Weak Optimality, and the Meaning of Sharing

Thibaut Balabonski
INRIA, Gallium team
thibaut.balabonski@inria.fr

Abstract

In this paper we investigate laziness and optimal evaluation strategies for functional programming languages. We consider the weak λ-calculus as a basis of functional programming languages, and we adapt to this setting the concepts of optimal reductions that were defined for the full λ-calculus. We prove that the usual implementation of call-by-need using sharing is optimal, that is, normalizing any λ-term with call-by-need requires exactly the same number of reduction steps as the shortest reduction sequence in the weak λ-calculus without sharing. Furthermore, we prove that optimal reduction sequences without sharing are not computable. Hence sharing is the only computable means to reach weak optimality.

Categories and Subject Descriptors I.1.3 [*Languages and Systems*]: Evaluation strategies

General Terms Theory, Languages

Keywords Strategies, Laziness, Weak reduction, Sharing, Optimality, Computability.

1. Introduction

The computation steps described by a functional program can usually be scheduled in many different ways. However, the schedules do not all have the same efficiency. Consider for instance the following program:

```
let    id  x  =  x
and  cst1  x  =  1
and  diag  x  =  (x, x)
in
     diag (cst1 (id 2))
```

Its *reduction space*, that is, the directed graph that sums up all the possible evaluations of the program, is pictured in Fig. 1. Each evaluation strategy choses a path in this graph.

In particular, *call-by-value* reaches the result in three evaluation steps, with one inefficiency: it evaluates the argument `id 2`, which is *not needed*, since its value is going to be discarded by the function `cst1`. *Call-by-name* also reaches the result in three evaluation steps, with another inefficiency: it performs twice the call to the function `cst1` that is *duplicated* by the function `diag`.

ICFP '13, September 25–27, 2013, Boston, MA, USA.

ACM 978-1-4503-2326-0/13/09…$15.00.
http://dx.doi.org/10.1145/2500365.2500606

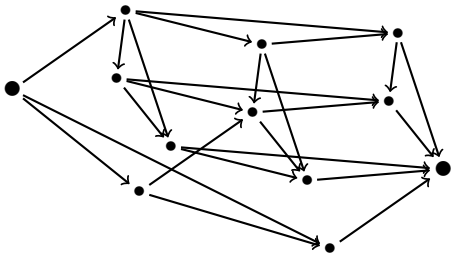


Figure 1. Reduction space

There are two ways of evaluating this program in only two steps. The first one is an hybrid strategy that first performs the call to `cst1` and then the call to `diag`. The second one, *call-by-need*, combines call-by-name with an additional mechanism of *memoization* or *sharing*, which enables a re-use of already-computed results, and which is meant to make up for duplications. This additional mechanism adds shortcuts in the original single-step reduction space.

Since its introduction by Wadsworth's in 1971, call-by-need is considered as a good strategy for avoiding at the same time non-needed and duplicated computation steps. The point of this paper is to turn this folkloric fact into a precise theorem.

In particular we will prove that call-by-need is an *optimal* strategy with respect to some reasonable restriction of the λ-calculus, that is, call-by-need always reaches the result of a computation in the least number of evaluation steps, or, call-by-need always picks the shortest path in some realistic reduction space.

The restriction that we consider comes from the following remark: the λ-calculus is too powerful for most functional programming languages. Indeed, while different languages use different strategies, most of them use only a subset of the reduction space known as *weak evaluation*, in which no anticipated evaluation of a function body is performed before the function is applied to its arguments. With this restriction to weak evaluation we lose some strategies, and the shortest path in the full reduction space of the λ-calculus may not be preserved in the weak reduction space, as shown in the following example.

Example 1.1.
Let a be a fresh variable. Consider the λ-term

$$t \quad = \quad (\lambda x.x(xa))(\lambda y.(\lambda z.z)y)$$

Its normal form is a, and it can be reached in four β-steps by first reducing $\lambda y.(\lambda z.z)y$ to $\lambda y.y$. If we restrict ourselves to weak reduction however, we need five steps to reach the result.

However, we consider weak evaluation as more realistic than the full λ-calculus, since this restriction is crucial for the implementation, and thus in this paper we investigate *weak optimality*,

that is, the evaluation strategies that are optimal in the restricted-, weak-reduction space.

The main contributions are the following:

- We adapt Lévy's theory of optimal evaluation to the weak λ-calculus (Section 4), and identify two strategies that reach optimality (Section 6). One of these weakly optimal strategies is the original *call-by-need*, which proves the good behavior of Wadsworth's technique. The second weakly optimal strategy does not use sharing at all, which proves that the new reduction sequences allowed by sharing are not shorter than the shortest single-step reduction sequences.
- We prove that optimal single-step reduction strategies cannot be computable (Section 3). This result completes the previous statement by showing that, while sharing does not enable strictly shorter reduction sequences, call-by-need nevertheless offers a computable way of reaching weak optimality.
- We use a labelling of the weak λ-calculus as a simple way to give a faithful account of all the causal dependencies between computation events (Section 4).
- We show that the weak λ-calculus, while it has the appearance of the λ-calculus, has the properties of a first-order rewriting system (Section 5).

Since it is a better fit for a conference format, we focus here on the case of the weak λ-calculus. However, the results of the paper hold for the more general case of weak reduction in orthogonal higher-order rewriting. The interested reader is referred to the author's doctoral dissertation [Bal12b].

2. The weak λ-calculus

In this section we recall the definitions and the basic properties of the weak λ-calculus. As in the previous paper [Bal12c], the λ-calculus is expressed in the higher-order rewriting framework of *Combinatory Reduction Systems (CRS)*. This may seem artificial at the beginning, but it will allow a clean handling of the labels and of the connection between weak reduction and first-order rewriting.

First, let us introduce the *CRS* in a nutshell. We see here only the basic syntax and mechanisms; a comprehensive presentation can be found in [KvOvR93].

Definition 2.1 (*Combinatory Reduction Systems / CRS*)**.** *The grammar of* **metaterms** *in a* CRS *is:*

$$t \quad ::= \quad x \mid [x]t \mid f(t_1, ..., t_n) \mid Z(t_1, ..., t_n)$$

where x is a **variable**, *$[x]$ denotes the* **binding** *of a variable, f is an n-ary* **function symbol** *taken in a* **signature** *$\mathcal{S}$, and Z is an n-ary* **metavariable**.

A **term** *is a metaterm without metavariable, and an n-ary* **context** *is a term that contains n occurrences of the special nullary symbol $\Box$ (the* **hole**). *A* **reduction rule** *is a pair $L \to R$ of closed metaterms satisfying the following conditions:*

- *the metavariables in L appear as $Z(x_1, ..., x_n)$ with $x_1, ..., x_n$ distinct bound variables; and*
- *all the metavariables of R also appear in L.*

A rule matches a term by application of a **valuation** *σ that maps n-ary metavariables to n-ary contexts avoiding variable capture.* **Reduction** *by a rule $L \to R$ with valuation σ in a context c is $c[L^\sigma] \to c[R^\sigma]$.*

Example 2.2.

The λ-calculus can be seen as a CRS *with:*

- *a signature $\mathcal{S}_\lambda = \{\lambda, @\}$ where the symbol λ is unary and the application symbol $@$ is binary;*
- *a unique reduction rule $@(\lambda([x]Z(x)), Z') \to Z(Z')$.*

The λ-term $(\lambda x.x)y$ is encoded in the CRS *term $@(\lambda([x]x), y)$. However, we will still use usual λ-calculus notation wherever this can lighten the presentation.*

Example 2.3.

First order rewriting systems, also called Term Rewriting Systems (TRS) *are also a particular case of* CRS*, where metaterms do not contain any binders.*

The absence of binders has some consequences. In particular, since any metavariable Z appearing in the left member L of a reduction rule $L \to R$ is supposed to appear as $Z(x_1, \ldots, x_n)$ where the x_i's are bound variables, we are restricted to nullary metavariables that appear simply as Z. This restricts a lot the reduction rules that can be defined, and explains the simpler behavior of first-order compared to higher-order rewriting.

In this paper we will consider a signature that strictly contains the one of the λ-calculus.

Definition 2.4 (Terms)**.** *From now on we consider the signature*

$$\mathcal{S} = \{\varepsilon, \lambda, @\} \cup \mathcal{F}$$

where:

- *ε is a unary symbol called* **dummy** *symbol, and*
- *$\mathcal{F} = \bigcup_{n\in\mathbb{N}} \mathcal{F}_n$ where for all n, $\mathcal{F}_n$ is a countable set of n-ary symbols called* **supercombinators**.

From now on, by **term** *we mean a CRS term over the signature $\mathcal{S}$. Write Λ for the set of terms. We use the usual notion of* **position** *in a term. Write $t|_p$ for the* **subterm** *of t rooted at position p, and call two positions p_1 and p_2* **disjoint** *if none is a prefix of the other. We also use the usual notion of* **free variables**. *Write $t^{\{x:=u\}}$ for the* **substitution** *by u of all the free occurrences of the variable x in t. Write $\mathtt{dom}(\sigma)$ for the domain of a n-ary substitution σ. A* **context** *is a term also containing the nullary symbol $\Box$, called* **hole**. *Replacement $c[t]$ of a hole in a context c by a term t is as usual.*

The supercombinator symbols in the sets $\mathcal{F}_n$ are used in Section 5 for the reduction of the weak λ-calculus to a first-order rewriting system. Until then they play no role and may be safely ignored.

The dummy symbol ε has no meaning in itself. It is needed for the causal interpretation of labels (Section 4.5), and serves in particular as a container for dynamically created labels. As a consequence, occurrences of ε should not interfere with β-reduction. This leads to the following countable set of rules to simulate β-reduction by allowing any number of ε's between the application and the λ-abstraction.

Definition 2.5 (Beta-reduction)**.** *The rules for β-reduction are as follows:*

$$\begin{array}{lrcl}
\beta_0\text{:} & @(\lambda x.Z(x), Z') & \to_\beta & \varepsilon(Z(Z')) \\
\beta_1\text{:} & @(\varepsilon(\lambda x.Z(x)), Z') & \to_\beta & \varepsilon(Z(Z')) \\
\beta_2\text{:} & @(\varepsilon(\varepsilon(\lambda x.Z(x))), Z') & \to_\beta & \varepsilon(Z(Z')) \\
\ldots &&&
\end{array}$$

Write $\rho : t \to t'$ a reduction step ρ from a term t to a term t'. Write $\mathtt{src}(\rho) = t$ for the **source** *of ρ, $\mathtt{tgt}(\rho) = t'$ for its* **target**, *and $\mathtt{root}(\rho)$ for the position in $\mathtt{src}(\rho)$ of the redex reduced by ρ. A* **normal form** *is a term that is the source of no reduction step. The usual notions of* **ancestors** *and* **descendants** *[Ter03], which track subterms along reduction in the λ-calculus are straightforwardly adapted[1], as illustrated in Example 2.6. A* **residual** *of a redex r is a*

[1] The book [Ter03] gives several definitions of descendants that differ in particular in how they treat variables. In this paper we only need to consider the descendants of applications, hence the simplest definition is enough.

descendant of r which is still a redex. A redex r in the target $\mathtt{tgt}(\rho)$ of a reduction step ρ is **created** *by ρ if it is not a residual of a redex in the source $\mathtt{src}(\rho)$. A* **development** *of a set of redexes $\mathcal{R}$ in a term t is a reduction sequence $\varrho : t \twoheadrightarrow t'$ where each step reduces a residual of a redex in $\mathcal{R}$. A development ϱ is* **complete** *if there is no remaining residual of any redex in $\mathcal{R}$. A redex r in a term t is* **needed** *if any reduction sequence from t to a normal form reduces at least one residual of r.*

The ε's in the right hand sides of the β-rules are used for the correct labelling of collapsing reductions (Section 4.5). The use of the dummy symbol ε is inspired by the notion of *expansion* in term rewriting systems [Ter03, Chap. 8].

