Pattern Matching and Fixed Points: Resource Types and Strong Call-By-Need

Pablo Barenbaum Eduardo Bonelli Kareem Mohamed

PPDP 2018
September 5, 2018
Frankfurt, Germany

Outline

Weak vs. strong reduction

Typical programming languages use **weak** reduction:

- Bodies of functions are not evaluated (until applied).
- Well-formed programs are assumed to be closed.
- The result of a computation is a weak head normal form.

Example (weak reduction in OCaml)

```
>>>   fun x -> x / 0
      = <fun>
      : int -> int
```

Weak vs. strong reduction

Contrast with **strong** reduction:

- To fully evaluate a function, the body must be evaluated.
- Programs may be open, involving symbolic variables.
- The result of a computation is a (strong) normal form.

Example (strong reduction in Coq)

```
>>>   Eval lazy in (fun x => 2 + id x)
      = fun x : nat => S (S x)
      : nat -> nat
```

Motivations to study strong reduction

Motivation: partial evaluation

$$\begin{array}{rcl} \mathsf{pow}\ x\ n & := & \textbf{if}\ n == 0\ \textbf{then}\ 1\ \textbf{else}\ x * \mathsf{pow}\ x\ (n-1) \\ \mathsf{square} & := & \lambda x.\ \mathsf{pow}\ x\ 2 \end{array}$$

$$\mathsf{square} = \lambda x.\ \mathsf{pow}\ x\ 2 \quad \rightsquigarrow \quad \lambda x.\ x * x$$

Motivations to study strong reduction

Motivation: implementation of proof assistants

Proof assistants based on dependent type theory (Coq, Agda, ...) typically include the following typing rule (**conversion**):

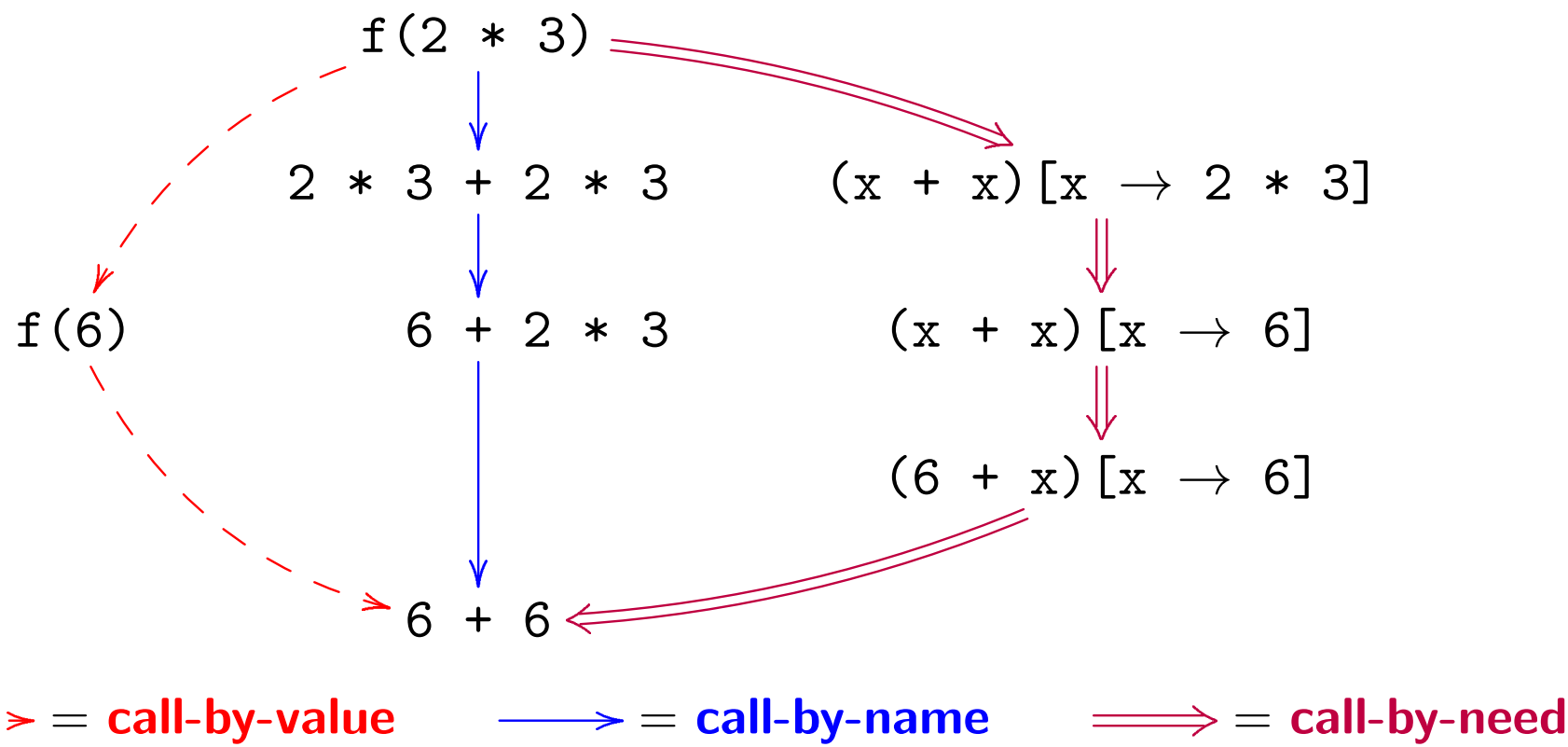
$$\frac{\Gamma \vdash t : A \qquad A \equiv B}{\Gamma \vdash t : B}$$

