp. a dB-redex).*
(2) *If t is a φ-normal form (resp. φ-normal structure) then t_0 is a φ-normal form (resp. φ-normal structure).*
(3) *If $t = \mathsf{C}[\![x]\!]$ where C is a φ-evaluation context (resp. non-answer φ-evaluation context), then t_0 is of the form $\mathsf{C}_0[\![x]\!]$, where C_0 is a φ-evaluation context (resp. a non-answer φ-evaluation context).*

Note that in item (3) of this statement, the variable x does not take part in the result; it is a placeholder used to turn the context into a term. The proof of (1) in this auxiliary lemma is by induction on the context under which the internal step takes place. The proof of (2) (resp. (3)) is by induction on the derivation that t is a normal form (resp. that C is an evaluation context).

□

Proposition 5.4 (Factorization). *Let $\mathsf{fv}(t) \subseteq \varphi$. If $t \twoheadrightarrow_{\mathsf{c}^\dagger} r$ then, there is s such that $t \overset{\varphi}{\rightsquigarrow\!\!\!\!\rightarrow} s \overset{\neg\varphi}{\twoheadrightarrow} r$. Diagramatically, see Fig. 10.*

Proof. This is a consequence of Theorem 5.2 in Accattoli [2012], and our postponement Lem. 5.2. Actually, one shows that $(\xrightarrow{\neg\varphi}_{\mathsf{dB}}, \overset{\varphi}{\rightsquigarrow}_{\mathsf{dB}}, \xrightarrow{\neg\varphi}_{\mathsf{lsv}}, \overset{\varphi}{\rightsquigarrow}_{\mathsf{lsv}})$ forms a *square factorization system* according

to the terminology of Accattoli [2012], taking $\overset{\varphi}{\rightsquigarrow}_{\mathsf{dB}}$ (resp. $\overset{\varphi}{\rightsquigarrow}_{\mathtt{lsv}}$) to be the external dB (resp. lsv) reduction, and $\xrightarrow{\neg\varphi}_{\mathsf{dB}}$ (resp. $\xrightarrow{\neg\varphi}_{\mathtt{lsv}}$) to be the internal dB (resp. lsv) reduction. □

5.3 Erasure of Final Internal Steps

The previous two subsections ensure that any $\to_{\mathsf{c}}$ reduction sequence can be factored into a $\overset{\varphi}{\rightsquigarrow}$ reduction prefix followed by internal or gc steps. Here we further show that if the $\to_{\mathsf{c}}$ reduction sequence reaches a $\to_{\mathsf{c}}$-normal form, then all the internal steps factored out by Prop. 5.4 can be erased by gc steps.

Lemma 5.5 (Inclusion of Normal Forms). *Let φ, t be such that $\mathsf{fv}(t) \subseteq \varphi$. If $t \in \mathsf{NF}(\to_{\mathsf{c}})$, then $t \in \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$.*

Proof. This is immediate since $\overset{\varphi}{\rightsquigarrow} \subseteq \to_{\mathsf{c}}$. □

Lemma 5.6 (Normal Forms Modulo Internal and gc steps). *Let φ, t be such that $\mathsf{fv}(t) \subseteq \varphi$.*

(1) *If $t \to_{\mathsf{gc}} \mathsf{n}_\varphi$ with $\mathsf{n}_\varphi \in \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$ then $t \in \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$.*

(2) *If $t \xrightarrow{\neg\varphi} \mathsf{n}_\varphi$ with $\mathsf{n}_\varphi \in \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$ then $t \in \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$ and there is u such that $t \twoheadrightarrow_{\mathsf{gc}} u$ and $\mathsf{n}_\varphi \twoheadrightarrow_{\mathsf{gc}} u$.*

Diagramatically, see Fig. 10.

Proof. We show that the following conditions are equivalent for any term t such that $\mathsf{fv}(t) \subseteq \varphi$. They imply items (1) and (2) of this lemma: (i) t is a $\overset{\varphi}{\rightsquigarrow}$-normal form, (ii) $\downarrow_{\mathsf{gc}}(t)$ is a $\to_{\mathsf{c}}$-normal form, (iii) $t \equiv_{\neg\varphi} u$ for some $u \in \mathsf{NF}(\to_{\mathsf{c}})$, (iv) $t \equiv_{\neg\varphi} u$ for some $u \in \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$. Here $\equiv_{\neg\varphi}$ stands for the least equivalence relation containing $\to_{\mathsf{gc}} \cup \xrightarrow{\neg\varphi}$. □

As an example of this lemma, consider the sequence $x[y\backslash z[z\backslash \mathsf{id}]] \xrightarrow{\neg\varphi} x[y\backslash \mathsf{id}[z\backslash \mathsf{id}]] \to_{\mathsf{gc}} x$. All three terms are in $\mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$: 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 $x[y\backslash z[z\backslash \mathsf{id}]] \to_{\mathsf{gc}} x[y\backslash \mathsf{id}[z\backslash \mathsf{id}]]$, both terms reduce in one gc-step to the same term x.

5.4 Relation Between the c-Calculus and the Strong Call-by-Need Strategy

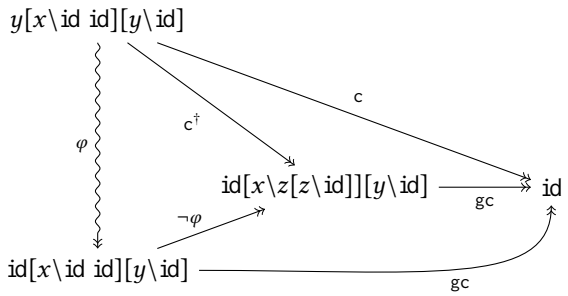
The results in this section can now be assembled to complete the argument outlined in Fig. 10.

