A Unified Approach to Fully Lazy Sharing

Thibaut Balabonski

Univ Paris Diderot, Sorbonne Paris Cité,
PPS, UMR 7126, CNRS,
F-75205 Paris, France
thibaut.balabonski@pps.jussieu.fr

Abstract

We give an axiomatic presentation of sharing-via-labelling for weak λ-calculi, that makes it possible to formally compare many different approaches to fully lazy sharing, and obtain two important results. We prove that the known implementations of full laziness are all equivalent in terms of the number of β-reductions performed, although they behave differently regarding the duplication of terms. We establish a link between the optimality theories of weak λ-calculi and first-order rewriting systems by expressing fully lazy λ-lifting in our framework, thus emphasizing the first-order essence of weak reduction.

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

General Terms Theory, Languages

Keywords Sharing, Full laziness, Lambda-lifting, Rewriting, Lambda-calculus, Weak reduction, Optimality, Labelling.

1. Introduction

In the implementation of functional programming languages, a fundamental problem is the efficient evaluation of β-reduction. This problem has been studied for a long time. Its difficulty comes from the fact that one has to minimize the number of β-steps as well as control the actual (amortized) cost of single β-reduction steps. The minimization of the number of β-steps requires, in turn, to handle two different issues: avoiding non-needed computations, and minimizing duplications of unfinished work.

In λ-calculus, some reduction strategies [BKKS87] can completely avoid non-needed computations. However, it is also known that no reduction strategy can completely avoid duplications [Lam90]. Hence, in any case, one has to cope with duplications that still occur, and find some appropriate ways to deal with them.

This is exactly the point of sharing: building implementations in which the duplicated occurrences of a given original subterm keep a unique shared representation. This allows one to evaluate all the copies simultaneously, as if they were only one. The idea is to make sure that some parts of a program which are *logically* duplicated (in the term representation of the program) remain *physically* single pieces (in the memory of the evaluator).

POPL'12, January 25–27, 2012, Philadelphia, PA, USA.

Sharing cannot be achieved using only λ-terms: it requires the use of other technical tools, for instance graphs, closures, or program transformations. The various resulting formalisms may be hardly comparable.

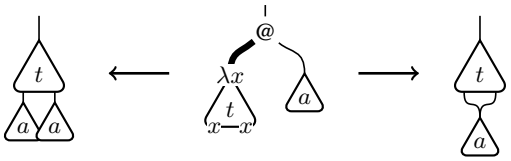
This paper focuses on one particularly rich flavor of sharing called *fully lazy sharing* (described in section 1.2) and aims at formally relating its various implementations. This is done by defining a framework called *sharing-via-labelling systems* in which they can all be expressed and compared.

This unified approach provides compiler writers with increased knowledge on the wide panorama of full laziness. In particular, it replaces a series of sometimes informal justifications of equivalence by a central theorem ensuring that all the considered approaches are equivalent with respect to the number of shared β-steps. As a consequence, one can safely restrict any subsequent comparison of two fully lazy models to other parameters of interest such as their space consumption or the actual cost of maintaining sharing. Having a unified framework will also simplify the task of comparing full laziness to the other efficient implementation techniques, such as other degrees of laziness (up to optimality), or partial evaluation.

The rest of the introduction is organized as follows: Section 1.1 presents the various technical tools commonly used to specify sharing, Section 1.2 describes how these tools have been used and combined over the past 40 years to propose different definitions of fully lazy sharing, and Section 1.3 details our approach and the contributions of the paper.

1.1 Many tools for sharing

Graphs. The most intuitive way of expressing sharing might be by using graphs. In the pictures, the binary node @ represents application, and redexes[1] are marked with bold lines. For instance in the center of the following picture, an abstraction $\lambda x.t$ is applied to an argument a. The function body t contains two occurrences of the variable x: the argument a is thus logically duplicated (on the left).

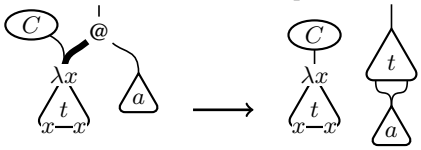


The simplest notion of sharing, which may be referred to as *lazy sharing*, or just *laziness*, prevents the previous duplication by keeping a physically unique, with two pointers to its location (right hand side part of the previous picture).