- To decide definitional equality $A \equiv B$, implementations compare the strong normal forms of A and B.

Weak evaluation strategies

Many well-known evaluation strategies for **weak reduction**.

For instance, if `f(x) = x + x`:



Goal of this work

- Define a **call-by-need** strategy for **strong** reduction.
- It must contemplate **pattern matching** and **recursion**, extending previous work by the first two authors with Balabonski and Kesner (ICFP'17).
- Show that the strategy is complete.

Plan

$\lambda_{\mathtt{e}}$ — Our starting point: an extended λ-calculus with pattern matching and fixed points.

$\lambda_{\mathtt{sh}}$ — A variant of $\lambda_{\mathtt{e}}$ with subterm sharing (explicit substitutions).

$\blacktriangleright_{\mathtt{sh}}$ — A deterministic **strong call-by-need** evaluation strategy for $\lambda_{\mathtt{sh}}$.

Main result (**Completeness**)

If $t \twoheadrightarrow_{\mathtt{e}} s$ for some $s \in NF(\lambda_{\mathtt{e}})$

then $t \blacktriangleright\!\blacktriangleright_{\mathtt{sh}} s'$ for some $s' \in NF(\blacktriangleright_{\mathtt{sh}})$.

Key technical tool

A **non-idempotent intersection type system** for $\lambda_{\mathtt{e}}/\lambda_{\mathtt{sh}}$.

Outline

The Extended Lambda Calculus (λ_e)

Untyped subset of Gallina, the specification language of Coq.

Proposed by Grégoire and Leroy (ICFP'02) to study strong call-by-value reduction.

The Extended Lambda Calculus (λ_e)

$$
\begin{array}{rcll}
t, s, \ldots & ::= & x & \text{variable} \\
& | & \lambda x.t & \text{abstraction} \\
& | & t\, s & \text{application} \\
& | & \mathtt{fix}(x.t) & \text{fixed-point} \\
& | & \mathbf{c} & \text{constant (\textit{e.g.} \textbf{true}, \textbf{nil}, \textbf{cons})} \\
& | & \mathtt{case}\ t\ \mathtt{of}\ \overline{(\mathbf{c}\bar{x} \Rightarrow t)} & \text{case expression}
\end{array}
$$