Theorem 5.7 (Completeness of $\overset{\varphi}{\rightsquigarrow}$ with respect to $\to_{\mathsf{c}}$). *Let φ, t be such that $\mathsf{fv}(t) \subseteq \varphi$. If $t \twoheadrightarrow_{\mathsf{c}} \mathsf{n}_{\mathsf{c}}$, where $\mathsf{n}_{\mathsf{c}} \in \mathsf{NF}(\to_{\mathsf{c}})$, then there exists a term $\mathsf{n}_\varphi \in \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$ such that $t \overset{\varphi}{\rightsquigarrow\!\!\!\!\rightsquigarrow} \mathsf{n}_\varphi$ and $\mathsf{n}_\varphi \twoheadrightarrow_{\mathsf{gc}} \mathsf{n}_{\mathsf{c}}$.*

Proof. The structure of the proof is shown in Fig. 10, which details all the necessary steps to obtain the general Fig. 5(c) introduced in Sec. 3. More precisely, suppose $t \twoheadrightarrow_{\mathsf{c}} \mathsf{n}_{\mathsf{c}}$ with $\mathsf{n}_{\mathsf{c}} \in \mathsf{NF}(\to_{\mathsf{c}})$ (so that in particular $\mathsf{n}_{\mathsf{c}} \in \mathsf{NF}(\to_{\mathsf{gc}})$). By Lem. 5.1 there is a term r such that the reduction sequence $t \twoheadrightarrow_{\mathsf{c}} \mathsf{n}_{\mathsf{c}}$ can be decomposed as $t \twoheadrightarrow_{\mathsf{c}^\dagger} r \twoheadrightarrow_{\mathsf{gc}} \mathsf{n}_{\mathsf{c}}$, and by Prop. 5.4 there is a term s such that the reduction sequence $t \twoheadrightarrow_{\mathsf{c}^\dagger} r$ can in turn be decomposed as $t \overset{\varphi}{\rightsquigarrow\!\!\!\!\rightsquigarrow} s \xrightarrow{\neg\varphi}\!\!\!\!\twoheadrightarrow r$. Finally, since $\mathsf{fv}(\mathsf{n}_{\mathsf{c}}) \subseteq \varphi$, by Lem. 5.5 we have $\mathsf{n}_{\mathsf{c}} \in \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$, and Lem. 5.6 allows us to deduce that both r and s are also in $\mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$. Moreover, using convergence of $\to_{\mathsf{gc}}$ (which is straightforward), Lem. 5.6 further allows us to deduce that $s \twoheadrightarrow_{\mathsf{gc}} \mathsf{n}_{\mathsf{c}}$. □

Example in Fig. 11 illustrates the previous completenes result, where $t = y[x\backslash \mathsf{id}\ \mathsf{id}][y\backslash \mathsf{id}]$ and the $\to_{\mathsf{c}}$ normalization sequence is: $y[x\backslash \mathsf{id}\ \mathsf{id}][y\backslash \mathsf{id}] \xrightarrow{\neg\varphi} y[x\backslash z[z\backslash \mathsf{id}]][y\backslash \mathsf{id}] \to_{\mathsf{gc}} y[y\backslash \mathsf{id}] \overset{\varphi}{\rightsquigarrow} \mathsf{id}[y\backslash \mathsf{id}] \to_{\mathsf{gc}} \mathsf{id}$.

Fig. 11 Completeness of $\overset{\varphi}{\rightsquigarrow}$ with respect to $\to_{\mathsf{c}}$ - Example



6 COMPLETENESS OF THE STRONG CALL-BY-NEED STRATEGY

This final section on completeness assembles the main results of the previous two sections in order to obtain the completeness property for $\overset{\varphi}{\rightsquigarrow}$ (*cf.* Thm. 6.1). The main results from the previous section that we are interested in are:

(1) Thm. 4.7: If t has a β-normal form, then $t \in \mathsf{WN}(\to_{\mathsf{c}})$.
(2) Thm. 5.7: Any $\to_{\mathsf{c}}$ normalization sequence can be factorized as a $\overset{\varphi}{\rightsquigarrow}$ normalization sequence followed by gc-steps.

Theorem 6.1 (Completeness of $\overset{\varphi}{\rightsquigarrow}$). *Let φ, t be such that $\mathsf{fv}(t) \subseteq \varphi$. If $t \twoheadrightarrow_{\beta} \mathsf{n}_{\beta}$, where $\mathsf{n}_{\beta} \in \mathsf{NF}(\to_{\beta})$, then there exists a term $\mathsf{n}_{\varphi} \in \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$ such that $t \overset{\varphi}{\rightsquigarrow\!\!\!\!\twoheadrightarrow} \mathsf{n}_{\varphi}$ and $\mathsf{n}_{\varphi}^{\diamond} = \mathsf{n}_{\beta}$.*

Proof. The proof follows the decomposition given in Fig. 5.

More precisely, suppose $t \twoheadrightarrow_{\beta} \mathsf{n}_{\beta}$ with $\mathsf{n}_{\beta} \in \mathsf{NF}(\to_{\beta})$. By Thm. 4.7 there is a term $\mathsf{n}_{\mathsf{c}} \in \mathsf{NF}(\to_{\mathsf{c}})$ such that $t \twoheadrightarrow_{\mathsf{c}} \mathsf{n}_{\mathsf{c}}$ and $\mathsf{n}_{\mathsf{c}}^{\diamond} = \mathsf{n}_{\beta}$. Then by Thm. 5.7, for any φ such that $\mathsf{fv}(t) \subseteq \varphi$ there is a term $\mathsf{n}_{\varphi} \in \mathsf{NF}(\overset{\varphi}{\rightsquigarrow})$ such that $t \overset{\varphi}{\rightsquigarrow\!\!\!\!\twoheadrightarrow} \mathsf{n}_{\varphi}$ and $\mathsf{n}_{\varphi} \twoheadrightarrow_{\mathsf{gc}} \mathsf{n}_{\mathsf{c}}$. The latter allows us to deduce that $\mathsf{n}_{\varphi}^{\diamond} = \mathsf{n}_{\mathsf{c}}^{\diamond} = \mathsf{n}_{\beta}$. □

Example in Fig. 12 illustrates the previous completenes result and its decomposition, where $t = (\lambda y.\lambda w.\lambda x.w(xx))\Omega(z\ \mathsf{id})(\mathsf{id}\ \mathsf{id})$.