Example 2.6.

The rule β_1 enables a reduction step

$$\rho \quad : \quad @(\varepsilon(\lambda x.@(x,x)),y) \quad \to \quad \varepsilon(@(y,y))$$

The two occurrences of y in the target are the descendants of the y in the source, and the latter is the ancestor of the formers. The ε in the source has no descendant and the ε in the target has no ancestor.

In the usual, non-weak λ-calculus, β-reduction can be applied in any context. In contrast, the weak λ-calculus introduces some restrictions on the contexts in which a reduction rule can be applied. Intuitively, reduction will be forbidden at any position (called *frozen*) that belongs to the "effective scope" of a binder, which we call *skeleton*. This restriction is formally based on the following definitions.

Definition 2.7 (Free expression)**.** *A* **free expression**[2] *of a λ-term $\lambda x.t$ is a subterm s of t such that any free variable occurrence in s is also free in $\lambda x.t$. A free expression is* **maximal** *if it is not a subterm of another free expression.*

Definition 2.8 (Prefix)**.** *A n-***ary prefix** *of a λ-term t is a n-ary context c such that there are n λ-terms $t_1, \ldots, t_n$ satisfying $t = c[t_1, \ldots, t_n]$.*

We will be mostly interested in the prefix that is obtained by removing all the maximal free expressions of a λ-abstraction $\lambda x.t$, which represents the parts of the term that are not independent from the binder, and which we call **skeleton** of $\lambda x.t$. Example 2.9 illustrates this notion, and definition 2.10 then gives a direct definition.

Example 2.9.

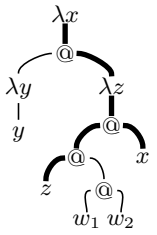
Suppose w_1 and w_2 are two variables, and consider the λ-term

$$t = \lambda x.(\lambda y.y)(\lambda z.z(w_1 w_2)x)$$

Its maximal free expressions are $\lambda y.y$ and $w_1 w_2$, while its skeleton is the binary prefix

$$\lambda x.\Box(\lambda z.z\Box x)$$

The skeleton is marked with bold lines in the following picture.



Definition 2.10 (Skeleton)**.** *For any set of variables θ, call θ-***skeleton** *of a λ-term t and write $\langle\langle t\rangle\rangle^\theta$ for the prefix of t defined as follows:*

$$\begin{array}{rcll} \langle\langle t\rangle\rangle^\theta & = & \Box & \theta \cap \mathtt{fv}(t) = \emptyset \\ \textit{Otherwise:} & & & \\ \langle\langle x\rangle\rangle^\theta & = & x & x \in \theta \\ \langle\langle \varepsilon(t)\rangle\rangle^\theta & = & \varepsilon(\langle\langle t\rangle\rangle^\theta) & \\ \langle\langle \lambda x.t\rangle\rangle^\theta & = & \lambda x.\langle\langle t\rangle\rangle^{\theta\cup\{x\}} & \\ \langle\langle t_1 t_2\rangle\rangle^\theta & = & \langle\langle t_1\rangle\rangle^\theta \langle\langle t_2\rangle\rangle^\theta & \\ \langle\langle f(t_1,\ldots,t_n)\rangle\rangle^\theta & = & f(\langle\langle t_1\rangle\rangle^\theta,\ldots,\langle\langle t_n\rangle\rangle^\theta) & \end{array}$$

Call **skeleton** *of a λ-abstraction $\lambda x.t$ its prefix $\lambda x.\langle\langle t\rangle\rangle^{\{x\}}$. Write $\langle\langle\Lambda\rangle\rangle$ for the set of all skeletons.*

Definition 2.11 (Frozen positions)**.** *A position p of a λ-term t is called* **frozen** *if it is contained in the skeleton of a λ-abstraction of t. Call a* **frozen redex** *any β-redex whose main @ symbol (the root of the redex) occurs at a frozen position.*

Definition 2.12 (Weak β-reduction)**.** **Weak β-reduction** *is defined by the application of the β-reduction rules to any β-redex that is not frozen.*

From now on we call **redex** only a *non-frozen* redex, that is a redex for weak reduction. A **weak normal form** is a term that contains no (non-frozen) redex.

Example 2.13.

Let a be a fresh variable. Consider the λ-term

$$t \quad = \quad \lambda x.@(\lambda y.y, @(x, @(\lambda z.z, a)))$$

Its frozen positions are ϵ, 1, 12, and 121 (corresponding to λx, x, and the two applications in between). Hence the redex $@(\lambda y.y, @(x, @(\lambda z.z, a)))$ is frozen, while the redex $@(\lambda z.z, a)$ is not. The weak normal form of t is $\lambda x.@(\lambda y.y, @(x, a))$.

This notion of weak β-reduction first appeared in works by Howard [How70] and by Çagman and Hindley [cH98], as a mean to describe in the syntax of the λ-calculus the reduction behavior of the combinators S and K of Combinatory Logic. For this reason it has been called *combinatory weak reduction*. In this setting, Çagman and Hindley proved in particular that the weak λ-calculus had the Church-Rosser property.

Proposition 2.14 (Church-Rosser, Çagman and Hindley [cH98])**.** *The weak λ-calculus is confluent.*

3. (Un-)Computability

In this section we prove the Uncomputability Theorem 3.14, which states that any optimal strategy for the weak λ-calculus, that is any reduction strategy that selects a shortest weak reduction sequence for any λ-term, is necessarily uncomputable. A similar result has already been proved by Barendregt et al. for the usual non-weak λ-calculus [BBKV76]. However, a key step of their proof does not hold if we restrict ourselves to weak reduction. In Section 3.1 we review the structure of the proof of Barendregt et al. and show where it gets broken by weak reduction. Then in Section 3.2 we patch their proof thanks to an original use of the λ-lifting transformation.

In this whole section we ignore the dummy symbol ε and the supercombinator symbols in $\mathcal{F}$.

3.1 Barendregt et al.'s proof of uncomputability

The proof by Barendregt et al. proceeds by exhibiting two recursively inseparable sets of λ-terms $\mathcal{A}$ and $\mathcal{B}$[3] and by showing how to separate them using a hypothetical optimal reduction strategy. The set $\mathcal{A}$ (resp. $\mathcal{B}$) is defined as "the set of all λ-terms admitting some

[2] Originally called *abstractable expression* by Wadsworth.

[3] $\mathcal{A}$ and $\mathcal{B}$ are **recursively inseparable** if there is no recursive set $\mathcal{C}$ such that $\mathcal{A} \subset \mathcal{C}$ and $\mathcal{B} \cap \mathcal{C} = \emptyset$.

cleverly-chosen t_a (resp. t_b) as normal form", and these two sets $\mathcal{A}$ and $\mathcal{B}$ are known to be recursively inseparable thanks to a result by Scott [Sco68].

The two sets $\mathcal{A}$ and $\mathcal{B}$ are then separated using a total computable function $\varphi(.)$ on λ-terms such that:

- Each $\varphi(t)$ contains exactly two β-redexes, written r_t^a and r_t^b.
- From $\varphi(t)$, any optimal reduction strategy reduces r_t^a if $t \twoheadrightarrow t_a$, and r_t^b if $t \twoheadrightarrow t_b$ (and nothing is specified if t reduces neither to t_a nor to t_b).

Their first step in defining the λ-term $\varphi(t)$ is to build an *enumerator*, that is a λ-term e in normal form such that, if n_t is a Gödel number for a λ-term t, and if $\ulcorner n_t \urcorner$ is a representation in the λ-calculus of n_t, then $e\ulcorner n_t \urcorner \twoheadrightarrow t$. This enumerator e works by building applicative combinations of the standard S, K, and I combinators. In particular $e\ulcorner n_t \urcorner$ reduces to $(t)_{CL}$, the usual encoding of t in combinatory logics S, K, I. Then, after reverting all the combinators into λ-terms, which gives a λ-term $((t)_{CL})_\lambda$, it is folklore that $((t)_{CL})_\lambda \twoheadrightarrow t$.

However, the latter is not true anymore in the weak λ-calculus, as the following example shows.

Example 3.1.
Consider the λ-term

$$t = \lambda x.xx$$

Then we have

$$\begin{array}{rcl}(t)_{CL} & = & SII \\ ((t)_{CL})_\lambda & = & (\lambda xyz.xz(yz))(\lambda x.x)(\lambda x.x)\end{array}$$

and $((t)_{CL})_\lambda \twoheadrightarrow t$ in the non-weak λ-calculus. However, the weak normal form of $((t)_{CL})_\lambda$ is

$$\lambda z.((\lambda x.x)z)((\lambda x.x)z)$$

Hence we do not have $((t)_{CL})_\lambda \twoheadrightarrow t$ in the weak λ-calculus.

The next section provides a new function $\varphi(.)$ that is suited for use with the weak λ-calculus.

3.2 Uncomputability for the weak λ-calculus

First, let us state that Scott's inseparability result is still valid for the weak λ-calculus.

Proposition 3.2 (Inseparability, Scott [Sco68])**.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two non-empty sets of λ-terms that are closed under weak β-reduction. Then $\mathcal{A}$ and $\mathcal{B}$ are recursively inseparable.*

Proof. Similar to Barendregt's proof [Bar84, Chap. 6], that does not need non-weak β-reduction. □

The main mechanism of our function $\varphi(.)$ is a decomposition of a λ-term into several components, all of which are in weak normal form. We rely on the fact that any skeleton is a normal form for weak reduction (all the positions of a skeleton are indeed frozen), and we use a technique inspired by λ-lifting to represent a λ-term as an assembly of skeletons.

The first step, corresponding to what Danvy and Schultz call "parameter lifting" [DS00], makes each λ-abstraction closed by extracting its maximal free expressions[4].

[4] Note that ordinary λ-lifting extracts only the free variables; switching to free expressions corresponds to fully lazy λ-lifting [PJ87].