Reduction rules

$$
\begin{array}{rcl}
(\lambda x.t)s & \rightarrow_e & t\{x \rightarrow s\} \\
\mathtt{fix}(x.t) & \rightarrow_e & t\{x \rightarrow \mathtt{fix}(x.t)\} \\
\mathtt{case}\ \mathbf{c}_j \bar{t}\ \mathtt{of}\ (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} & \rightarrow_e & s_j\{\bar{x}_j \rightarrow \bar{t}\} \\
 & & \text{if } j \in I \text{ and } |\bar{t}| = |\bar{x}_j|
\end{array}
$$

The Theory of Sharing ($\lambda_{\mathtt{sh}}$)

A variant of λ_{e} with explicit substitutions, for sharing subterms.

Ancestors:

- The call-by-need lambda calculus
  of Ariola, Felleisen, Maraist, Odersky, and Wadler (1997).
- Explicit substitutions *at a distance*
  introduced by Accattoli and Kesner (2010).

The Theory of Sharing ($\lambda_{\mathtt{sh}}$)

$t, s, \ldots$	$::=$	x	variable
	$\mid$	$\lambda x.t$	abstraction
	$\mid$	$t\, s$	application
	$\mid$	$\mathtt{fix}(x.t)$	fixed-point
	$\mid$	$\mathbf{c}$	constant (*e.g.* **true**, **nil**, **cons**)
	$\mid$	$\mathtt{case}\ t\ \mathtt{of}\ \overline{(\mathbf{c}\bar{x} \Rightarrow t)}$	case expression
	$\mid$	$t[x \to s]$	explicit substitution

Substitution contexts	$\mathtt{L}$	$::=$	$\Box \mid \mathtt{L}[x \to t]$
Constant contexts	$\mathtt{A}$	$::=$	$\Box \mid \mathtt{A}\mathtt{L}\, t$
Values	v	$::=$	$\lambda x.t \mid \mathtt{A}[\mathbf{c}]$

The Theory of Sharing ($\lambda_{\mathtt{sh}}$)

Interaction rules

$$
\begin{array}{rcll}
(\lambda x.t)\mathtt{L}\,s & \to_{\mathtt{sh}} & t[x \to s] & \\
\mathtt{fix}(x.t) & \to_{\mathtt{sh}} & t[x \to \mathtt{fix}(x.t)] & \\
\mathtt{case}\ \mathtt{A}[\mathbf{c}_j]\mathtt{L}\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} & \to_{\mathtt{sh}} & s_j[\bar{x}_j \to \mathtt{A}]\mathtt{L} & \\
 & & \quad \text{if } j \in I \text{ and } |\mathtt{A}| = |\bar{x}_j| &
\end{array}
$$

where $t[\bar{x} \to \mathtt{A}]$ is defined recursively:

$$
\begin{array}{rcl}
t[\epsilon \to \Box] & := & t \\
t[\bar{x}, y \to \mathtt{A}\mathtt{L}t] & := & t[\bar{x} \to \mathtt{A}]\mathtt{L}[y \to t]
\end{array}
$$

Substitution rules

$$
\begin{array}{rcll}
\mathtt{C}[\![x]\!][x \to v\mathtt{L}] & \to_{\mathtt{sh}} & \mathtt{C}[\![v]\!][x \to v]\mathtt{L} & \text{if } \mathtt{C} \text{ is any context, } v \text{ is a value} \\
t[x \to s] & \to_{\mathtt{sh}} & t & \text{if } x \notin \mathrm{fv}(t)
\end{array}
$$

Example

Let:

$$\texttt{MAP} := \texttt{fix}(\mathit{map}.\,\lambda f.\,\lambda \ell.\ \texttt{case}\ \ell\ \texttt{of} \mid \mathbf{nil} \Rightarrow \mathbf{nil} \mid \mathbf{cons}\,x\,xs \Rightarrow \mathbf{cons}\,(f\,x)\,(\mathit{map}\,f\,xs))$$