Fig. 12 Completeness of $\overset{\varphi}{\rightsquigarrow}$ - Example

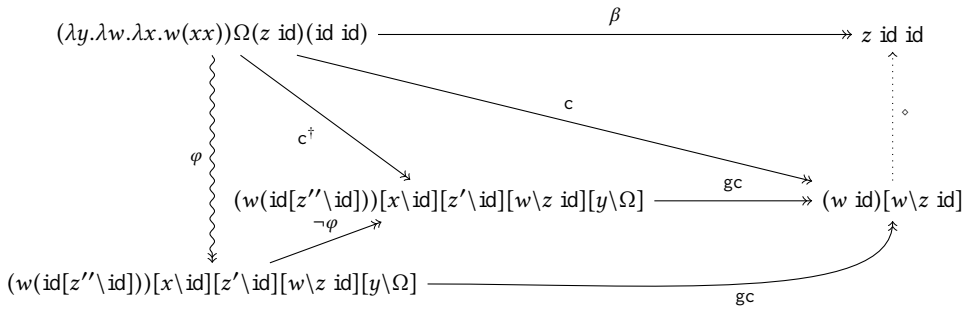


Fig. 13 Ariola et al's Standard Call-by-Need Reduction

$$\begin{array}{rcl} M & ::= & x \mid \lambda x.M \mid MM \mid \mathtt{let}\, x = M \,\mathtt{in}\, M \\ V & ::= & \lambda x.M \\ A & ::= & V \mid \mathtt{let}\, x = M \,\mathtt{in}\, A \\ \mathcal{E} & ::= & \Box \mid \mathcal{E}M \mid \mathtt{let}\, x = M \,\mathtt{in}\, \mathcal{E} \mid \mathtt{let}\, x = \mathcal{E} \,\mathtt{in}\, \mathcal{E}[x] \end{array}$$

$$\begin{array}{rcl} (\lambda x.M)\, N & \overset{\mathrm{R}}{\to}_{\mathtt{I}} & \mathtt{let}\, x = N \,\mathtt{in}\, M \\ \mathtt{let}\, x = V \,\mathtt{in}\, \mathcal{E}[x] & \overset{\mathrm{R}}{\to}_{\mathtt{V}} & \mathtt{let}\, x = V \,\mathtt{in}\, \mathcal{E}[V] \\ (\mathtt{let}\, x = M \,\mathtt{in}\, A)\, N & \overset{\mathrm{R}}{\to}_{\mathtt{C}} & \mathtt{let}\, x = M \,\mathtt{in}\, A\, N \\ \mathtt{let}\, x = (\mathtt{let}\, y = M \,\mathtt{in}\, A) \,\mathtt{in}\, \mathcal{E}[x] & \overset{\mathrm{R}}{\to}_{\mathtt{A}} & \mathtt{let}\, y = M \,\mathtt{in}\, \mathtt{let}\, x = A \,\mathtt{in}\, \mathcal{E}[x] \end{array}$$

Remark that, in constrast to what happened in the previous example of Fig. 11, where we considered an arbitrary $\to_{\mathtt{c}}$-normalizing sequence, the $\to_{\mathtt{c}}$-normalization sequence in this example is obtained from Thm. 4.7. This particular $\to_{\mathtt{c}}$-normalization sequence causes the $\overset{\neg\varphi}{\longrightarrow}$ reduction sequence to be empty. For example, even if the β-normalization sequence taken as a starting point happens to contain a subreduction $\Omega \to_\beta \Omega$, inspection of the proof of Thm. 4.7 shows that the resulting $\to_{\mathtt{c}}$-reduction sequence built by this theorem does not keep this (internal) step[4]. Then the reduction sequence obtained after gc-postponement (Lem 5.1) does not contain any internal step either, and the $\overset{\neg\varphi}{\longrightarrow}$ reduction sequence foreseen by the factorization result (Prop. 5.4) happens to be empty.

7 CONSERVATIVITY

This section shows that our proposed strong call-by-need strategy is conservative over weak call-by-need. For that, we relate our strategy to the original notion of weak call-by-need reduction in [Ariola and Felleisen 1997; Ariola, Felleisen, Maraist, Odersky, and Wadler 1995], as well as to the more recent formulation of weak call-by-need in [Accattoli, Barenbaum, and Mazza 2014]. Moreover, we take the opportunity to put in evidence the general scheme followed by any reduction sequence of [Ariola, Felleisen, Maraist, Odersky, and Wadler 1995] (*cf.* Lem. 7.2), and we also state a clear relation between these two mentioned notions of weak call-by-need (*cf.* Lem. 7.3)

Original Notion of Weak Call-by-Need. The system in [Ariola and Felleisen 1997; Ariola, Felleisen, Maraist, Odersky, and Wadler 1995] is depicted in Fig. 13. It will be convenient to replace "let" by our explicit substitutions so as to facilitate the comparison: rather than writing $\mathtt{let}\, x = M \,\mathtt{in}\, N$ we write $t[x\backslash u]$. This also entails that V becomes v and A becomes a. Moreover, we have to adapt weak evaluation contexts to our LSC setting: the set of **weak contexts** is defined as follows:

$$\mathsf{E} ::= \Box \mid \mathsf{E}\, t \mid \mathsf{E}[x\backslash t] \mid \mathsf{E}[[x]][x\backslash \mathsf{E}]$$