Here the term *laziness* is to be taken in literal sense: the duplications are postponed as long as it is possible, but some of them will

[1] a **red***ucible* **ex***pression*, or **redex**, designs a place where an evaluation step can take place

eventually happen. For instance, a shared function has to be copied prior to any instantiation, as shown in the picture below.



Extraction of free parameters. The previous pictures feature graphs built with λ-calculus constructs, and in particular with binders. This requires either to define variable renaming (α-conversion) or to add some special structure to represent binding. In any case the resulting graph formalism is quite complex. Graph reduction can be made easier by turning λ-terms (that are higher-order terms) to applicative expressions (that are first-order terms). This is the point when compiling the λ-calculus into combinators [Tur79] or supercombinators [Hug82], techniques that finally led to the λ-lifting program transformation [Joh85, Jon87].

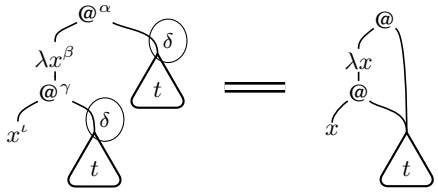
This transformation extracts all the free variables from a function and replaces what remains of the function by a symbol called supercombinator. New reduction rules are added to deal with the new symbols.

Closures and memory heaps. While graphs are a simple and old way to describe lazy sharing, the reference system for the semantics of lazy evaluation is J. Launchbury's natural semantics [Lau93]. It introduces *let ... in ...* constructs to name the arguments of the applications, and then puts these arguments in a heap. Sharing then appears as memoization: when one needs to access the content of a variable, the corresponding expression found in the heap is evaluated, and the heap is updated with the obtained result.

In contrast to the previous big-step approach, a small-step description of lazy evaluation based on terms is the call-by-need λ-calculus of Z. Ariola et al. [AFM$^+$95]. Here again, sharing is expressed thanks to additional *let ... in ...* constructs used as closures. Similar effects can also be achieved by expressing sharing using explicit substitutions [Yos94]. The big-step and small-step styles can be related by well-known transformations [DMMZ10].

Labels and weak reduction. Finally, laziness is seen in [Yos94, Mar91] as the optimal way of sharing in weak λ-calculi: variants of the λ-calculus where reduction under λ-abstractions is restricted. First, J.-J. Lévy [Lé80] described optimal sharing for the plain λ-calculus (unrestricted, untyped, usual λ-calculus) by means of labelled λ-terms. Then L. Maranget [Mar91] adapted these ideas for a weak λ-calculus and for first-order rewriting and got an additional result that is not valid for the plain λ-calculus: labelled terms represent graphs implementing optimal sharing.

In [Mar91] the link between labelled terms and graphs is made by interpreting the label of a term as its location in memory, or graphically by its coordinate:



We call this principle *sharing-via-labelling*. The idea is also explored in [DLLL05]. In this setting the equality of labels corresponds to the physical equality of two terms, which should in turn imply their syntactic equality: two terms stored/drawn at the same place ought to be equal. The reduction of a graph-redex is simulated by the reduction of all the labelled term-redexes with a given label. One then needs to ensure that the *sharing property* (terms with equal labels are syntactically equal) is preserved by reduction.

1.2 Full laziness: State of the art

The main idea. Full laziness is based upon the following remark: the *constant parts* of a function body are not affected by the instantiation of the function, hence they need not be duplicated.

This can be formalized by means of the notion of free expression. We recall the definition given in [Jon87]. Say a subterm s of t is **free** in $\lambda x.t$ if all the free variables of s are free in $\lambda x.t$. A **maximal free expression** of $\lambda x.t$ is a free expression of $\lambda x.t$ which is not contained into any other free expression of $\lambda x.t$.

Fully lazy sharings. The various definitions of fully lazy sharing come from a combination of the previous idea with one or more of the technical tools described in section 1.1.