Definition 3.3 (Free Expressions Lifting ($\mathtt{fel}$))**.**

$$\begin{array}{rcl}\mathtt{fel}(x) & = & x \\ \mathtt{fel}(t_1 t_2) & = & \mathtt{fel}(t_1)\mathtt{fel}(t_2) \\ \mathtt{fel}(\lambda x.t) & = & (\lambda y_1 \dots y_n x.s[y_1, \dots, y_n])\mathtt{fel}(t_1)\dots\mathtt{fel}(t_n) \\ & & s = \langle\langle t \rangle\rangle^{\{x\}} \\ & & t = s[t_1, \dots, t_n] \\ & & y_1, \dots, y_n \textit{ fresh variables}\end{array}$$

Lemma 3.4 (Free expressions lifting)**.** *For any λ-term t we have $\mathtt{fel}(t) \twoheadrightarrow t$.*

Proof. By induction on t. □

Example 3.5.
Suppose a, b and c are three free variables and consider the term

$$t \quad = \quad \lambda x_1.((\lambda x_2.x_2 a)b)(x_1 c)$$

Then free expression lifting proceeds as follows:

$$\begin{array}{rcl}\mathtt{fel}(t) & = & (\lambda y_1 y_2 x_1.y_1(x_1 y_2))\mathtt{fel}((\lambda x_2.x_2 a)b)\mathtt{fel}(c) \\ & = & (\lambda y_1 y_2 x_1.y_1(x_1 y_2))(\mathtt{fel}(\lambda x_2.x_2 a)\mathtt{fel}(b))\mathtt{fel}(c) \\ & = & (\lambda y_1 y_2 x_1.y_1(x_1 y_2))((\lambda y_1' x_2.x_2 y_1')\mathtt{fel}(a)\mathtt{fel}(b))\mathtt{fel}(c) \\ & = & (\lambda y_1 y_2 x_1.y_1(x_1 y_2))((\lambda y_1' x_2.x_2 y_1')ab)c\end{array}$$

Moreover, we have $\mathtt{fel}(t) \twoheadrightarrow t$ in three weak reduction steps.

The second step, corresponding to what Danvy and Schultz call "block floating", moves each function to toplevel. This is done by replacing each skeleton by a fresh variable and mapping these new variables to the skeletons they denote.

Definition 3.6 (Skeletons Floating ($\mathtt{sf}$))**.**

$$\begin{array}{rcl}\mathtt{sf}(x) & = & (x, \emptyset) \\ \mathtt{sf}(t_1 t_2) & = & (u_1 u_2, \sigma_1 \uplus \sigma_2) \\ & & \forall i \in \{1, 2\}, \mathtt{sf}(t_i) = (u_i, \sigma_i) \\ \mathtt{sf}(\lambda x.t) & = & (z, \{z := \lambda x.t\}) \qquad z \textit{ fresh}\end{array}$$

where $\uplus$ is the union of two substitutions with disjoint domains.

Lemma 3.7 (Skeletons floating)**.** *Let t be a λ-term and p a position. Write $\mathtt{sf}(t) = (u, \sigma)$. Then $u^\sigma = t$.*

Proof. By induction on t. □

Example 3.8.
Consider the term $\mathtt{fel}(t)$ obtained at the end of example 3.5. Skeletons floating $\mathtt{sf}(\mathtt{fel}(t))$ yields the term

$$u \quad = \quad z_1(z_2 ab)c$$

and the substitution

$$\begin{array}{rcl}\sigma & = & \{\, z_1 := \lambda y_1 y_2 x_1.y_1(x_1 y_2)\,; \\ & & \quad z_2 := \lambda y_1' x_2.x_2 y_1' \quad \}\end{array}$$

such as $u^\sigma = \mathtt{fel}(t)$.

Finally, the result of the two previous steps allows us to decompose a λ-term t as an application $s_0 s_1 \dots s_n$ with $n \geq 1$ and where each s_i is a weak normal form.

Definition 3.9 (Decomposition)**.** *Let t be a λ-term. Write*

$$\mathtt{sf}(\mathtt{fel}(t)) = (u, \sigma)$$

- *If $\mathtt{dom}(\sigma) = \emptyset$, then define*

$$\mathtt{dec}(t) = (\lambda z.u)I$$

- *Otherwise write $\mathtt{dom}(\sigma) = \{z_1, \dots, z_n\}$, and define*

$$\mathtt{dec}(t) = (\lambda z_1 \dots z_n.u)\sigma(z_1)\dots\sigma(z_n)$$

Lemma 3.10 (Decomposition)**.** *For any λ-term t we have* $\mathtt{dec}(t) \twoheadrightarrow t$.

Proof. By case on whether skeletons floating yields an empty substitution or not. □

Definition 3.11 (Separating function)**.** *The* **separating function** $\varphi(.)$ *is defined as follows: let t be a λ-term; write* $\mathtt{dec}(t) = s_0 s_1 ... s_n$*; then*

$$\varphi(t) = (\lambda x.xs_0x)(\lambda y.ys_1 \dots s_n(II))$$

where $I = \lambda z.z$ is the identity.

Example 3.12.

Consider the term t from example 3.5. Its decomposition is

$$\begin{array}{rcl} \mathtt{dec}(t) & = & (\lambda z_1 z_2.z_1(z_2 ab)c) \\ & & (\lambda y_1 y_2 x_1.y_1(x_1 y_2)) \\ & & (\lambda y_1' x_2.x_2 y_1') \end{array}$$

where each of the three components is a weak normal form, and where we can check that $\mathtt{dec}(t) \twoheadrightarrow t$ *in five weak reduction steps. Finally we define*

$$\begin{array}{rcl} \varphi(t) & = & (\lambda x.x(\lambda z_1 z_2.z_1(z_2 ab)c)x) \\ & & (\lambda y.y(\lambda y_1 y_2 x_1.y_1(x_1 y_2))(\lambda y_1' x_2.x_2 y_1')(II)) \end{array}$$

Lemma 3.13 (Separation)**.** *Consider an optimal strategy $\xrightarrow{\mathtt{opt}}$ for the weak λ-calculus, and let s_0, s_1, ..., s_n be λ-terms in weak normal form. Write $u = (\lambda x.xs_0x)(\lambda y.ys_1 \dots s_n(II))$, where $I = \lambda z.z$ is the identity. Then the following holds:*

- *If* $s_0 s_1 \dots s_n \twoheadrightarrow \lambda xyz.z$ *then* $u \xrightarrow{\mathtt{opt}} (\lambda y.ys_1 \dots s_n(II))s_0(\lambda y.ys_1 \dots s_n(II))$
- *If* $s_0 s_1 \dots s_n \twoheadrightarrow a$ *then* $u \xrightarrow{\mathtt{opt}} (\lambda x.xs_0x)(\lambda y.ys_1 \dots s_n I)$

Proof. Similar to Barendregt's [Bar84, Lemma 13.5.5]. □

Now we are able to prove the Uncomputability Theorem 3.14.

Theorem 3.14 (Uncomputability)**.** *There is no computable optimal reduction strategy for the weak λ-calculus.*

Proof. Define $\mathcal{A} = \{t \in \Lambda \mid t \twoheadrightarrow \lambda xyz.z\}$ and $\mathcal{B} = \{t \in \Lambda \mid t \twoheadrightarrow a\}$. Both sets are closed by weak β-reduction, hence by Scott's Inseparability Proposition 3.2 the sets $\mathcal{A}$ and $\mathcal{B}$ are recursively inseparable.

Now suppose there is a recursive optimal strategy $\xrightarrow{\mathtt{opt}}$ for the weak λ-calculus. Then the set

$$\mathcal{C} = \{t \in \Lambda \mid \varphi(t) \xrightarrow{\mathtt{opt}} t' \text{ at position } \epsilon\}$$

is computable. And yet, by Separation Lemma 3.13 the set $\mathcal{C}$ separates $\mathcal{A}$ and $\mathcal{B}$. Contradiction. □

Finally, we have proved that optimal reduction strategies for the weak λ-calculus cannot be computable. In the following sections we will investigate other approaches to optimality, in particular by using sharing.

4. Weak optimality

In this section we adapt Vuillemin-Lévy's theory of optimality [Lé80] to the weak λ-calculus, which will give a characterization of optimal reduction in a setting in which some sharing of computation is allowed.

4.1 A short introduction to Vuillemin-Lévy's optimality

This approach is intended to give a lower bound on the number of evaluation steps needed for normalizing a term, by characterizing an ideal case in which there is no unneeded computation step and no duplication at all. Since such an ideal reduction sequence cannot be reached in the pure λ-calculus –it is either non-existing or non-computable– this characterization of optimality proceeds by identifying *families* of computation steps that regroup all the duplicates of a given original redex. Hence, the families describe which redexes should be *shared* in order to achieve a computation without duplication.

The main idea behind families is the following: consider a reduction sequence $\varrho : t_1 \to t_2 \to \dots \to t_n$ and two redexes r_1 and r_2 in t_n. If there is a redex r_0 in one of the t_i's that is a common ancestor to r_1 and r_2, then r_1 and r_2 are in the same family (they are two duplicate copies of the original r_0). Hence two redexes are in the same family if their duplication could have been avoided by first reducing a common ancestor.

Example 4.1.

In this reduction sequence we follow the underlined expression $\underline{(\lambda z.z)(\lambda w.w)}$.

$$\begin{array}{rl} & (\lambda x.xx)(\lambda y.(\underline{(\lambda z.z)(\lambda w.w)}y)) \\ \to & (\lambda y.(\underline{(\lambda z.z)(\lambda w.w)}y))(\lambda y.(\underline{(\lambda z.z)(\lambda w.w)}y)) \\ \to & \underline{(\lambda z.z)(\lambda w.w)}(\lambda y.(\underline{(\lambda z.z)(\lambda w.w)}y)) \end{array}$$

The two occurrences of $\underline{(\lambda z.z)(\lambda w.w)}$ *are residuals of the first occurrences; they are in the same family.*

However, this criterion is not enough: two redexes can be in the same family even if their common ancestor is not visible in the sequence ϱ.

Example 4.2.

Now reduce the two underlined occurrences in the last term of example 4.1. We observe two new redexes (underlined below) that have no visible ancestor, and therefore no visible common ancestor.

$$\underline{(\lambda w.w)(\lambda y.\underline{(\lambda w.w)}y)}$$

Such a common ancestor exists somewhere though. Consider this other possible evaluation sequence and follow the underlined terms.

$$\begin{array}{rl} & (\lambda x.xx)(\lambda y.((\lambda z.z)\underline{(\lambda w.w)}y)) \\ \to & (\lambda x.xx)(\lambda y.(\underline{(\lambda w.w)}y)) \\ \to & (\lambda y.(\underline{(\lambda w.w)}y))(\lambda y.(\underline{(\lambda w.w)}y)) \\ \to & \underline{(\lambda w.w)(\lambda y.\underline{(\lambda w.w)}y)} \end{array}$$

Hence, by permuting the original sequence we have found a common ancestor to the two underlined redexes, which are thus in the same family. Reducing them both finally yields the term

$$\lambda y.y$$

Hence, the definition of families requires to reason globally on the reduction spaces and the permutations of their steps. To circumvent this problem, Lévy also gave a direct characterization of families, through a labelled λ-calculus that is shown to be stable by permutation of reduction steps. With this additional tool, the labels directly tell which redexes should be reduced in a shared way. Unfortunately the definition does not give an implementation. One of the difficulties is that optimal sharing in the λ-calculus requires sharing contexts (as the context $(\lambda w.w)\Box$ in the previous example). The first algorithms realizing optimal sharing in the non-weak λ-calculus were found ten years later [Lam90].