The rules at the bottom of Fig. 13, thus become:

$$\begin{array}{rcl} (\lambda x.t)\, u & \overset{\mathrm{R}}{\to}_{\mathtt{I}} & t[x\backslash u] \\ \mathsf{E}[[x]][x\backslash v] & \overset{\mathrm{R}}{\to}_{\mathtt{V}} & \mathsf{E}[v][x\backslash v] \\ a[x\backslash t]\, u & \overset{\mathrm{R}}{\to}_{\mathtt{C}} & (a\, u)[x\backslash t] \\ \mathsf{E}[[x]][x\backslash a[y\backslash t]] & \overset{\mathrm{R}}{\to}_{\mathtt{A}} & \mathsf{E}[[x]][x\backslash a][y\backslash t] \end{array}$$

[4]More generally, we conjecture that it does not contain any internal step at all, but this is beyond the scope of the paper.

This completes our presentation of the system of Fig. 13; henceforth we shall work with these rephrased rules when refering to Ariola et al's notion of weak call-by-need. Reduction in this rephrased system is defined as $\to_{\mathtt{needlet}} \stackrel{\text{def}}{=} \to_{\mathtt{I}} \cup \to_{\mathtt{V}} \cup \to_{\mathtt{C}} \cup \to_{\mathtt{A}}$, where $\to_{\mathtt{I}} \stackrel{\text{def}}{=} \mathsf{E}[\stackrel{\mathsf{R}}{\to_{\mathtt{I}}}]$, $\to_{\mathtt{V}} \stackrel{\text{def}}{=} \mathsf{E}[\stackrel{\mathsf{R}}{\to_{\mathtt{V}}}]$, $\to_{\mathtt{C}} \stackrel{\text{def}}{=} \mathsf{E}[\stackrel{\mathsf{R}}{\to_{\mathtt{C}}}]$, and $\to_{\mathtt{A}} \stackrel{\text{def}}{=} \mathsf{E}[\stackrel{\mathsf{R}}{\to_{\mathtt{A}}}]$. It turns out that $\to_{\mathtt{needlet}}$ is deterministic.

Proposition 7.1 (Determinism of $\to_{\mathtt{needlet}}$). *$t \to_{\mathtt{needlet}} u$ implies there exists a unique context $\mathcal{E}$ s.t. $t = \mathcal{E}[t']$, $u = \mathcal{E}[u']$ and $t' \stackrel{R}{\to} u'$, where $\stackrel{R}{\to} \stackrel{\text{def}}{=} \stackrel{R}{\to_{\mathtt{I}}} \cup \stackrel{R}{\to_{\mathtt{V}}} \cup \stackrel{R}{\to_{\mathtt{C}}} \cup \stackrel{R}{\to_{\mathtt{A}}}$.*

Using this property, one can observe that any reduction sequence in $\to_{\mathtt{needlet}}$ is organized into clusters of the form $\twoheadrightarrow_{\mathtt{C}}\to_{\mathtt{I}}$ or $\twoheadrightarrow_{\mathtt{A}}\to_{\mathtt{V}}$. More precisely,

Lemma 7.2 (Organisation of $\to_{\mathtt{needlet}}$ reduction sequences).

$$\twoheadrightarrow_{\mathtt{needlet}} \quad = \quad ((\twoheadrightarrow_{\mathtt{C}}\to_{\mathtt{I}}) \cup (\twoheadrightarrow_{\mathtt{A}}\to_{\mathtt{V}}))^*(\twoheadrightarrow_{\mathtt{C}} \cup \twoheadrightarrow_{\mathtt{A}})$$

Weak Call-by-Need Revisited. The **weak call-by-need strategy** $\rightsquigarrow_{\mathtt{wneed}}$ introduced in Accattoli, Barenbaum, and Mazza [2014] is defined as the union of the two rewrite rules in Fig. 14, closed by weak contexts (note that this relation is not indexed over φ given that reduction never takes place under abstractions):

Fig. 14 Reduction Rules of the Weak Call-by-Need Strategy

$$\begin{array}{llll} (\lambda x.t)\mathsf{L}\, u & \rightsquigarrow_{\mathtt{dB}} & t[x\backslash u]\mathsf{L} & (\mathtt{dB}) \\ \mathsf{E}[\![x]\!][x\backslash v\mathsf{L}] & \rightsquigarrow_{\mathtt{lsv}} & \mathsf{E}[\![v]\!][x\backslash v]\mathsf{L} & (\mathtt{lsv}) \end{array}$$

The set of terms defined by the grammar $\mathsf{WNF}_\varphi ::= v\mathsf{L} \mid \mathsf{E}[\![x]\!]$, with $x \in \varphi$, characterizes the set of normal forms with respect to the weak call-by-need strategy.

It is quite straightforward to deduce that $\rightsquigarrow_{\mathtt{wneed}}$ is included in $\to_{\mathtt{needlet}}$, in particular, a dB step (resp. lsv) step translates to a $\twoheadrightarrow_{\mathtt{C}}\to_{\mathtt{I}}$ cluster (resp. $\twoheadrightarrow_{\mathtt{A}}\to_{\mathtt{V}}$ cluster). These clusters in fact characterize $\rightsquigarrow_{\mathtt{wneed}}$.

Lemma 7.3 (Decomposition of $\rightsquigarrow_{\mathtt{wneed}}$).

$$\rightsquigarrow_{\mathtt{wneed}} \quad = \quad (\twoheadrightarrow_{\mathtt{C}}\to_{\mathtt{I}}) \cup (\twoheadrightarrow_{\mathtt{A}}\to_{\mathtt{V}})$$

Proof.