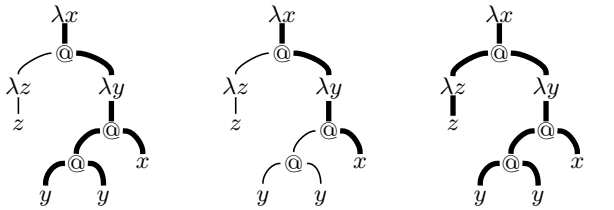
The first description of fully lazy sharing is in the *graph* evaluation technique presented by C.P. Wadsworth [Wad71]. This graph reduction performs only a partial copy of a duplicated function body, by avoiding the copy of its maximal free expressions (see Example 1a). O. Shivers and M. Wand [SW05] enrich the graph structure of [Wad71] to allow a simple and efficient implementation. For this they also use a different characterization of what has to be copied, which we detail in Section 3.1.

Two other approaches combine *graphs* with other tools. First S. Peyton-Jones [Jon87] reaches a simple graph formalism thanks to a fully-lazy version of λ-lifting. Following [Hug82], fully-lazy λ-lifting replaces the extraction of the free variables of a function by the *extraction* of its maximal free expressions. Second, T. Blanc, J.-J. Lévy and L. Maranget [BLM07] derive a graph implementation of fully lazy sharing through the sharing-via-labelling principle, using *labels* that characterize optimal sharing for a weak λ-calculus studied in [ÇH98]. This approach can copy *fewer* graph nodes of the duplicated abstractions (see Example 1b).

Finally, Z. Ariola and M. Felleisen [AF97] and P. Sestoft [Ses97] use the *extraction* of maximal free expressions to build fully lazy versions of (respectively) the call-by-need λ-calculus [AFM$^+$95] and Launchbury's natural semantics for laziness [Lau93]. Both solutions are based on *closures* represented by *let ... in ...* constructs. The former solution [AF97] uses a more restrictive definition of free expressions and hence may in some cases copy *more* nodes than the others (see Example 1c).

Example 1.

Bold lines identify the parts of the function that are duplicated by the different models. See Section 3.1 for a formal statement.



(a) [Wad71] (b) [BLM07] (c) [AF97]

Summary. The following table sums up how each of the previous works gives its own view on fully lazy sharing, with different interpretations of the same main idea and using various combinations of technical tools that are sometimes hardly comparable. We use the symbol = to mean "as many copied nodes as [Wad71]".

		Tools			
	Dupl.	Graphs	Extraction	Closures	Labels
[Wad71]	=	✓			
[Jon87]	=	✓	✓		
[AF97]	More		✓	✓	
[Ses97]	=		✓	✓	
[SW05]	=	✓			
[BLM07]	Fewer	✓			✓

This paper proposes a formal setting in which all the approaches mentioned above can be expressed. This allows us to formally compare them and leads us to the two following conclusions:

- The previous approaches correspond to at least three different graph implementations. This means that, strictly speaking, they do not all induce the same amount of sharing. Hence, despite the fact that all these approaches intend to implement the same idea their equivalence is not obvious.
- However, all these approaches have the same reduction space. This means that the different implementations of fully lazy sharing perform the same number of β-reductions. In other words, any further comparison of these approaches need not anymore take this parameter into account.

1.3 Content of the paper

An axiomatic framework for sharing-via-labelling. We build in Section 2 an axiomatic framework which generalizes the work of T. Blanc, J.-J. Lévy and L. Maranget [BLM07] and allows us to express all the previous approaches. We use labelled terms to describe the graphs realizing optimal sharing for a given notion of weak reduction. Various weak reduction notions are defined thanks to an axiomatic description of the parts of the program where reduction is forbidden. In any case the restrictions concern only evaluation in the body of a non-instantiated function, called partial evaluation. This implies that the call-by-value and call-by-name strategies are always valid. However, the different weak calculi may or may not be confluent (see [ÇH98] and Example 3).

This approach of sharing-via-labelling allows us in Section 3 to relate all the definitions of fully lazy sharing that do not rely on supercombinators and λ-lifting. In other words this axiomatic framework, which is designed in higher-order rewriting, covers the definitions of full laziness which directly operate in the higher-order world [Wad71, AF97, Ses97, SW05, BLM07]. The remaining approaches using a translation to first-order rewriting by λ-lifting [Hug82, Jon87] are studied separately in Section 4. The translation to first-order by means of combinators of D. Turner [Tur79] is out of the scope of the present paper, since these combinators simulate explicit substitutions and then introduce additional reduction steps.