In this section we characterize families and optimality for the weak λ-calculus. We use an abstract characterization of Vuillemin-

Lévy optimality that has been proposed by Glauert and Khasidashvili.

4.2 Glauert & Khasidashvili's Deterministic Family Structures

Glauert and Khasidashvili [GK96] propose an abstract framework for characterizing optimality that is composed by two layers:

1. a well-behaved notion of residual (so-called *Deterministic Residual Structure* or *DRS*); and
2. a notion of family built above the notion of residual and enjoying additional good properties (so-called *Deterministic Family Structure*, or *DFS*).

Deterministic Residual Structures are defined by only two axioms, which are usually satisfied by the natural notions of residual in λ-calculus-based systems.

Definition 4.3 (Deterministic Residual Structure, DRS [GK96])**.** *A* **Deterministic Residual Structure (DRS)** *is a rewriting system equipped with a residual relation satisfying the following properties:*

- **[FD]** *All developments are terminating; all complete developments of a given set of redexes have the same target; all complete developments of a given set of redexes define the same residuals.*
- **[Acyclicity]** *Let r and r_e be two distinct redexes of a term t such that r erases[5] r_e. Then r_e does not erase r.*

The definition of *Deterministic Family Structures* is more complex. First, since the families of redexes in a term t are linked to the history of t –that is to [the permutations of] a reduction sequence of which t is the target– this definition considers families of *reduction steps with history*.

Definition 4.4 (Reduction step with history)**.** *A* **reduction step with history** *is a pair (ϱ, ρ) where ϱ is a reduction sequence, ρ is a reduction step, and $\mathtt{tgt}(\varrho) = \mathtt{src}(\rho)$. Two reduction steps with history (ϱ_1, ρ_1) and (ϱ_2, ρ_2) are* **coinitial** *if $\mathtt{src}(\varrho_1) = \mathtt{src}(\varrho_2)$.*

Second, we need to include the notions of residual and of permutation in the definition of families. For this they define a notion of *copy*, which has to be understood as a residual relation modulo permutation.

Definition 4.5 (Copy)**.** *Let (ϱ_1, ρ_1) and (ϱ_2, ρ_2) be two coinitial reduction steps with history. Suppose ϱ_1 has no residual after ϱ_2*[6]*. Let ϱ_2/ϱ_1 be a complete development of the residuals of ϱ_2 after ϱ_1. If ρ_2 is a residual of ρ_1 after ϱ_2/ϱ_1, then (ϱ_2, ρ_2) is said to be a* **copy** *of (ϱ_1, ρ_1), written $(\varrho_1, \rho_1) \rhd (\varrho_2, \rho_2)$.*

Definition 4.6 (Deterministic Family Structure [GK96])**.** *Let R be a DRS. Let $\simeq$ be an equivalence relation over coinitial reduction steps with history, whose equivalence classes are called* **families***. Write $\mathtt{Fam}(\varrho, \rho)$ for the family (*ie *the equivalence class) of a reduction step with history (ϱ, ρ). Let $\hookrightarrow$ be a binary relation over families, called* **contribution** *relation. The triple $(R, \simeq, \hookrightarrow)$ is a* **Deterministic Family Structure (DFS)** *if the following axioms are satisfied:*

- **[Initial]** *For any coinitial distinct reduction steps ρ_1 and ρ_2 we have $\mathtt{Fam}(\emptyset, \rho_1) \neq \mathtt{Fam}(\emptyset, \rho_2)$.*
- **[Copy]** *$\rhd \subseteq \simeq$.*
- **[FFD]** *Any reduction sequence that contracts redexes of a finite number of families is finite.*
- **[Creation]** *For any reduction step with history (ϱ, ρ), if ρ creates a reduction step ρ' then $\mathtt{Fam}(\varrho, \rho) \hookrightarrow \mathtt{Fam}(\varrho\rho, \rho')$.*
- **[Contribution]** *For any two families ϕ_1 and ϕ_2, we have $\phi_1 \hookrightarrow \phi_2$ if and only if for any reduction step with history $(\varrho_2, \rho_2) \in \phi_2$ there is a reduction step with history $(\varrho_1, \rho_1) \in \phi_1$ such that $\varrho_1\rho_1$ is a prefix of ϱ_2.*

Definition 4.7 (Family reduction)**.** *In a DFS, a* **family reduction step** *is a complete development of a set $\mathcal{R}$ of redexes of a term t which all belong to a given family ϕ. It is* **complete** *if $\mathcal{R}$ contains all the redexes of t that belong to the family ϕ. It is* **needed** *if one of the redexes of $\mathcal{R}$ is needed.*

Family reduction gives a new, richer reduction space in which shared evaluation is possible. From now on, we are interested in optimality in this bigger reduction space, that is in optimality with sharing.

Proposition 4.8 (Optimality in DFS [GK96])**.** *In any DFS, a family reduction sequence that reduces only complete needed families is optimal.*

Our goal for the rest of the section is to build a DFS over the weak λ-calculus, in order to characterize weak optimality. That means we need to give a concrete definition of the abstract concept of family and to prove that all the axioms are satisfied. Here is a summary of the required ingredients:

- we need to exhibit concrete notions of family ($\simeq$) and of contribution ($\hookrightarrow$); following Lévy, we will do this through a labelling of the weak λ-calculus presented in Section 4.4;
- in Section 4.3 we derive a classification of redex creations that both guide the definition of labels and enables the proof of the [Creation] axiom; and
- in Section 4.5 we show that the labels give a faithful account of the causal dependencies between reduction steps, which will be the core of the proof of the [Contribution] axiom.

The proof of the finite development axioms [FD] and [FFD] is postponed to Section 5, where we establish a connection between the weak labelled λ-calculus and a first-order rewriting system, which allows us to re-use finite development results from the first-order rewriting literature.

4.3 Redex creation in the weak λ-calculus

In this subsection we do not use the dummy symbol ε yet.

In the λ-calculus, a redex is a connection between a λ-abstraction and an argument. Hence a redex is created each time a new connection of this kind appears.

Example 4.9.

The three reduction steps below show the three ways of connecting a λ-abstraction $\underline{\lambda x.t}$ to an argument $\underline{u}$ described by Lévy [Lé78].

$$\begin{array}{rcl} ((\lambda y.\underline{\lambda x.t})v)\underline{u} & \to & (\lambda x.t^{\{y:=v\}})u \\ ((\lambda y.y)\underline{\lambda x.t})\underline{u} & \to & (\lambda x.t)u \\ (\lambda y.y\underline{u})\underline{\lambda x.t} & \to & (\lambda x.t)u \end{array}$$

These examples are valid independently of the restriction to weak reduction.

In the weak λ-calculus there is a new creation case, which can be observed when a frozen redex gets unfrozen.

Example 4.10.

In the source of the following reduction step, the expression $(\underline{\lambda x.yx})\underline{u}$ is not a redex because it is frozen by the variable y.

$$(\lambda y.((\underline{\lambda x.yx})\underline{u}))t \quad \to \quad (\lambda x.tx)u$$

For the rest of this paper, it is enough to classify the redex creations in two categories: either the left part of an application

[5] r **erases** r_e means that r_e has no residual after the reduction of r.

[6] This means that any "task" of ϱ_1 is either completed or erased by ϱ_2.

evaluates to a λ-abstraction, or an application subterm is affected by a substitution. This classification is formalized by Redex creation Theorem 4.12. We begin with a lemma that considers the case of a redex creation at the root of a term.

Lemma 4.11 (Root redex creation)**.** *Let $\rho : t = c[r] \to c[r'] = t'$ be a reduction step with $r = (\lambda x.u)v$ and $r' = u^{\{x := v\}}$. Suppose $c \neq \Box$ and t' is a redex $r_c = (\lambda y.u_c)v_c$ that is created by ρ. Then the following holds:*

(1) *$c = \Box v_c$ and $r' = \lambda y.u_c$.*

Proof. Case analysis on $c \neq \Box$:

- Case $c = \lambda z.c'$, or $c = f(t_1, \dots, c', \dots, t_n)$. Then there exists u' such that $t' = \lambda z.u'$ (resp. $t' = f(t_1, \dots, u', \dots, t_n)$). Contradiction with the hypothesis that t' is a redex.
- Case $c = (\lambda y.u_c)c'$. Then the ancestor t of r_c is already a redex. Contradiction with the hypothesis that the redex r_c is created by ρ.
- Case $c = c'v_c$. Case analysis on c':
  - Case $c' = \Box$. Then $c = \Box v_c$ and $r' = \lambda y.u_c$. Hence proposition (1) holds.
  - Case $c' = t_1 c''$ or $c''t_2$. Then $c = t_1 c'' v_c$ or $c = c''t_2 v_c$, and there exists a pair t'_1, t'_2 such that $t' = t'_1 t'_2 v_c$. Contradiction with the hypothesis that t' is a redex.
  - Case $c' = f(t_1, \dots, c'', \dots, t_n)$ is similar.
  - Case $c' = \lambda y.c''$. Then $c = (\lambda y.c'')v_c$, and the ancestor t of r_c is already a redex. Contradiction with the hypothesis that the redex r_c is created by ρ. □

Theorem 4.12 (Redex creation)**.** *Let $\rho : t = c[r] \to c[r'] = t'$ be a reduction step with $r = (\lambda x.u)v$ and $r' = u^{\{x := v\}}$. Let $r_c = (\lambda y.u_c)v_c$ be a redex in t' that is created by ρ. Write $t' = c_c[r_c]$. Then one of the following holds:*

(1) *$c = c_c[\Box v_c]$ and $r' = \lambda y.u_c$.*
(2) *The position $\mathtt{root}(r_c)$ is a descendant of a position of the x-skeleton of u.*

Proof. Case analysis on the relative positions $\mathtt{root}(r')$ and $\mathtt{root}(r_c)$.

- If $\mathtt{root}(r')$ and $\mathtt{root}(r_c)$ are disjoint, then the redex r_c already exists in t. Contradiction with the hypothesis that the redex r_c is created by ρ.
- If $\mathtt{root}(r_c)$ is a prefix of $\mathtt{root}(r')$, then there is a c' such that $r_c = c'[r']$ and $t' = c_c[c'[r']]$. Since r_c is created by ρ, its ancestor $c'[r]$ is not a redex in $t = c_c[c'[r]]$. Suppose $c'[r]$ is a frozen redex. Then in $c'[r]$ there is a free occurrence of a variable x that is bound in c_c. Case analysis on the position of this variable occurrence:
  - If x appears in r, then r is a frozen redex. Contradiction with the hypothesis that the redex r is reduced.
  - If x appears in c', then $r_c = c'[r']$ still contains x and r_c is a frozen redex. Contradiction with the hypothesis that r_c is a redex.