Then:

$$\texttt{MAP}\ F\ (\mathbf{cons}\ A\ \mathbf{nil})$$

$$\twoheadrightarrow_{\texttt{sh}}\ (\texttt{case}\ \ell\ \texttt{of} \mid \mathbf{nil} \Rightarrow \mathbf{nil} \mid \mathbf{cons}\,x\,xs \Rightarrow \mathbf{cons}\,(f\,x)\,(\mathit{map}\,f\,xs))[\ell \to \mathbf{cons}\ A\ \mathbf{nil}][f \to F][\mathit{map} \to \texttt{MAP}]$$

$$\twoheadrightarrow_{\texttt{sh}}\ (\texttt{case}\ \mathbf{cons}\ A\ \mathbf{nil}\ \texttt{of} \mid \mathbf{nil} \Rightarrow \mathbf{nil} \mid \mathbf{cons}\,x\,xs \Rightarrow \mathbf{cons}\,(f\,x)\,(\mathit{map}\,f\,xs))[f \to F][\mathit{map} \to \texttt{MAP}]$$

$$\to_{\texttt{sh}}\ (\mathbf{cons}\,(f\,x)\,(\mathit{map}\,f\,xs))[x \to A][xs \to \mathbf{nil}][f \to F][\mathit{map} \to \texttt{MAP}]$$

Outline

The Strong Call-by-Need Strategy ($\blacktriangleright_{\mathtt{sh}}$)

The Strong Call-by-Need Strategy ($\blacktriangleright_{\mathtt{sh}}$)

Recall the popular saying:

$$\text{call-by-need} = \text{“sharing} + \text{lazy evaluation”}$$

The Theory of Sharing $\lambda_{\mathtt{sh}}$ is (just) a **calculus**:

- The rewriting relation $\to_{\mathtt{sh}}$ defines a theory of equality.
- It allows for sharing.
- It does not impose any evaluation order.

We define a deterministic evaluation **strategy** $\blacktriangleright_{\mathtt{sh}}$ to impose a lazy evaluation order.

The Strong Call-by-Need Strategy ($\blacktriangleright_{\mathtt{sh}}$)

Design principles of the Strong Call-by-Need Strategy $\blacktriangleright_{\mathtt{sh}}$

We follow the guidelines from the preceding ICFP'17 work:

- Evaluation focuses on a subterm t under a context $\mathtt{C}$.
- Evaluate t until reaching a **weak head normal form**.
  The possible weak head normal forms are roughly:

Abstractions	$\lambda x.t$	
Variable structures	$x\, t_1 \ldots t_n$	
Constant structures	$\mathbf{c}\, t_1 \ldots t_n$	
Stuck case expressions	$\mathtt{case}\ t\ \mathtt{of}\ \bar{b}$	if t cannot possibly match any branch.

  (Sprinkled with explicit substitutions).
- If $\mathtt{C}$ can interact with t, zoom out and continue.
  Otherwise, recursively evaluate the subterms of t.

The Strong Call-by-Need Strategy ($\blacktriangleright_{\mathtt{sh}}$)

Strong call-by-need evaluation is **context-dependent** along two dimensions.

Example (Interaction with the context)

In $(\mathbf{succ}\, t)$ the subterm t may be the focus of evaluation or not:

$$\begin{array}{rcl} \mathbf{succ}\,(\mathbf{succ}\, t) & \blacktriangleright_{\mathtt{sh}} & \mathbf{succ}\,(\mathbf{succ}\, t') \\ \underbrace{\mathtt{case}\ \mathbf{succ}\, t\ \mathtt{of}\ \mathbf{succ}\, x \Rightarrow x} & \blacktriangleright_{\mathtt{sh}} & x[x \to t] \end{array}$$

depending on whether $(\mathbf{succ}\, t)$ can interact with the context.