Notably due to its axiomatic nature, our framework is not suitable for an immediate implementation. On the other hand, this approach teaches us something about full laziness in general and on its various concrete implementations. The novelty of our framework lies in the fact that it cannot be seen as a straightforward generalization of any of the aforementioned embodiments of full laziness taken in isolation: the axiomatization rather comes from an analysis of the similarities and the differences of all the concrete systems. This yields a new system whose specific properties may be understood as the *intersection* of the particular properties of the various concrete systems. In other words, our axiomatization tries to grasp the essence of full laziness.

A formal coding of higher order into first order by λ-lifting. The λ-lifting program transformation turns a λ-term into a first-order term. The main feature of λ-lifting is the transformation of λ-abstractions into function symbols, also called supercombinators, over which first-order reduction rules are defined. As emphasized in [LM09], this transformation unveils a tight relation between weak λ-calculus and first-order rewriting.

Usual definitions of λ-liftings [Joh85] proceed by first defining the transformation of λ-abstractions, and then iteratively applying the process to a λ-term until it contains no more λ-abstractions. Definitions differ in particular in the way in which a single λ-abstraction is transformed and on the order in which the iteration is applied. For instance, [Jon87] describes a bottom-up transformation, while [Ses97] iterates in an unspecified order. We ensure the consistence of these two views by giving a definition of fully-lazy λ-lifting in which the order of the iterative process is irrelevant.

Since λ-lifting is an iterative process that turns progressively a λ-term into a first-order term, none of the intermediate steps is in either of these worlds. Nevertheless, we would like to embody the source, the target, and all the intermediate steps of the transformation into a single formalism. To this aim we use Combinatory Reduction Systems (CRS), a higher-order rewriting framework introduced by J.W. Klop and reviewed in [KvOvR93] that mixes abstractions and symbols. The β-reduction as well as the target first-order reduction have a straightforward encoding into CRS rules. Moreover, fully lazy λ-lifting itself can then be seen as a rewriting process: it is expressed in Section 4 as a confluent and strongly normalizing CRS reduction relation.

We provide a new proof of correctness of fully lazy λ-lifting by showing that the transformation preserves reduction sequences: each single reduction step in the source (resp. target) system is simulated by exactly one single step in the target (resp. source) system. The proof is small-step: the reduction sequences are proved to be preserved in every intermediate step of the transformation. Moreover, we prove that the notion of optimal sharing is also preserved, which has two consequences:

- The direct [Wad71] and the λ-lifting based [Jon87] approaches of full laziness are reduction-wise equivalent.
- Fully-lazy λ-lifting establishes a link between optimal sharing in the weak λ-calculus [BLM07] and the better known optimality theory of first-order rewriting [Mar91, Ter03]. This emphasizes in a new way the "first-order" nature of weak reduction, without any de Bruijn indices or explicit substitutions (contrary to [Mar91]).

A final bonus remark is an incidental point which happens to have some theoretical significance: while β-reduction and λ-lifting considered separately can be seen as orthogonal systems[2], their combination cannot. As far as the author is aware, the system derived in this paper is the first successful optimality-oriented labelling of a non-orthogonal system.

Outline. The paper comprises three main parts: Section 2 presents the abstract notions of prefix, weak reduction, and sharing-via-labelling and gives a proof of the sharing property for the axiomatic framework. Section 3 restricts the axiomatics to enforce full laziness and proves a generic equivalence between several notions of fully lazy sharing. Section 4 focuses on the particular fully lazy system of [Wad71, SW05] whose properties allow a clean definition of fully lazy λ-lifting which establishes a strong link between weak β-reduction and first-order rewriting.

For lack of space most proofs are only sketched here. The full versions are in a companion technical report [Bal11].

2. Sharing and β-reduction

We define in this section an axiomatic framework in which the higher-order approaches [Wad71, AF97, Ses97, SW05, BLM07] to fully lazy sharing can be expressed. We propose an axiomatic notion of *weak β-reduction* in Subsection 2.1, whose optimal sharing is characterized by the *sharing-via-labelling systems* introduced in Section 2.2. Section 2.3 then shows that reduction of labelled terms in sharing-via-labelling systems represents *reduction of graphs*.