  Hence $c'[r]$ is not a frozen redex, and we can apply Root redex creation Lemma 4.11 to the step $c'[r] \to c'[r']$ to conclude.
- If $\mathtt{root}(r')$ is a prefix of $\mathtt{root}(r_c)$, then r_c is a subterm of $r' = u^{\{x := v\}}$, and there are three cases to consider:
  - If r_c is in a substituted occurrence of v, then its ancestor is already a redex. Contradiction with the hypothesis that the redex r_c is created by ρ.
  - If $\mathtt{root}(r_c)$ is in the x-skeleton of u but is not the position of an occurrence of x, then proposition (2) holds.
  - If $\mathtt{root}(r_c)$ is in u but not in the x-skeleton of u, then the ancestor of r_c is equal to r_c and is already a redex. Contradiction with the hypothesis that the redex r_c is created by ρ. □

4.4 Weak labelled λ-calculus

From now on we use the full signature $\mathcal{S}$.

The classification of redex creations 4.12 will guide the definition of the labels following a simple principle: the labels have to record all possible redex creations. More precisely, each redex r (and by extension each reduction step ρ) is given a *name* that contains all the information relevant to the creation of r. Thus we expect at least two things:

- If a redex r is created by a reduction step ρ, then the name of ρ should influence the name of r in some way.
- Once a redex is created, its name never changes.

These two properties are formalized in Direct contribution Lemma 4.21 and Residuals Lemma 4.22. These properties will allow us to use the labels to define Lévy-style redex families.

A label is either an atomic label (here a position for technical convenience) or a combination $[\Omega, \alpha]$ of a name Ω and a label α. A name is a sequence of labels obtained by collecting the individual labels of the meaningful positions of the corresponding redex. Most of these definitions are taken from the previous paper [Bal12c]; the obtained system is a slight simplification of a labelling presented by Blanc, Lévy and Maranget [BLM07].

Definition 4.13 (Labels)**.** *The set $\mathcal{L}$ of* **labels** *and the set $\mathcal{N}$ of* **redex names** *are defined by the following grammar:*

Labels $\mathcal{L}$	α	$::=$	$p \mid [\Omega, \alpha]$	*p position*
Names $\mathcal{N}$	Ω	$::=$	$\alpha_1 \dots \alpha_n$	$n \geq 2$

The labelled signature $\mathcal{S}^{\mathcal{L}}$ is defined as

$$\mathcal{S}^{\mathcal{L}} = \{f^{\alpha} \mid f \in \mathcal{S},\ \alpha \in \mathcal{L}\}$$

where arities are preserved. Let $(.)^{\bullet}$ be the forgetful morphism that maps terms over $\mathcal{S}^{\mathcal{L}}$ to terms over $\mathcal{S}$ by removing their labels. For any term t over $\mathcal{S}^{\mathcal{L}}$ and any position $p \in \mathtt{pos}(t)$, write $\tau_p(t)$ for the label of t at position p. In this section, call **initial** *a term over $\mathcal{S}^{\mathcal{L}}$ whose labels are all atomic (that is, a position) and different.*

The *contribution* relation is a straightforward containment relation on the labels. The strength of this simple syntactic notion is that it is equivalent to a more semantic notion of causal dependency between reduction steps, which we will prove in Section 4.5.

Definition 4.14 (Contribution)**.** *The* **direct contribution** *relation $\hookrightarrow_{dc}$ is defined on names by the following criterion: $\Omega \hookrightarrow_{dc} \Omega_1[\Omega, \alpha]\Omega_2$ for any name Ω, any possibly empty names Ω_1, Ω_2, and any label α. The* **contribution** *relation $\hookrightarrow_c$ is the transitive closure of $\hookrightarrow_{dc}$.*

Definition 4.15 (Name of a redex)**.** *A* **labelled redex** *is a term over $\mathcal{S}^{\mathcal{L}}$ of the form*

$$r = @^{\omega}(\varepsilon^{\alpha_1}(\dots \varepsilon^{\alpha_n}(\lambda^{\gamma} x.t)), u)$$

Its **name** *is the sequence*

$$\mathtt{name}(r) = \omega\alpha_1 \dots \alpha_n\gamma$$

Labelled β-reduction introduces the name of the reduced redex at all the places identified by Redex creation Theorem 4.12 as possibly related to a redex creation: the labels of the skeleton are modified, and a new label is created at the root of the reduced redex.

Definition 4.16 (Uniform relabelling)**.** *For any labelled n-ary context c and any label Ω, the* uniform relabelling *$[\Omega, c]$ of c by*

Ω *is defined by the following equations:*

$$\begin{aligned}
[\Omega, \Box] &= \Box \\
[\Omega, x] &= x \\
[\Omega, \varepsilon^{\alpha}(c)] &= \varepsilon^{[\Omega,\alpha]}(c) \\
[\Omega, \lambda^{\alpha} x.c] &= \lambda^{[\Omega,\alpha]} x.[\Omega, c] \\
[\Omega, @^{\alpha}(c_1, c_2)] &= @^{[\Omega,\alpha]}([\Omega, c_1], [\Omega, c_2]) \\
[\Omega, f^{\alpha}(c_1, \ldots, c_n)] &= f^{[\Omega,\alpha]}([\Omega, c_1], \ldots, [\Omega, c_n])
\end{aligned}$$

Definition 4.17 (Labelled β-reduction)**. Labelled β-reduction** *is defined by the following rule scheme:*

$$@^{\omega}(\varepsilon^{\alpha_1}(\ldots \varepsilon^{\alpha_n}(\lambda^{\gamma} x.s[Z_1, \ldots, Z_n])), Z) \underset{\beta}{\rightarrow} [\omega\alpha_1 \ldots \alpha_n\gamma, \varepsilon^{\epsilon}(s)]^{\{x := Z\}}[Z_1, \ldots, Z_n]$$

where s is a $\{x\}$-skeleton.

Example 4.18.

In this example we work on a labelling of the ordinary λ-term $((\lambda x.xa)(\lambda y.y))b$. Consider the labelled term

$$t \quad = \quad @^{\alpha}(@^{\beta}(\varepsilon^{\gamma}(\lambda^{\delta} x.@^{\eta}(\varepsilon^{\iota}(x), \varepsilon^{\nu}(a))), \lambda^{\mu} y.y), \varepsilon^{\kappa}(b))$$

The term t contains a labelled redex

$$r \quad = \quad @^{\beta}(\varepsilon^{\gamma}(\lambda^{\delta} x.@^{\eta}(\varepsilon^{\iota}(x), \varepsilon^{\nu}(a))), \lambda^{\mu} y.y)$$

with name $\Omega = \beta\gamma\delta$. The skeleton of the λ-abstraction of r is

$$\lambda^{\delta} x.@^{\eta}(\varepsilon^{\iota}(x), \Box)$$

This skeleton is the only part whose labels are modified by labelled β-reduction. Hence

$$t \quad \rightarrow \quad @^{\alpha}(\varepsilon^{[\beta\gamma\delta,\epsilon]}(@^{[\beta\gamma\delta,\eta]}(\varepsilon^{[\beta\gamma\delta,\iota]}(\lambda^{\mu} y.y), \varepsilon^{\nu}(a))), \varepsilon^{\kappa}(b))$$

In the target of this reduction step, a labelled redex

$$r_c \quad = \quad @^{[\beta\gamma\delta,\eta]}(\varepsilon^{[\beta\gamma\delta,\iota]}(\lambda^{\mu} y.y), \varepsilon^{\nu}(a))$$

with name $\Omega_c = [\beta\gamma\delta, \eta][\beta\gamma\delta, \iota]\mu$ such that $\Omega \hookrightarrow_{dc} \Omega_c$.

Finally, we define a notion of *parallel labelled reduction* that is meant to represent shared evaluation.

Definition 4.19 (Parallel reduction)**.** *Write $\bar{\rho} : t \Rightarrow t'$ if there exists a name Ω and a complete development $\varrho : t \twoheadrightarrow t'$ of all the labelled β-redexes in t of name Ω.*

4.5 Causality

In this section we show that the labels give a faithful account of causality in our system. Causal soundness Lemma 4.26 and Causal completeness Lemma 4.28 show that the set $\mathcal{N}$ of the names that contribute to a given redex name Ω is exactly the set of the names of the reduction steps that are needed to create a redex of name Ω from an initial term.

We formalize this fact using the following notion of *needed names*, which extends the notion of needed redexes.

Definition 4.20 (Needed name)**.** *A name Ω_1 is said to be* **needed** *for a name Ω_2, written $\Omega_1 \hookrightarrow_n \Omega_2$, if every reduction sequence $\varrho\rho_2$ with $\mathtt{src}(\varrho\rho_2)$ initial and $\mathtt{name}(\rho_2) = \Omega_2$ is such that ϱ contains a reduction step of name Ω_1.*

This subsection is devoted to the proof that the semantic notion of neededness $\hookrightarrow_n$ is equivalent to the syntactic notion of contribution $\hookrightarrow_c$ (Causality Theorem 4.29). We begin with three lemmas that are not new, but that are useful to derive stronger causality properties.

Lemma 4.21 (Direct contribution)**.** *For any reduction step $\rho : t \rightarrow t'$ of name Ω, if ρ_c is a reduction step from t' of name Ω_c that is created by ρ, then $\Omega \hookrightarrow_{dc} \Omega_c$.*

Proof. By a case analysis on the creation of ρ_c, guided by Redex creation Theorem 4.12. □

Lemma 4.22 (Residuals)**.** *For any reduction step $\rho : t \rightarrow t'$, if ρ_a is a reduction step from t of name Ω, then any descendant in t' of $\mathtt{root}(\rho_a)$ is the root of a redex with same name Ω.*

Proof. Similar to the proof given by Blanc, Lévy, and Maranget [BLM07]. The main point is that a non-frozen redex cannot be relabelled. □

Lemma 4.23 (Finite developments)**.** *Let t be a labelled λ-term and $\mathcal{R}$ a set of reduction steps from t. Then the three following hold:*

- *All the developments of $\mathcal{R}$ are finite.*
- *All the complete developments of $\mathcal{R}$ have the same target.*
- *All the complete developments of $\mathcal{R}$ define the same descendants and residuals.*

Proof. Deduced from finite developments for orthogonal TRS once the results from Section 5 are established. □

Causal soundness is the inclusion of $\hookrightarrow_c$ in $\hookrightarrow_n$. It is proved by tracking the origins of the labels and by checking that the labels do not record spurious contributions.