Example (Frozen variables)

In $(\mathtt{case}\ x\ \mathtt{of}\ \mathbf{zero} \Rightarrow t)$ the focus of evaluation may be x or t:

$$\begin{array}{rcl} \lambda x.\,(\mathtt{case}\ x\ \mathtt{of}\ \mathbf{zero} \Rightarrow t) & \blacktriangleright_{\mathtt{sh}} & \lambda x.\,(\mathtt{case}\ x\ \mathtt{of}\ \mathbf{zero} \Rightarrow t') \\ (\mathtt{case}\ x\ \mathtt{of}\ \mathbf{zero} \Rightarrow t)[x \to \mathbf{zero}] & \blacktriangleright_{\mathtt{sh}} & (\mathtt{case}\ \mathbf{zero}\ \mathtt{of}\ \mathbf{zero} \Rightarrow t)[x \to \mathbf{zero}] \end{array}$$

depending on whether x is **frozen** by the context or not.

The Strong Call-by-Need Strategy ($\blacktriangleright_{\mathtt{sh}}$)

For each **discriminator** h and each set of **frozen variables** ϑ we define a family $\mathcal{E}^{h}_{\vartheta}$ of **evaluation contexts**.

[Details in the paper.]

The Strong Call-by-Need Strategy ($\blacktriangleright_{\mathtt{sh}}$)

Formal definition of the Strong Call-by-Need Strategy ($\blacktriangleright_{\mathtt{sh}}$)

$$
\begin{array}{lcll}
\mathtt{C}[(\lambda x.t)\mathtt{L}\,s] & \blacktriangleright_{\mathtt{sh}} & \mathtt{C}[t[x \to s]] & \\
\mathtt{C}[\mathtt{fix}(x.t)] & \blacktriangleright_{\mathtt{sh}} & \mathtt{C}[t[x \to \mathtt{fix}(x.t)]] & \\
\mathtt{C}[\mathtt{case}\ \mathtt{A}[\mathbf{c}_j]\mathtt{L}\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I}] & \blacktriangleright_{\mathtt{sh}} & \mathtt{C}[s_j[\bar{x}_j \to \mathtt{A}]\mathtt{L}] & \\
 & & \qquad \text{if } j \in I \text{ and } |\mathtt{A}| = |\bar{x}_j| & \\
\mathtt{C}_1[\mathtt{C}_2[\![x]\!][x \to v\mathtt{L}]] & \blacktriangleright_{\mathtt{sh}} & \mathtt{C}_1[\mathtt{C}_2[\![v]\!][x \to v]\mathtt{L}] &
\end{array}
$$

In every case, $\mathtt{C}$ must be an evaluation context, $\mathtt{C} \in \mathcal{E}^h_\varnothing$.

(In the last case, $\mathtt{C} := \mathtt{C}_1[\mathtt{C}_2[\![\Box]\!][x \to v\mathtt{L}]]$).

The Strong Call-by-Need Strategy ($\blacktriangleright_{\mathtt{sh}}$)

Example (Strong call-by-need evaluation)

$$\lambda w.\underline{(\lambda x.\mathtt{case}\ x\ \mathtt{of}\ \mathbf{succ}\, y \Rightarrow x)((\lambda z.\mathbf{succ}\, z)\, w)}$$

$$\blacktriangleright_{\mathtt{sh}}\quad \lambda w.(\mathtt{case}\ x\ \mathtt{of}\ \mathbf{succ}\, y \Rightarrow x)[x \to \underline{(\lambda z.\mathbf{succ}\, z)\, w}]$$

$$\blacktriangleright_{\mathtt{sh}}\quad \lambda w.(\mathtt{case}\ \underline{x}\ \mathtt{of}\ \mathbf{succ}\, y \Rightarrow x)[x \to (\mathbf{succ}\, z)[z \to w]]$$

$$\blacktriangleright_{\mathtt{sh}}\quad \lambda w.(\underline{\mathtt{case}\ \mathbf{succ}\, z\ \mathtt{of}\ \mathbf{succ}\, y \Rightarrow x})[x \to \mathbf{succ}\, z][z \to w]$$

$$\blacktriangleright_{\mathtt{sh}}\quad \lambda w.\underline{x}[y \to z][x \to \mathbf{succ}\, z][z \to w]$$

$$\blacktriangleright_{\mathtt{sh}}\quad \lambda w.(\mathbf{succ}\, z)[y \to z][x \to \mathbf{succ}\, z][z \to w]$$