All this is expressed in Combinatory Reduction Systems (CRS). For lack of space, we only recall the basic syntax and mechanisms.

[2] in brief, a system is orthogonal when no two rules are applicable to overlapping sets of positions of a term, see for instance [Ter03, Bru03]

We refer the reader to [KvOvR93] for a comprehensive presentation. 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** Σ, and Z is an n-ary **meta-variable**. A **term** is a metaterm without meta-variable, and a **reduction rule** is a pair $L \to R$ of closed metaterms satisfying the following conditions: the meta-variables in L appear as $Z(x_1, ..., x_n)$ with $x_1, ..., x_n$ distinct bound variables, and all the meta-variables of R also appear in L. A rule matches a term by application of a **valuation** σ that maps n-ary meta-variables to n-ary contexts avoiding variable capture. **Reduction** by a rule $L \to R$ with valuation σ in a context c is $c[L^\sigma] \to c[R^\sigma]$.

2.1 Weak β-reduction systems

This section gives an abstract definition of *weak reduction* in the λ-calculus and states one of its crucial properties: disjoint redexes remain disjoint along any reduction sequence (Lemma 1). This lemma serves in particular in the definition of graph reduction in Section 2.3.

Weak reduction forbids the reduction of so-called *frozen redexes*, which are identified by their belonging to the *prefix* of some λ-abstraction. Prefixes are parts of λ-abstractions defined by a *prefix function* satisfying the axioms of a *weak β-reduction system*.

Weak β-reduction systems are CRS over the signature Σ comprising:

- a binary symbol @ for application,
- a unary symbol λ for λ-abstraction,
- a unary dummy symbol ϵ,
- for all $n \in \mathbb{N}$, a countable set $\mathcal{F}_n$ of n-ary symbols.

From now on, by **term** we mean a CRS term over the signature Σ (notation t, u, v, w, a). We use the usual notion of **positions** of terms (notation q), contexts and free variables [KvOvR93]. We write $t^{\{x:=u\}}$ the substitution by u of all the free occurrences of the variable x in t.

Application and λ-abstraction symbols are used to embody λ-terms in this signature, which is made in the usual way: the λ-term $(\lambda x.x)y$ for instance is encoded in the CRS term $@(\lambda([x]x), y)$. We write $\lambda x.t$ as a shorthand for $\lambda([x]t)$. Hence the encoding of $(\lambda x.x)y$ is simply written $@(\lambda x.x, y)$, and the usual β-reduction is represented by the CRS rule $@(\lambda x.Z(x), Z') \to Z(Z')$.

The symbols in the sets $\mathcal{F}_n$ are used in Section 4 to represent supercombinators. Until then they play no role and may be ignored.

The dummy symbol ϵ has no meaning in itself. It is needed for labelling (Subsection 2.2), and serves in particular as a container for dynamically created labels. In the graphical interpretation of labelled terms, the occurrences of ϵ will represent indirections (see Subsection 3.1). 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:

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

The two ϵ's in the right hand sides are used for the correct labelling of collapsing reductions (see Subsection 2.2). The use of the dummy symbol ϵ is inspired by the notion of *expansion* in term rewriting systems [Ter03, Chap. 8].

Write $\rho : t \to t'$ a reduction ρ of a term t to a term t'. The usual notions of **ancestors** and **descendants**, which track subterms along reduction in the λ-calculus are straightforwardly adapted, as illustrated in Example 2. A **residual** of a redex r is a descendant of r which is still a redex.

Example 2.

The term $t = @(\epsilon(\lambda x.@(x, x)), y)$ reduces by rule β_1 to $t' = \epsilon(@(\epsilon(y), \epsilon(y)))$. The two occurrences of y in t' are the descendants of the y in t, and the latter is the ancestor of the formers. The ϵ in t has no descendant and the ϵ's in t' have no ancestor.

We call **plain** λ-calculus the usual reduction relation where the previous rules can be applied in any context. Weak reduction consists in restricting this reduction relation. Particularly, it affects the reduction under λ-abstractions. Before introducing the formal definition, let us present two different well-known examples:

Example 3.