Lemma 4.24 (Reversed direct contribution)**.** *Let ϱ be a reduction sequence from an initial term. If the target of ϱ contains a label of the form $[\Omega, \alpha]$, then the sequence ϱ contains a reduction step of name Ω.*

Proof. By induction on the length of ϱ, remarking that, at each step, any label $[\Omega, \alpha]$ is created or descends from an identical label. □

Lemma 4.25 (Direct neededness)**.** *Let ϱ be a reduction sequence from an initial term. Let ρ be a reduction step from $t = \mathtt{tgt}(\varrho)$ and name Ω. If Ω' is a name such that $\Omega' \hookrightarrow_{dc} \Omega$, then ϱ contains a reduction step of name Ω'.*

Proof. Write $\Omega = \omega_1 \ldots \omega_n$. By definition there exists i such that $\omega_i = [\Omega', \alpha]$. By definition of the name of a redex, there exists a position $p \in \mathtt{pos}(t)$ such that $\tau_p(t) = [\Omega', \alpha]$ for some label α. Then by Lemma 4.24 the reduction sequence ϱ contains a reduction step of name Ω'. □

Lemma 4.26 (Causal soundness)**.** *Let Ω_1 and Ω_2 be two names such that $\Omega_1 \hookrightarrow_c \Omega_2$. Then $\Omega_1 \hookrightarrow_n \Omega_2$.*

Proof. By induction on the length of a shortest sequence $\Omega_1 \hookrightarrow_{dc} \ldots \hookrightarrow_{dc} \Omega_2$.

- If $\Omega_1 \hookrightarrow_{dc} \Omega_2$, then by Direct neededness Lemma 4.25 any reduction sequence from an initial term to a term containing of redex of name Ω_2 contains a reduction step of name Ω_1, which means that $\Omega_1 \hookrightarrow_n \Omega_2$.
- If $\Omega_1 \hookrightarrow_{dc} \Omega_3 \hookrightarrow_c \Omega_2$ and $\Omega_3 \hookrightarrow_n \Omega_2$, then similarly $\Omega_1 \hookrightarrow_n \Omega_3$ and by transitivity of neededness $\Omega_1 \hookrightarrow_n \Omega_2$. □

Causal completeness is the inclusion of $\hookrightarrow_n$ in $\hookrightarrow_c$. It is proved by checking that reduction steps whose names are not comparable can be permuted.

Lemma 4.27 (Permutation)**.** *If $t_0 \Rightarrow_{\Omega_0} t_1 \Rightarrow_{\Omega} t_2$ and $\Omega_0 \not\hookrightarrow_c \Omega$, then one of the following holds:*

- *$t_0 \Rightarrow_{\Omega} t_2$.*
- *There is a t_1' such that $t_0 \Rightarrow_{\Omega} t_1' \Rightarrow_{\Omega_0} t_2$.*

Proof. Since $\Omega_0 \not\hookrightarrow_c \Omega$, by contraposition of Direct contribution Lemma 4.21 the redexes of name Ω in t_1 are the residuals of redexes in t_0. In particular $t_0 \Rightarrow_{\Omega_0} t_1 \Rightarrow_{\Omega} t_2$ is a complete development of the set of all redexes in t_0 of name Ω_0 or Ω. By Finite developments Lemma 4.23 the term t_2 is the target of any complete development of these redexes, and in particular of the one first developing all the redexes of name Ω and then developing all the redexes of name Ω_0 if they have not been erased by the first step. □

Lemma 4.28 (Causal completeness)**.** *Let Ω_0 and Ω be two names such that $\Omega_0 \hookrightarrow_n \Omega$. Then $\Omega_0 \hookrightarrow_c \Omega$.*

Proof. Write $\bar{\mathcal{R}}$ for the set of all parallel labelled reduction sequences of minimal length from an initial term to a term containing redexes of name Ω. There is a n such that every sequence $\bar{\varrho} \in \bar{\mathcal{R}}$ has the form $\Rightarrow_{\Omega_1^\varrho} \ldots \Rightarrow_{\Omega_n^\varrho}$. Write $\bar{\mathcal{R}}^*$ for the set of all sequences $\bar{\varrho} \in \bar{\mathcal{R}}$ for which there is at least one $i \in \{1, \ldots, n\}$ such that $\Omega_i^\varrho \not\hookrightarrow_c \Omega$.

Suppose $\bar{\mathcal{R}}^*$ is not empty and let i be the greatest integer such that there is a sequence $\bar{\varrho} \in \bar{\mathcal{R}}^*$ satisfying $\Omega_i^\varrho \not\hookrightarrow_c \Omega$. Let $\bar{\varrho}$ be such a sequence in $\bar{\mathcal{R}}^*$ satisfying $\Omega_i^\varrho \not\hookrightarrow_c \Omega$. The sequence $\bar{\varrho}$ has the form $\Rightarrow_{\Omega_1^\varrho} \ldots \Rightarrow_{\Omega_i^\varrho} \Rightarrow_{\Omega_{i+1}^\varrho} \ldots \Rightarrow_{\Omega_n^\varrho}$.

By definition of $\bar{\varrho}$ we have $\Omega_{i+1}^\varrho \hookrightarrow_c \Omega$. Since $\Omega_i^\varrho \not\hookrightarrow_c \Omega$, by transitivity of $\hookrightarrow_c$ we get $\Omega_i^\varrho \not\hookrightarrow_c \Omega_{i+1}^\varrho$. Then by Permutation Lemma 4.27 we can build one of the following sequences:

- A sequence $\Rightarrow_{\Omega_1^\varrho} \ldots \Rightarrow_{\Omega_{i-1}^\varrho} \Rightarrow_{\Omega_{i+1}^\varrho} \ldots \Rightarrow_{\Omega_n^\varrho}$ strictly shorter than $\bar{\varrho}$, which contradicts the minimality of the length of $\bar{\varrho}$.
- A sequence $\Rightarrow_{\Omega_1^\varrho} \ldots \Rightarrow_{\Omega_{i+1}^\varrho} \Rightarrow_{\Omega_i^\varrho} \ldots \Rightarrow_{\Omega_n^\varrho}$ with $\Omega_i^\varrho \not\hookrightarrow_c \Omega$, of same length as $\bar{\rho}$ but which contradicts the maximality of i.

Hence the set $\bar{\mathcal{R}}^*$ must be empty, and every name Ω_0 that appears in a reduction sequence in $\bar{\mathcal{R}}$ satisfies $\Omega_0 \hookrightarrow_c \Omega$.

Finally, let Ω_0 be a name such that $\Omega_0 \hookrightarrow_n \Omega$. By definition 4.20, every reduction sequence ϱ from an initial term to a term that contains redexes of name Ω contains a step of name Ω_0. It is in particular the case of any reduction sequence developing a sequence in $\bar{\mathcal{R}}$. Hence $\Omega_0 \hookrightarrow_c \Omega$. □

Finally, Causal soundness Lemma 4.26 and Causal completeness Lemma 4.28 can be combined to prove that the syntactic contribution relation $\hookrightarrow_c$ and the semantic neededness relation $\hookrightarrow_n$ are equal.

Theorem 4.29 (Causality)**.** $\hookrightarrow_c = \hookrightarrow_n$.

4.6 The labelled weak λ-calculus as a DFS

Using the labels of Section 4.4 we get a straightforward definition for families.

Definition 4.30 (Families)**.** *The* **family relation** $\simeq$ *over reduction steps with history is defined by the following condition: $(\varrho_1, \rho_1) \simeq (\varrho_2, \rho_2)$ if and only if the reduction steps ρ_1 and ρ_2 have the same name. Hence families correspond to names, and we extend the* **contribution relation** $\hookrightarrow_c$ *to families.*

Notice that we do not use the histories ϱ_1 and ϱ_2 in the previous definition. Indeed, thanks to the causal labelling, the labels in the target of a reduction sequence ϱ already contains all the relevant pieces of information about ϱ.

Lemma 4.31. *The labelled weak λ-calculus equipped with its family relation is a Deterministic Family Structure.*

Proof. We first check that the labelled weak λ-calculus is a *DRS*:

[FD] Direct application of the Finite developments Lemma 4.23.

[Acyclicity] In the λ-calculus, if a redex r erases a redex r_e, then r_e is a strict subterm of r. The subterm relation being acyclic, there is no possible cross-erasure.

The axioms for DFS are then checked as follows:

[Initial] Let ρ_1 and ρ_2 be two reduction steps from a common initial term t. If ρ_1 and ρ_2 are different, in particular they apply to different positions and have different names.

[Copy] Suppose $(\varrho_1, \rho_1) \rhd (\varrho_2, \rho_2)$, then in particular $\rho_2 \in \rho_1/(\varrho_2/\varrho_1)$. By Residuals Lemma 4.22 the reduction steps ρ_2 and ρ_1 have the same name, hence (ϱ_1, ρ_1) and (ϱ_2, ρ_2) are in the same family.

[FFD] Orthogonal TRS satisfy the Finite Family Developments property [vO97]. Hence the weak λ-calculus does the same by Bisimulation Proposition 5.10.

[Creation] Direct application of Direct contribution Lemma 4.21.

[Contribution] This axiom can be reworded as $\hookrightarrow_c = \hookrightarrow_n$. Conclusion by Causality Theorem 4.29. □

Thus, by Glauert and Khasidashvili's Optimality Proposition 4.8, any complete needed weak family reduction sequence is optimal, which we call *weak optimality*. Such concrete reduction strategies will be detailed in Section 6.

5. Weak reduction and first-order rewriting

In this section we show that the weak λ-calculus behaves as an orthogonal first-order rewriting system[7]. First in Section 5.1 we recall the formalization of λ-lifting as a higher-order rewriting system given in the previous paper [Bal12c], which can be used to prove that the weak λ-calculus is strongly bisimilar to an orthogonal first-order term rewriting system; then in Section 5.2 we show that the transformation moreover preserves concepts and properties such as residuals or neededness.

5.1 Fully lazy λ-lifting

We formalize fully lazy λ-lifting as a higher-order rewriting system that replaces a skeleton by a single supercombinator symbol. The labelling of the transformation presented here is a slight simplification of the one of the previous paper [Bal12c], but it has roughly the same properties.

Definition 5.1 (Expansor)**.** *An* **expansor** *is a bijection $\psi : \mathcal{F} \to \langle\langle \Lambda \rangle\rangle$ that preserves arity. For any $f, g \in \mathcal{F}$ we say that g* **contains** *f and we write $g \supset f$ if f appears in the expansion $\psi(g)$. An expansor ψ is* **well-founded** *if there is no infinite sequence $f_1 \supset f_2 \supset \ldots$.*

For the rest of the paper, let ψ be a fixed well-founded expansor. Such a bijection between $\mathcal{F}$ and $\langle\langle \Lambda \rangle\rangle$ exists since for any n the set of n-ary contexts is countable.

For any λ-term t containing $\mathcal{F}$-symbols, the expansor ψ describes a structure that is implicit in t, in some sense hidden in the $\mathcal{F}$-symbols. The notion of *extended position* in a λ-term t gives a partial account of this structure by indicating ordinary positions of t as well as "internal" positions of the $\mathcal{F}$-symbols.