Outline

Completeness

Theorem (Completeness)

If $t \twoheadrightarrow_{\mathtt{e}} s$ *for some* $s \in NF(\lambda_{\mathtt{e}})$

then $t \blacktriangleright\!\blacktriangleright_{\mathtt{sh}} u$ *for some* $u \in NF(\blacktriangleright_{\mathtt{sh}})$.

Furthermore, $u^{\diamond} = s$.

$$t[x \to s]^{\diamond} := t^{\diamond}\{x \to s^{\diamond}\}$$

Completeness

Theorem (Completeness)

If $t \twoheadrightarrow_{\mathtt{e}} s$ *for some* $s \in NF(\lambda_{\mathtt{e}})$
then $t \blacktriangleright\!\blacktriangleright_{\mathtt{sh}} u$ *for some* $u \in NF(\blacktriangleright_{\mathtt{sh}})$.

Furthermore, $u^{\diamond} = s$.

$$t[x \to s]^{\diamond} := t^{\diamond}\{x \to s^{\diamond}\}$$

Proof

The proof is cut in three steps.

1. **Normalizing $\to$ typable.** If $t \in WN(\lambda_{\mathtt{e}})$ then t is **typable**.
2. **Typable $\to$ normalizing.** If t is **typable** then $t \in WN(\lambda_{\mathtt{sh}})$.
3. **Factorization.** If $t \in WN(\lambda_{\mathtt{sh}})$ then $t \blacktriangleright\!\blacktriangleright_{\mathtt{sh}} u$ and $u^{\diamond} = s$.

The key tool is a **non-idempotent intersection type system**.

Intersection Types for the Theory of Sharing

Reminder: non-idempotent intersection types

- Subterms are typed as many times as they are used.
- A subterm can be untyped (if it is not used).
- No unique/principal typing.
- Typability characterizes normalization:

$$\begin{array}{rcl} t \text{ has a head normal form} & \iff & t \text{ is typable} \\ t \text{ has a (strong) normal form} & \iff & t \text{ is typable} + \text{constraints} \end{array}$$

- Typability is undecidable.

The non-idempotent intersection type system $\mathcal{T}$

1. **Multi type ($\mathcal{M} = [\tau_1, \ldots, \tau_n]$).** — Type of a term that has many types simultaneously: $\mathcal{M}$ is a multiset of types.
2. **Function type ($\mathcal{M} \to \sigma$)** — Type of a function that uses its argument once for each type in $\mathcal{M}$, and returns σ.
3. **Datatype ($\mathbf{c}\mathcal{M}_1 \ldots \mathcal{M}_n$)** — Type of a constant **c** applied to n arguments, the i-th of which has type $\mathcal{M}_i$.
4. **Error type ($\langle \Bbbk\, \tau\, (\bar{\mathcal{M}}_i \Rightarrow \sigma_i)_{i=1}^{n} \rho_1 \ldots \rho_j \rangle \rho_{j+1} \ldots \rho_k$)** — Type of a stuck case expression, with condition of type τ, branches of types $(\bar{\mathcal{M}}_i \Rightarrow \sigma_i)$, that has been applied to arguments of types $\rho_1, \ldots, \rho_j$, and expects arguments of types $\rho_{j+1}, \ldots, \rho_k$.

[Formal details in the paper.]