- The inclusion $\rightsquigarrow_{\mathtt{wneed}} \subseteq (\twoheadrightarrow_{\mathtt{C}}\to_{\mathtt{I}}) \cup (\twoheadrightarrow_{\mathtt{A}}\to_{\mathtt{V}})$ is proved by cases on the kind of redex contracted.
  (1) dB **redex.**
  $$\mathsf{E}[(\lambda x.t)\mathsf{L}\, u] \twoheadrightarrow_{\mathtt{C}} \mathsf{E}[((\lambda x.t)\, u)\mathsf{L}] \to_{\mathtt{I}} \mathsf{E}[t[x\backslash u]\mathsf{L}]$$
  (2) lsv **redex.**
  $$\mathsf{E}[\mathsf{E}'[\![x]\!][x\backslash v\mathsf{L}]] \twoheadrightarrow_{\mathtt{A}} \mathsf{E}[\mathsf{E}'[\![x]\!][x\backslash v]\mathsf{L}] \to_{\mathtt{V}} \mathsf{E}[\mathsf{E}'[\![v]\!][x\backslash v]\mathsf{L}]$$
- The inclusion $(\twoheadrightarrow_{\mathtt{C}}\to_{\mathtt{I}}) \cup (\twoheadrightarrow_{\mathtt{A}}\to_{\mathtt{V}}) \subseteq \rightsquigarrow_{\mathtt{wneed}}$ follows from the remarks stated below and determinism of $\to_{\mathtt{needlet}}$ (Prop. 7.1).

$$\begin{array}{lll} t \to_{\mathtt{I}} u & \text{implies} & t \to_{\mathtt{dB}} u \\ t \to_{\mathtt{V}} u & \text{implies} & t \to_{\mathtt{lsv}} u \\ t \to_{\mathtt{C}} u & \text{implies} & \exists u'.\ (u \twoheadrightarrow_{\mathtt{C}} \to_{\mathtt{I}} u') \wedge (t \to_{\mathtt{dB}} u') \\ t \to_{\mathtt{A}} u & \text{implies} & \exists u'.\ (u \twoheadrightarrow_{\mathtt{A}} \to_{\mathtt{V}} u') \wedge (t \to_{\mathtt{lsv}} u') \end{array}$$

□

Conservativity. We start by observing that our strong call-by-need strategy is conservative with respect to $\rightsquigarrow_{\mathsf{wneed}}$. For that, let $\overset{\varphi}{\rightsquigarrow}_{\mathsf{nonwneed}}$ denote $\overset{\varphi}{\rightsquigarrow} \setminus \rightsquigarrow_{\mathsf{wneed}}$. Then,

Theorem 7.4 (Conservativity). *If* $t_0 \overset{\varphi}{\rightsquigarrow} t_1 \overset{\varphi}{\rightsquigarrow} \dots t_{n-1} \overset{\varphi}{\rightsquigarrow} t_n$ *then there exists an* $i \in 1..n$ *such that*

(1) $t_0 \rightsquigarrow_{\mathsf{wneed}} t_1 \rightsquigarrow_{\mathsf{wneed}} \dots t_{n-1} \rightsquigarrow_{\mathsf{wneed}} t_i$;
(2) $t_i \overset{\varphi}{\rightsquigarrow}_{\mathsf{nonwneed}} t_{i+1} \overset{\varphi}{\rightsquigarrow}_{\mathsf{nonwneed}} \dots t_{n-1} \overset{\varphi}{\rightsquigarrow}_{\mathsf{nonwneed}} t_n$; *and*
(3) *If* $i < n$, *then* $t_j \in \mathsf{WNF}_\varphi$ *for all* $i \le j \le n$.

As a corollary of Thm. 7.4 and Lem. 7.3, we deduce that our strategy $\overset{\varphi}{\rightsquigarrow}$ has Ariola et al's notion of reduction as prefix.

Corollary 7.5. $t \overset{\varphi}{\rightsquigarrow\!\!\!\!\rightarrow} u$ *implies there exists* s *s.t.* $t((\rightarrow\!\!\!\!\rightarrow_{\mathsf{C}} \rightarrow_{\mathsf{I}}) \cup (\rightarrow\!\!\!\!\rightarrow_{\mathsf{A}} \rightarrow_{\mathsf{V}}))^* s$ *and* $s \overset{\varphi}{\rightsquigarrow\!\!\!\!\rightarrow}_{\mathsf{nonwneed}} u$. *Moreover, if* $u \in \mathsf{N}_\varphi$, *then* s *is a normal form for* $\rightarrow_{\mathsf{needlet}}$ *up to a finite number of* $\overset{R}{\rightarrow}_{\mathsf{C}} \cup \overset{R}{\rightarrow}_{\mathsf{A}}$ *steps.*

8 RELATED WORKS

Weak Call-by-Need. Lazy evaluation, introduced by Henderson and Morris [1976]; Wadsworth [1971], delays the evaluation of an expression until its value is needed, thus avoiding repeated evaluation of terms by introducing a notion of *sharing*. This is typically[5] achieved by using let-constructs or explicit substitutions which allows arguments to be shared. Numerous characterizations of weak call-by-need calculi have appeared in the literature. Worth highlighting, for the role they played in furthering this line of research, are that of Ariola and Felleisen [1997]; Launchbury [1993]; Maraist, Odersky, and Wadler [1998]. There is ample additional literature on call-by-need for weak reduction; for further references in that line of work the reader may consult Chang and Felleisen [2012]. Completeness of weak call-by-need was first proved in Ariola and Felleisen [1997] by means of syntactical methods. A more abstract completeness proof using intersection types appears in Kesner [2016]. In Danvy and Zerny [2013] an analysis of how let-based theories of call-by-need reduction relate to store based implementations is presented. It also discusses connections between multiple machine based implementations of call-by-need. Other references on prior work by Danvy, on establishing a functional correspondence between calculi and their abstract machine based implementations, are provided in Danvy and Zerny [2013].