Definition 5.2 (Extended positions)**.** *The set of the* **ψ-extended positions** *of a λ-term, or simply* **ψ-positions** *of t, written $\mathtt{pos}_\psi(t)$, is defined as follows:*

$$\begin{aligned}
\mathtt{pos}_\psi(x) &= \{\epsilon\} \\
\mathtt{pos}_\psi(\lambda x.t) &= \{\epsilon\} \cup 1 \cdot \mathtt{pos}_\psi(t) \\
\mathtt{pos}_\psi(t_1 t_2) &= \{\epsilon\} \cup 1 \cdot \mathtt{pos}_\psi(t_1) \cup 2 \cdot \mathtt{pos}_\psi(t_2) \\
\mathtt{pos}_\psi(f(t_1, ..., t_n)) &= \{\epsilon\} \cup (0 \cdot \mathtt{pos}_\psi(\psi(f))) \cup (\textstyle\bigcup_i i \cdot \mathtt{pos}_\psi(t_i))
\end{aligned}$$

[7] A rewriting system is **orthogonal** when its rewriting rules cannot interfere with each other.

This set if finite since ψ is well-founded.

Example 5.3.

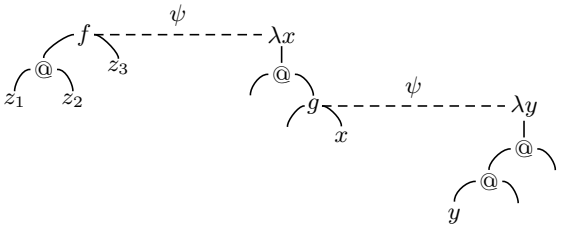
Consider the λ-term

$$t \quad = \quad f(@(z_1, z_2), z_3)$$

and suppose the following expansions (with f and g two binary symbols):

$$\begin{array}{rcl} \psi(f) & = & \lambda x.\Box g(\Box, x) \\ \psi(g) & = & \lambda y.y\Box\Box \end{array}$$

The following picture shows t and the expansions of the symbols f and g.



The set of positions of t is $\mathtt{pos}(t) = \{\epsilon, 1, 11, 12, 2\}$. The set of ψ-positions of t is:

$$\mathtt{pos}_\psi(t) = \{\epsilon, 1, 11, 12, 2, 0, 01, 012, 0122, 0120, 01201, 012011, 0120111\}$$

The following table gives a correspondence between some ψ-positions of t and symbols from t, $\psi(f)$, or $\psi(g)$.

ϵ	f
0	λx
012	g
0120	λy

With the next two definitions we propose an initial labelling that takes into account the expansions of the term. Intuitively, each node is labelled by the position it would have if the term were fully expansed.

Definition 5.4 (Expansed positions)**.** *Let c be a n-ary context and $p \in \mathtt{pos}(c)$ a position of c. The* **expansion** *of p relative to c is the position $(p)^{c,\psi}$ defined by:*

$$\begin{array}{rcll} (\epsilon)^{c,\psi} & = & \epsilon & \\ (1 \cdot p)^{\varepsilon(c),\psi} & = & 1 \cdot (p)^{c,\psi} & \\ (1 \cdot p)^{\lambda x.c,\psi} & = & 1 \cdot (p)^{c,\psi} & \\ (i \cdot p)^{c_1 c_2,\psi} & = & i \cdot (p)^{c_i,\psi} & i \in \{1, 2\} \\ (i \cdot p)^{f(c_1,\ldots,c_n),\psi} & = & (q_i)^{\psi(f),\psi} \cdot (p)^{c_i,\psi} & i \in \{1, \ldots, n\} \\ & & & q_i \text{ position of } i^{th} \text{ hole in } \psi(f) \end{array}$$

Definition 5.5 (Extended initial labelling)**.** *For any position p, the function $i_p(.)$ maps λ-terms to labelled λ-terms through the following rules:*

$$\begin{array}{rcll} i_p(x) & = & x & \\ i_p(\varepsilon(t)) & = & \varepsilon^p(i_{p\cdot 1}(t)) & \\ i_p(\lambda x.t) & = & \lambda^p x.i_{p\cdot 1}(t) & \\ i_p(t_1 t_2) & = & @^p(i_{p\cdot 1}(t_1), i_{p\cdot 2}(t_2)) & \\ i_p(f(t_1, \ldots, t_n)) & = & f^p(i_{p\cdot q_1}(t_1), \ldots, i_{p\cdot q_n}(t_n)) & \\ & & \forall j.\ q_j = (j)^{f(t_1,\ldots,t_n),\psi} & \end{array}$$

Let t be a λ-term. The **initial labelling** *of t is the labelled λ-term $i_\epsilon(t)$, also written $i(t)$.*

Example 5.6.

Consider the λ-term $t = f(@(z_1, z_2), z_3)$ from example 5.3. Then

$$i(t) \quad = \quad f^\epsilon(@^{11}(z_1, z_2), z_3)$$

With this extended initial labelling, we can now define the *target system*, that is the first-order rewriting system to which the weak λ-calculus is bisimilar.

Definition 5.7 (Target redex)**.** *A* **target redex** *is a term over $\mathcal{S}^{\mathcal{L}}$ of the form*

$$@^\omega(\varepsilon^{\alpha_1}(\ldots\varepsilon^{\alpha_n}(f^\gamma(t_1, \ldots, t_n))), u)$$

Its **name** *is the sequence*

$$\mathtt{name}(r) = \omega\alpha_1 \ldots \alpha_n\gamma$$

Definition 5.8 (Source & Target reductions)**.** *The* **source reduction** *is the labelled β-reduction of definition 4.17. The* **target reduction** *is given by the following rule scheme:*

$$\begin{array}{c} @^\omega(\varepsilon^{\alpha_1}(\ldots\varepsilon^{\alpha_n}(f^\gamma(Z_1, \ldots, Z_n))), Z) \\ \rightarrow_f \\ [\omega\alpha_1 \ldots \alpha_n\gamma,\ [\Omega_k, \ldots [\Omega_1, \varepsilon^\epsilon(i_p(s))]]\ ]^{\{x\,:=\,Z\}}[Z_1, \ldots, Z_n] \end{array}$$

where $\gamma = [\Omega_k, \ldots [\Omega_1, p]]$ and $\psi(f) = \lambda x.s$.

Definition 5.9 (Lambda-lifting)**.** *The λ-lifting rewriting system is defined by the following rules applied in any context (no restriction to weak reduction here).*

- *For any label α, variable x, and $\{x\}$-skeleton s, the pair $\lambda^\alpha x.s[Z_1, ..., Z_n] \quad \rightarrow_c \quad (\psi^{-1}(\lambda x.s^\bullet))^\alpha(Z_1, ..., Z_n)$ is a* **λ-lifting rule of rank** 0.
- *A pair $f(Z_1, \ldots, Z_n) \rightarrow f'(Z_1, \ldots, Z_n)$ is called a* **λ-lifting rule of rank** $n+1$ *if there is a reduction step $\psi(f) \rightarrow \psi(f')$ by a λ-lifting rule of rank n.*

Proposition 5.10 (Bisimulation, Balabonski [Bal12c])**.** *The λ-lifting relation $\twoheadrightarrow_{\mathtt{lft}}$ is a strong bisimulation in the global system.*

It is also known that the λ-lifting reduction relation is convergent, and compatible with sharing. In the next section we add new results that ensure that $\twoheadrightarrow_{\mathtt{lft}}$ preserves the properties that are useful to the study of optimality.

5.2 Properties preserved by the reduction to first order

For any term t, write $\mathtt{lift}(t)$ for the normal form of t by λ-lifting. Similarly, for any position p of a term t, write $\mathtt{lift}(p)$ for the set of its descendants along any complete lifting of t, and for any source reduction step ρ from t, write $\mathtt{lift}(\rho)$ for the residual of ρ after any complete lifting of t.

In this section we show that $\mathtt{lift}(.)$ preserves descendants, residuals, and neededness.

Lemma 5.11 (Labels and descendants)**.** *Let t be a labelled term whose labels are initial and different, and $t \twoheadrightarrow t'$ be a reduction sequence. Then a position $p' \in \mathtt{pos}(t')$ is a descendant of a position $p \in \mathtt{pos}(t)$ if and only if there are names $\Omega_1, \ldots, \Omega_n$ such that $\tau_{t'}(p') = [\Omega_n, \ldots [\Omega_1, \tau_t(p)]]$.*

Proof. By induction on the length of the sequence, with the following generalized statement:

For any labelled term t, reduction sequence $\varrho : t \rightarrow t'$, and position p_0, write

$$\begin{array}{l} P_\alpha = \{p \in \mathtt{pos}(t) \mid \exists k,\ \exists\Omega_k \ldots \Omega_1,\ \tau_p(t) = [\Omega_k, \ldots [\Omega_1, p_0]]\} \\ P'_\alpha = \{p' \in \mathtt{pos}(t') \mid \exists k',\ \exists\Omega_{k'} \ldots \Omega_1,\ \tau_p(t) = [\Omega_{k'}, \ldots [\Omega_1, p_0]]\} \end{array}$$

for the sets of the positions of t and t' where p_0 appears. Then

$$P'_\alpha = P_\alpha/\varrho$$

□

With this characterization of descendants, we can use confluence results on a labelled system to get results about descendants and residuals.

Lemma 5.12 (Preservation of descendants)**.** *Let t be a λ-term, $p \in \mathtt{pos}(t)$ a position, and $\rho : t \to t'$ a source reduction step. Then $\mathtt{lift}(p/\rho) = \mathtt{lift}(p)/\mathtt{lift}(\rho)$.*

Proof. By Bisimulation Proposition 5.10 we have the following diagram:

$$\begin{array}{ccc} t & \xrightarrow{\varrho_{lft}} & \mathtt{lift}(t) \\ \downarrow{\scriptstyle \rho} & & \downarrow{\scriptstyle \mathtt{lift}(\rho)} \\ t' & \xrightarrow{\varrho'_{lft}} & \mathtt{lift}(t') \end{array}$$

By definition of descendants along a reduction sequence we have $\mathtt{lift}(p/\rho) = (p/\rho)/\varrho'_{lft} = p/(\rho\varrho'_{lft})$ as well as $p/(\varrho_{lft}\mathtt{lift}(\rho)) = (p/\varrho_{lft})/\mathtt{lift}(\rho) = \mathtt{lift}(p)/\mathtt{lift}(\rho)$. Since the sequences $\rho\varrho'_{lft}$ and $\varrho_{lft}\mathtt{lift}(\rho)$ have same target, by Labels and descendants Lemma 5.11 we also have $p/(\rho\varrho'_{lft}) = p/(\varrho_{lft}\mathtt{lift}(\rho))$. By combining these equations we get $\mathtt{lift}(p/\rho) = \mathtt{lift}(p)/\mathtt{lift}(\rho)$. □

The preservation of residuals is an immediate corollary of the preservation of descendants.

Lemma 5.13 (Preservation of residuals)**.** *Let $\rho : t \to t'$, $\rho_a : t \to u$ and $\rho_r : t' \to u'$ be three source reduction steps. Then ρ_r is a residual of ρ_a after ρ if and only if $\mathtt{lift}(\rho_r)$ is a residual of $\mathtt{lift}(\rho_a)$ after $\mathtt{lift}(\rho)$.*

Finally, we deduce that λ-lifting also preserves the notion of neededness, which will allow us in the next section to transfer a result on needed strategies from first-order rewriting to the weak λ-calculus.