Example

$$\lambda x.\mathbf{cons}\, x\, x \quad : \quad [\alpha, \beta, \gamma] \to \mathbf{cons}\, [\alpha, \gamma]\, [\beta]$$

$$\lambda x.\underbrace{(\texttt{case}\ x\ \texttt{of}\ \mathbf{succ}\, y \Rightarrow y)}_{\langle \Bbbk\, \alpha\, ([\gamma] \Rightarrow \gamma) \rangle \beta}\, x \quad : \quad [\alpha, \beta] \to \langle \Bbbk\, \alpha\, ([\gamma] \Rightarrow \gamma) \beta \rangle$$

Completeness (1/3)

Step 1: **Normalizing $\to$ typable.**

If $t \in WN(\lambda_e)$ then
there exist Γ, Σ, τ such that $\Gamma; \Sigma \vdash t : \tau$
and the judgment is **good**.

Proof

- Prove the particular case when $t \in NF(\lambda_e)$.
- Prove **subject expansion** for $\to_e$:

$$\begin{aligned} \text{If} \quad & \Gamma; \Sigma \vdash s : \tau \text{ and } t \to_e s \\ \text{then} \quad & \Gamma; \Sigma \vdash t : \tau. \end{aligned}$$

Completeness (2/3)

Step 2: **Typable $\to$ normalizing.**

If $\Gamma; \Sigma \vdash t : \tau$ and the judgment is **good**,
then $t \in WN(\lambda_{\mathtt{sh}})$.

Proof

- Define a measure for typing derivations.
- Prove **weighted subject reduction** for $\to_{\mathtt{sh}}$:

$$\text{If} \quad \frac{\pi}{\Gamma; \Sigma \vdash t : \tau} \quad \text{and } t \notin NF(\lambda_{\mathtt{sh}})$$

$$\text{then} \quad \text{there is a step } t \to_{\mathtt{sh}} s$$

$$\text{such that} \quad \frac{\pi'}{\Gamma; \Sigma \vdash s : \tau} \quad \text{and } \#(\pi) > \#(\pi').$$

Completeness (3/3)

Step 3: **Factorization**

If $t \twoheadrightarrow_{\mathtt{sh}} s$ with $s \in NF(\lambda_{\mathtt{sh}})$
then $t \blacktriangleright\!\!\blacktriangleright_{\mathtt{sh}} u$ and $u^{\diamond} = s$.

Proof

Steps $t \blacktriangleright_{\mathtt{sh}} s$ are called **external**.
Steps that are not external are **internal**, written $t \triangleright_{\mathtt{sh}} s$.

By exhaustive case analysis, show that any two consecutive steps $\triangleright_{\mathtt{sh}} \blacktriangleright_{\mathtt{sh}}$ can be swapped to obtain $\blacktriangleright_{\mathtt{sh}} \twoheadrightarrow_{\mathtt{sh}}$.

This can be iterated to obtain a **standardization** result:

$$(\; t \quad \twoheadrightarrow_{\mathtt{sh}} \quad s \;) \qquad \Longrightarrow \qquad (\; t \quad \blacktriangleright\!\!\blacktriangleright_{\mathtt{sh}} \quad u \quad \triangleright\!\!\triangleright_{\mathtt{sh}} \quad s \;)$$

Moreover, $r_1 \triangleright_{\mathtt{sh}} r_2$ implies $r_1^{\diamond} = r_2^{\diamond}$.

Conclusions

- We have proposed a strong call-by-need strategy $\blacktriangleright_{\mathtt{sh}}$.
- It extends preceding work with pattern matching and recursion.
- The main challenge is dealing with **stuck case expressions**.
- The strategy is complete with respect to Grégoire and Leroy's $\lambda_{\mathtt{e}}$-calculus.
- Byproducts:
  - A Theory of Sharing $\lambda_{\mathtt{sh}}$ for $\lambda_{\mathtt{e}}$.
  - A non-idempotent intersection type system $\mathcal{T}$ characterizing weak normalization.

Future work

- Decide $t \equiv_{\lambda_{\mathtt{e}}} s$ using $\blacktriangleright_{\mathtt{sh}}$ non-naïvely.
  (Naïvely: reduce to $\blacktriangleright_{\mathtt{sh}}$-normal form, unfold, and compare).
- Big step semantics, abstract machine.