Substitutions at a Distance. A recent reformulation of Ariola and Felleisen's weak call-by-need calculus appears in Accattoli, Barenbaum, and Mazza [2014], which in turn is based on the *Linear Substitution Calculus*, a variation of the λ-calculus with Explicit Substitutions inspired from Milner [2007] and further developed in Accattoli and Kesner [2010]. The two rules of the weak call-by-need calculus in Accattoli, Barenbaum, and Mazza [2014] correspond to multiplicative and exponential cut-elimination rules in linear logic proof nets, where the commutative rules of Ariola and Felleisen's calculus can be completely absorbed. The resulting formalism is then more convenient to study the complexity of reduction Accattoli and Lago [2016] as well as to implement weak call-by-need abstract machines, as discussed by Accattoli, Barenbaum, and Mazza [2014].

(Non-Idempotent) Intersection Types. Intersection types, originally introduced in Coppo and Dezani-Ciancaglini [1980], provide models of the λ-calculus; they are able to characterize the sets of head, weak and strongly normalizing λ-terms. Typically, intersection types are idempotent,

[5] A similar objective may be achieved by "expanding" let-expressions into combinations of applications and abstraction [Maraist et al. 1998, Sec. 6], however sharing is less explicitly reflected.

however, in the last years, growing interest has been devoted to non-idempotent intersection types, as described by Gardner [1994] and de Carvalho [2007], since they allow to reason about quantitative properties of terms, both from a syntactical and a semantical point of view. Different assignment systems with non-idempotent intersection types have been studied in the literature for different purposes. The relation between the size of a non-idempotent intersection typing derivation and the head/weak normalization execution time of lambda-terms by means of abstract machines was established in de Carvalho [2007]. Non-idempotence is used to reason about the longest reduction sequence of strongly normalizing terms in both the lambda-calculus, as in the work of Bernadet and Lengrand [2011]; De Benedetti and Ronchi Della Rocca [2013] and in different lambda-calculi with explicit substitutions, as in the work of Bernadet and Lengrand [2013]; Kesner and Ventura [2014]. Non-idempotent types are also used to design resource aware semantics of sequent calculus in Kesner and Ventura [2015], to linearize the lambda-calculus in Kfoury [2000], and to deal with type inference for intersection types in Kfoury and Wells [2004].

Strong Reduction. As already mentioned in the introduction, Grégoire and Leroy [2002] proposed an implementation of strong call-by-value, by iterating the standard call-by-value strategy on open terms (terms with variables). In Boespflug, Dénès, and Grégoire [2011], it is noted that the implementation of Grégoire and Leroy [2002] requires modifying the OCaml abstract machine so they propose a native OCaml implementation where the tags that distinguish functions from accumulators are coded directly in OCaml itself. Crégut [1990, 2007] defined abstract machines for reduction to strong normal form. Other abstract machines for strong reduction have been studied too: de Carvalho [2009]; Ehrhard and Regnier [2006]; García-Pérez, Nogueira, and Moreno-Navarro [2013]. Accattoli and Guerrieri [2016] explore open call-by-value and Accattoli, Barenbaum, and Mazza [2015] study a (call-by-name) machine based on the linear substitution calculus for reduction to strong normal form. None of these mentioned works address however strong call-by-need.

The current implementation of The Coq Development Team [2017] uses an abstract machine implemented by B. Barras which computes strong normal form of terms using a call-by-need like strategy. Unfortunately, no completeness proof of this implementation seems to exist. Moreover, sharing does not seem to be fully exploited as we do here. More precisely, the aforementioned implementation is organized in two levels: pure terms (without explicit substitutions) of the lambda-calculus to be substituted, and a closure environment used to perform substitution. Although the substitution operation is only performed on needed occurrences of free variables of pure terms, the two level architecture does not provide as much sharing as expected. Thus for example, given a term $x[x\backslash yy][y\backslash M]$, where M is a big normal structure, Barras' two-level implementation computes the closure $[x\backslash MM]$, while our implementation does not perform substitution of structures.

Call-by-Need and Optimality. Variants of call-by-need have for long been considered in folklore as the "optimal" weak evaluation strategies, in the sense that they reach a value using a minimal number of β-reduction steps. This piece of trivia has actually been formalised and proved correct by Balabonski [2013] for the slight optimization over call-by-need commonly known as full laziness [Jones 1987; Wadsworth 1971].

However, actual optimality for strong reduction, as described by Asperti and Guerrini [1998]; Lamping [1990]; Lévy [1980], is more complex and it can be easily seen that this optimality property is far from true for our strong call-by-need strategy. Consider for instance the term $(\lambda x.xx)\ (\lambda y.\mathtt{id}\ y)$: in normalising this term, our strategy computes twice the subterm $\mathtt{id}\ y$, which is clearly suboptimal. This is a direct consequence of our choice to define a strategy that is an iteration of (a symbolic extension of) the well-known weak call-by-need strategy. Indeed the first iteration,

which corresponds to actual weak call-by-need evaluation, requires substituting the value $\lambda y.\,\mathtt{id}\ y$ before normalising its body which, in this case, prevents the normalisation from being optimal with respect to the number of β-reduction steps.

Remark however that, while we could devise better strong strategies, optimality is most likely out of reach: on one side Barendregt, Bergstra, Klop, and Volken [1976] have proved that any optimal strategy for the full λ-calculus (without sharing) is undecidable, and furthermore, implementing optimal reduction seems to require sharing of contexts rather than subterms, which is not doable in a usual setting of explicit substitutions or `let` constructs.