Theorem 5.14 (Preservation of neededness)**.** *Let t be a normalizing term. Then a source reduction step ρ from t is needed if and only if the target reduction step $\mathtt{lift}(\rho)$ is needed in the target system.*

Proof. Let ρ be a needed source reduction step from t. Let $\varrho^f = \rho_1^f \dots \rho_n^f$ be a normalizing target reduction sequence from $\mathtt{lift}(t)$. By Bisimulation Proposition 5.10 we have a normalizing source reduction sequence $\varrho = \rho_1 \dots \rho_n$ from t. By hypothesis ρ is needed, hence there exists i such that ρ_i is a residual of ρ. By Preservation of residuals Lemma 5.13, the reduction step ρ_i^f is a residual of $\mathtt{lift}(\rho)$. Hence $\mathtt{lift}(\rho)$ is needed.

We show similarly that if $\mathtt{lift}(\rho)$ is needed, then ρ is also needed. □

The properties stated in this section show that the weak λ-calculus, while it has the appearance of the λ-calculus, has the properties of a first-order rewriting system. This first-order essence of weak reduction will become even more apparent in the next section.

6. Two weakly optimal strategies

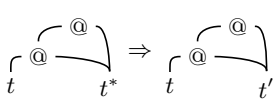
We conclude our investigation by identifying two strategies that realize the weak optimality characterized in Section 4. The first strategy (Section 6.1) is *call-by-need*: Wadsworth's well-known technique for efficient evaluation of the λ-calculus [Wad71], which is also the source of mainstream lazy evaluation mechanisms. The second strategy (Section 6.2) is a bit more intriguing, since it reaches optimality without using any sharing (but of course, following Uncomputability Theorem 3.14 the latter cannot be computable).

6.1 Call-by-need

As shown in the previous paper [Bal12c], the labelled weak λ-calculus defined in Section 4 describes a graph-based evaluation mechanism for the λ-calculus, which is exactly the one proposed by Wadsworth [Wad71]. This is done through the technique of sharing-via-labelling [Mar91, Bal12a], that has first been introduced by Maranget.

The main idea is as follows: if two subterms t^δ have the same label δ, then they represent the same node in the corresponding graph (this node is marked with a star in the picture below), and have to be reduced in only one step.

$$\begin{array}{l} @^\alpha(@^\beta(t^\gamma, \underline{t^\delta}), \underline{t^\delta}) \\ \Rightarrow @^\alpha(@^\beta(t^\gamma, t'^\zeta), t'^\zeta) \end{array}$$



This principle allows us to interpret labelled terms as graphs, provided a consistency property of the labels is satisfied: two subterms that have the same label must be equal.

Finally, the following proposition ensures that parallel reduction preserves the consistency of labels, and thus describes a graph rewriting system. Moreover, this graph rewriting system is known to correspond to Wadsworth's original technique [Bal12c].

Proposition 6.1 (Sharing-via-labelling, Balabonski [Bal12c])**.** *The following property is preserved by parallel reduction:*

Sharing *If $\tau_{p_1}(t) = \tau_{p_2}(t)$ then $t|_{p_1} = t|_{p_2}$.*

Hence Wadsworth's call-by-need implements complete family reduction. Moreover, call-by-need is known to reduce only needed redexes [BKKS87], and thus Glauert and Khasidashvili's Optimality Proposition 4.8 applies.

Theorem 6.2 (Optimality)**.** *Wadsworth's call-by-need is a weakly optimal reduction strategy.*

Besides marking a well-known reduction strategy as weakly optimal, this optimality theorem reveals a difference between weak and non-weak reduction: while any implementation of non-weak optimality requires a sharing of contexts (see example 4.2), the sharing of subterms expressible by sharing-via-labelling is enough for weak optimality.

6.2 Innermost needed reduction

Now we are going to combine the results of the previous sections to prove that the shortest weak reduction sequences without sharing have the same length as the complete needed family reduction sequences. First we transfer a sufficient condition for a reduction strategy to be optimal from the first-order rewriting literature to our system, then we check that this condition describes a subset of complete needed family reduction.

Proposition 6.3 describes a sufficient condition for a reduction strategy to be optimal in orthogonal term rewriting systems, that has been established by Khasidashvili. Using our formalization of λ-lifting, we transfer this result to the weak λ-calculus.

Proposition 6.3 (Khasidashvili's optimality criterion [Kha93])**.** *Let t be a term in an orthogonal* TRS*, and ϱ a normalizing reduction sequence of source t in which each step reduces an innermost needed redex. Then no normalizing reduction sequence of source t is shorter than ϱ.*

Theorem 6.4 (Optimality criterion)**.** *Let t be a λ-term, and ϱ a normalizing weak reduction sequence of source t in which each step reduces an innermost needed redex. Then no normalizing weak reduction sequence of source t is shorter than ϱ.*

Proof. Suppose there is a normalizing weak reduction sequence ϱ' of source t that is shorter than ϱ. Let ϱ_f (resp. ϱ'_f) be the image of ϱ (resp. ϱ') in the target system. By Bisimulation Proposition 5.10, ϱ_f and ϱ'_f have the same source, are both normalizing, and ϱ'_f is shorter than ϱ_f. Moreover, by Preservation of neededness Theorem 5.14, each reduction step in the sequence ϱ_f reduces an innermost needed redex. Hence by Khasidashvili's Optimality Proposition 6.3 the reduction sequence ϱ_f is of minimal length, which contradicts the existence of the sequences ϱ'_f and ϱ'. □

This criterion shows once again that weak reduction is closer to first-order rewriting than to the usual non-weak λ-calculus. Indeed, Guerrini proved with a counter-example that innermost needed reduction is not optimal in the non-weak λ-calculus [Gue96].

Lemma 6.5. *An innermost needed reduction step does not duplicate any needed redex.*

Proof. Deduced from Khasidashvili's results [Kha93] using λ-lifting. □

Thus an innermost needed reduction sequence performs only needed reduction steps on redexes that have not been duplicated, and such a sequence is an instance of complete needed family reduction.

Theorem 6.6 (Optimality)**.** *Innermost needed reduction is a weakly optimal reduction strategy.*

This optimality theorem gives away another important difference between weak and non-weak reduction. Indeed, Lamping exhibited an example for which non-weak Lévy-optimality cannot be realized without sharing [Lam90].

7. Conclusion

We characterized optimal evaluation in the weak λ-calculus and identified two interesting weakly optimal strategies: the well-known call-by-need strategy, and innermost needed reduction. The latter illustrates the fact that the weak λ-calculus has the properties of a first-order rewriting system, and also shows that weakly optimal evaluation does not require sharing. However, we also showed that the optimal strategies without sharing cannot be computable. Finally, in the setting of weak reduction the meaning of sharing is not to make the shortest path shorter, but rather to make it effectively reachable.

Acknowledgments

Thanks to Delia Kesner, Olivier Danvy and Vincent van Oostrom, who offered me the tools to explore this topic. Thanks also to François Pottier, Mike Rainey, and the ICFP reviewers for their helpful comments and suggestions.

References

[Bal12a] Thibaut Balabonski. Axiomatic sharing-via-labelling. In Ashish Tiwari, editor, *RTA*, volume 15 of *LIPIcs*, pages 85–100. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[Bal12b] Thibaut Balabonski. *La pleine paresse, une certaine optimalité*. Ph.D. thesis, Université Paris 7 – Diderot, 2012.

[Bal12c] Thibaut Balabonski. A unified approach to fully lazy sharing. In John Field and Michael Hicks, editors, *POPL*, pages 469–480. ACM, 2012.

[Bar84] Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, Amsterdam, revised edition edition, 1984.

[BBKV76] Hendrik Pieter Barendregt, Jan A. Bergstra, Jan Willem Klop, and Henri Volken. Some notes on lambda reduction. Technical Report 22, University of Utrecht, Dpt. of mathematics, 1976.

[BKKS87] Hendrik Pieter Barendregt, Richard Kennaway, Jan Willem Klop, and M. Ronan Sleep. Needed reduction and spine strategies for the lambda calculus. *Information and Computation*, 75(3):191–231, 1987.

[BLM07] Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. Sharing in the weak lambda-calculus revisited. In Erik Barendsen, Herman Geuvers, Venanzio Capretta, and Milad Niqui, editors, *Reflections on Type Theory, Lambda Calculus, and the Mind*, pages 41–50. ICIS, Faculty of Science, Radbout University Nijmegen, 2007. Essays Dedicated to Henk Barendregt on the Occasion of his 60th Birthday.

[cH98] Naim Çagman and J. Roger Hindley. Combinatory weak reduction in lambda calculus. *Theoretical Computer Science*, 198(1-2):239–247, 1998.

[DS00] Olivier Danvy and Ulrik Pagh Schultz. Lambda-dropping: transforming recursive equations into programs with block structure. *Theoretical Computer Science*, 248(1-2):243–287, 2000.

[GK96] John R. W. Glauert and Zurab Khasidashvili. Relative normalization in deterministic residual structures. In Hélène Kirchner, editor, *CAAP*, volume 1059 of *Lecture Notes in Computer Science*, pages 180–195. Springer, 1996.

[Gue96] Stefano Guerrini. *Theoretical and Practical Issues of Optimal Implementations of Functional Languages*. Ph.D. thesis, Università di Pisa, 1996.

[How70] William Alvin Howard. Assignment of ordinals to terms for primitive recursive functionals of finite type. In *Intuitionism and Proof Theory*, pages 443–458, 1970.

[Kha93] Zurab Khasidashvili. Optimal normalization in orthogonal term rewriting systems. In Claude Kirchner, editor, *RTA*, volume 690 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 1993.

[KvOvR93] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, 121(1&2):279–308, 1993.

[Lam90] John Lamping. An algorithm for optimal lambda calculus reduction. In Frances E. Allen, editor, *POPL*, pages 16–30. ACM Press, 1990.

[Lé78] Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul*. Ph.D. thesis, Université Paris VII, 1978.

[Lé80] Jean-Jacques Lévy. Optimal reductions in the lambda-calculus. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalisms*, pages 159–191, 1980.

[Mar91] Luc Maranget. Optimal derivations in weak lambda-calculi and in orthogonal terms rewriting systems. In David S. Wise, editor, *POPL*, pages 255–269. ACM Press, 1991.

[PJ87] Simon L. Peyton-Jones. *The Implementation of Functional Programming Languages*. Prentice-Hall, 1987.

[Sco68] Dana Scott. A system of functional abstraction, 1968. Lectures delivered at University of California, Berkeley, Cal., 1962/63.

[Ter03] Terese. *Term Rewriting Seminar – Terese*. Vol. 55 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003.

[vO97] Vincent van Oostrom. Finite family developments. In Hubert Comon, editor, *RTA*, volume 1232 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 1997.

[Wad71] Christopher P. Wadsworth. *Semantics and Pragmatics of the Lambda Calculus*. Ph.D. thesis, Oxford, 1971.