9 CONCLUSION

We presented a notion of call-by-need strategy to strong normal form (reduction inside the body of abstractions and substitutions is admitted) and show that it is complete with respect to β-reduction, in other words, that if a pure lambda term t β-reduces to a β-normal form u, then *essentially* the same normal form can be attained starting from t but using the φ-strong call-by-need strategy to $\overset{\varphi}{\rightsquigarrow}$*-normal form*. A salient aspect of our proof is the use of non-idempotent intersection types to characterize needed normalisation: β-weakly normalising pure lambda terms are typable and moreover, typability implies weak normalisation in the strong call-by-need calculus. Also, we have proved that the prefix of any reduction sequence constructed from $\overset{\varphi}{\rightsquigarrow}$ coincides with weak call-by-need.

In forthcoming work, we will report on two natural extensions of this paper: Kahn-style natural semantics of strong call-by-need, and an abstract machine implementing our strategy. The transitions of a call-by-need abstract machine are relatively more complex than in the call-by-name case, the main reason being that evaluation might take place inside the environment; *eg.* when evaluating $(xx)[x\backslash \mathtt{id}\ \mathtt{id}]$ the machine first creates an association $x \mapsto \mathtt{id}\ \mathtt{id}$ on the environment and then, after evaluating $\mathtt{id}\ \mathtt{id}$, it must update the binding for x. Moreover, the machine must take care to evaluate a term only as much as demanded by the context. For example, a term like $\lambda x.x(\mathtt{id}\ \mathtt{id})$ must be evaluated to full-normal form if it is at the top level, but not if it has been provided an argument t, in which case a beta-reduction $(\lambda x.x(\mathtt{id}\ \mathtt{id}))t \to (x(\mathtt{id}\ \mathtt{id}))[x\backslash t]$ should take place instead.

REFERENCES

Beniamino Accattoli. 2012. 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 (LIPIcs)*, Ashish Tiwari (Ed.), Vol. 15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 6–21.

Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. 2014. Distilling abstract machines. In *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming (ICFP'14), Gothenburg, Sweden, September 1-3, 2014*, Johan Jeuring and Manuel M. T. Chakravarty (Eds.). ACM, 363–376.

Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. 2015. A Strong Distillery. In *Programming Languages and Systems - 13th Asian Symposium (APLAS'15), Pohang, South Korea, November 30 - December 2, 2015. (LNCS)*, Xinyu Feng and Sungwoo Park (Eds.), Vol. 9458. Springer Verlag, 231–250.

Beniamino Accattoli and Giulio Guerrieri. 2016. Open Call-by-Value. In *Programming Languages and Systems - 14th Asian Symposium (APLAS'16), Hanoi, Vietnam, November 21-23, 2016. (LNCS)*, Atsushi Igarashi (Ed.), Vol. 10017. Springer, 206–226.

Beniamino Accattoli and Delia Kesner. 2010. The Structural *lambda*-Calculus. In *Computer Science Logic, 24th International Workshop (CSL'10), 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. (LNCS)*, Anuj Dawar and Helmut Veith (Eds.), Vol. 6247. Springer Verlag, 381–395.

Beniamino Accattoli and Ugo Dal Lago. 2016. (Leftmost-Outermost) Beta Reduction is Invariant, Indeed. *Logical Methods in Computer Science* 12, 1 (2016), 1–46.

Zena M. Ariola and Matthias Felleisen. 1997. The Call-By-Need Lambda Calculus. *Journal of Functional Programming* 7, 3 (1997), 265–301.

Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. 1995. The 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*, Ron K. Cytron and Peter Lee (Eds.). ACM Press, 233–246.

Andrea Asperti and Stefano Guerrini. 1998. *The Optimal Implementation of Functional Programming Languages*. Cambridge Tracts in Theoretical Computer Science, Vol. 45. Cambridge University Press.

Thibaut Balabonski. 2013. Weak optimality, and the meaning of sharing. In *ACM SIGPLAN International Conference on Functional Programming (ICFP'13), Boston, MA, USA - September 25 - 27, 2013*, Greg Morrisett and Tarmo Uustalu (Eds.). ACM, 263–274.

Hendrik Pieter Barendregt, Jan A. Bergstra, Jan Willem Klop, and Henri Volken. 1976. *Some Notes on Lambda Reduction*. Technical Report 22. University of Utrecht, Department of mathematics. 13–53 pages.

B. Barras. 2017. Personal Communication. (February 2017).

Alexis Bernadet and Stéphane Lengrand. 2011. Complexity of strongly normalising λ-terms via non-idempotent intersection types. In *Foundations of Software Science and Computation Structures (FOSSACS) (LNCS)*, Martin Hofmann (Ed.), Vol. 6604. Springer Verlag, 88–107.

Alexis Bernadet and Stéphane Lengrand. 2013. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science* 9, 4 (2013), 1–46.

Mathieu Boespflug, Maxime Dénès, and Benjamin Grégoire. 2011. Full Reduction at Full Throttle. In *Certified Programs and Proofs - First International Conference (CPP'11), Kenting, Taiwan, December 7-9, 2011. (LNCS)*, Jean-Pierre Jouannaud and Zhong Shao (Eds.), Vol. 7086. Springer, 362–377.

Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. 2017. Non-Idempotent Intersection Types for the Lambda-Calculus. *Logic Journal of the IGPL* (2017). To appear.

Stephen Chang and Matthias Felleisen. 2012. The Call-by-Need Lambda Calculus, Revisited. In *Programming Languages and Systems - 21st European Symposium on Programming (ESOP'12), Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. (LNCS)*, Helmut Seidl (Ed.), Vol. 7211. Springer Verlag, 128–147.

Mario Coppo and Mariangiola Dezani-Ciancaglini. 1980. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic* 21, 4 (1980), 685–693.

Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. 1981. Functional Characters of Solvable Terms. *Mathematical Logic Quarterly* 27, 2-6 (1981), 45–58.

Pierre Crégut. 1990. An Abstract Machine for Lambda-Terms Normalization. In *LISP and Functional Programming*. ACM, 333–340.

Pierre Crégut. 2007. Strongly reducing variants of the Krivine abstract machine. *Higher-Order and Symbolic Computation* 20, 3 (2007), 209–230.

Olivier Danvy and Ian Zerny. 2013. A synthetic operational account of call-by-need evaluation, See [Peña and Schrijvers 2013], 97–108.

Erika De Benedetti and Simona Ronchi Della Rocca. 2013. Bounding normalization time through intersection types. In *Proceedings of Sixth Workshop on Intersection Types and Related Systems (ITRS) (Electronic Proceedings in Theoretical Computer Science)*, Luca Paolini (Ed.), Vol. 121. Cornell University Library, 48–57.

Daniel de Carvalho. 2007. *Sémantiques de la logique linéaire et temps de calcul.* Ph.D. Dissertation. Universit é Aix-Marseille II.

Daniel de Carvalho. 2009. Execution Time of lambda-Terms via Denotational Semantics and Intersection Types. *CoRR* abs/0905.4251 (2009), 1–36.

Thomas Ehrhard and Laurent Regnier. 2006. Böhm Trees, Krivine's Machine and the Taylor Expansion of Lambda-Terms. In *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe (CiE'06), Swansea, UK, June 30-July 5, 2006. (LNCS)*, Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker (Eds.), Vol. 3988. Springer Verlag, 186–197.

Álvaro García-Pérez, Pablo Nogueira, and Juan José Moreno-Navarro. 2013. Deriving the full-reducing Krivine machine from the small-step operational semantics of normal order, See [Peña and Schrijvers 2013], 85–96.

Philippa Gardner. 1994. Discovering Needed Reductions Using Type Theory. In *International Conference on Theoretical Aspects of Computer Software (TACS'94) (LNCS)*, Masami Hagiya and John C. Mitchell (Eds.), Vol. 789. Springer Verlag, 555–574.

Benjamin Grégoire and Xavier Leroy. 2002. A compiled implementation of strong reduction. In *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP '02), Pittsburgh, Pennsylvania, USA, October 4-6, 2002.*, Mitchell Wand and Simon L. Peyton Jones (Eds.). ACM, 235–246.

Peter Henderson and James H. Morris. 1976. A Lazy Evaluator. In *Conference Record of the Third ACM Symposium on Principles of Programming Languages (POPL'76), Atlanta, Georgia, USA, January 1976*, Susan L. Graham, Robert M. Graham,

Michael A. Harrison, William I. Grosky, and Jeffrey D. Ullman (Eds.). ACM Press, 95–103.

Simon L. Peyton Jones. 1987. *The Implementation of Functional Programming Languages.* Prentice-Hall.

Delia Kesner. 2009. A Theory of Explicit Substitutions with Safe and Full Composition. *Logical Methods in Computer Science* 5, 3 (2009), 1–29. http://arxiv.org/abs/0905.2539

Delia Kesner. 2016. Reasoning About Call-by-need by Means of Types. In *Foundations of Software Science and Computation Structures - 19th International Conference (FOSSACS'16), Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016. (LNCS)*, Bart Jacobs and Christof Löding (Eds.), Vol. 9634. Springer Verlag, 424–441.

Delia Kesner and Daniel Ventura. 2014. 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. (LNCS)*, Josep Díaz, Ivan Lanese, and Davide Sangiorgi (Eds.), Vol. 8705. Springer Verlag, 296–310.

Delia Kesner and Daniel Ventura. 2015. A resource aware computational interpretation for Herberlin's syntax. In *Theoretical Aspects of Computing (ICTAC) (LNCS)*, Martin Leucker, Camilo Rueda, and Frank D. Valencia (Eds.), Vol. 9399. Springer Verlag, 1–16.

Assaf J. Kfoury. 2000. A linearization of the Lambda-calculus and consequences. *J. Log. Comput.* 10, 3 (2000), 411–436. https://doi.org/10.1093/logcom/10.3.411

Assaf J. Kfoury and Joe Wells. 2004. Principality and type inference for intersection types using expansion variables. *Theoretical Computer Science* 311, 1-3 (2004), 1–70.

Jean-Louis Krivine. 1993. *Lambda-calculus, types and models.* Ellis Horwood.

John Lamping. 1990. An algorithm for optimal lambda calculus reduction. In *Procedings of POPL*. ACM, San Francisco, California, 16–30.

John Launchbury. 1993. A Natural Semantics for Lazy Evaluation. In *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, January 1993*, Mary S. Van Deusen and Bernard Lang (Eds.). ACM Press, 144–154.

Jean-Jacques Lévy. 1980. Optimal Reductions in the lambda-calculus. In *To Haskell Brooks Curry: Essays in Combinatory Logic, Lambda Calculus and formalism*, Roger Hindley and Jonathan P. Seldin (Eds.). Academic Press, 159–191.

John Maraist, Martin Odersky, and Philip Wadler. 1998. The Call-by-Need Lambda Calculus. *Journal of Functional Programming* 8, 3 (1998), 275–317.

Robin Milner. 2007. Local Bigraphs and Confluence: Two Conjectures: (Extended Abstract). *Electronic Notes in Theoretical Computer Science* 175, 3 (2007), 65–73.

Ricardo Peña and Tom Schrijvers (Eds.). 2013. *15th International Symposium on Principles and Practice of Declarative Programming (PPDP '13), Madrid, Spain, September 16-18, 2013.* ACM.

William Tait. 1967. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic* 32, 2 (1967), 198–212.

The Coq Development Team. 2017. The Coq Proof Assistant (v8.6). (2017). https://github.com/coq/coq.

Steffen van Bakel. 1992. Complete Restrictions of the Intersection Type Discipline. *Theoretical Computer Science* 102, 1 (1992), 135–163.

Christopher P. Wadsworth. 1971. *Semantics and Pragmatics of the Lambda Calculus.* Ph.D. Dissertation. Oxford University.