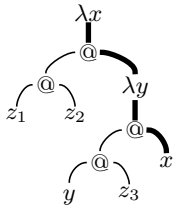
1. *The **naive weak reduction** simply forbids any reduction under λ-abstractions.*
2. *A more refined version, studied in particular in [ÇH98], allows no reduction between an occurrence of a bound variable and its binder. Formally, if r is a redex of contractum r', then the reduction $C[r] \to C[r']$ is allowed if and only if the context C binds no variable that appears free in r. We call this version **CH-weak reduction**.*

It is known that CH-weak reduction yields a confluent weak calculus while naive weak reduction does not [ÇH98].

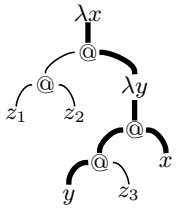
To specify the previous notions, we introduce a notion of prefix: call a n-ary **closed prefix** of a term t a n-ary context p which does not contain any free variable and such that there are terms $t_1, ..., t_n$ satisfying $t = p[t_1, ..., t_n]$. Example 4 gives two closed prefixes of the same term. Call a **prefix function** a function that takes a term t as input and returns a closed prefix of t.

Example 4.

Let $t = \lambda x.@(@(z_1, z_2), \lambda y.@(@(y, z_3), x))$. The two contexts $\lambda x.@([], \lambda y.@([], x))$ and $\lambda x.@([], \lambda y.@(@(y, []), x))$ are two closed prefixes of t, called respectively spine *and* skeleton *(see Section 3.1). These two prefixes are marked with bold lines in the two following pictures.*



Spine



Skeleton

A *weak β-reduction system* is defined below by a prefix function $\mathcal{P}$ satisfying some conditions. The first condition is a simple restriction linked to bound variables. The second condition controls the evolution of $\mathcal{P}(t)$ when free variables of t are substituted by terms. In particular, $\mathcal{P}(t^{\{x:=u\}})$ is required to contain $\mathcal{P}(t)$, and the extension from $\mathcal{P}(t)$ to $\mathcal{P}(t^{\{x:=u\}})$ has to be uniform. This uniformity is enforced by the use of an auxiliairy function $\mathcal{P}^\Box$.

Call a **weak β-reduction system** a prefix function $\mathcal{P}$ such that:

- For any term $\lambda x.t$ such that $\mathcal{P}(\lambda x.t) = p$ and $\lambda x.t = p[t_1, ..., t_n]$, the variable x does not appear free in any of the t_i's (which are called the **parameters** of $\lambda x.t$). In other words, $\mathcal{P}(\lambda x.t)$ contains all the occurrences of x that are free in t.
- There is an auxiliary prefix function $\mathcal{P}^\Box$ such that for any p in the codomain of $\mathcal{P}$ and for any terms $t_1, ..., t_n$ where no free

variable of a t_i is bound in p, the equation $\mathcal{P}(p[t_1, ..., t_n]) = p[\mathcal{P}^{\Box}(t_1), ..., \mathcal{P}^{\Box}(t_n)]$ holds.

A weak β-reduction system defines a notion of **weak reduction** as follows: β-reduction is forbidden in the prefix of any λ-abstraction. Call a **frozen position** of a term t a position that is in the prefix of some λ-abstraction of t. Call a **frozen β-redex** a redex whose main @ symbol occurs at a frozen position.

Example 5.

The two weak reductions of Example 3 can be captured by our axiomatic definition:

1. *Naive weak reduction is given by $\mathcal{P}_n$ such that $\mathcal{P}_n(t) = p$ where $t = p[x_1, ..., x_n]$ and $x_1, ..., x_n$ are all the free variable occurrences of t. The auxiliary function is $\mathcal{P}_n^{\Box} = \mathcal{P}_n$ (the whole substituted term is included into the prefix).*
2. *CH-weak reduction can be given by $\mathcal{P}_{ch}$ such that $\mathcal{P}_{ch}(t) = p$ where $t = p[t_1, ..., t_n]$ and $t_1, ..., t_n$ are all the* maximal *free expressions of t. The auxiliary function $\mathcal{P}_{ch}^{\Box}$ is the constant mapping returning the empty unary context $[]$ (the prefix is stable by substitution). We will see in Section 3.2 that $\mathcal{P}_{ch}$ is not the unique representation of CH-weak reduction.*

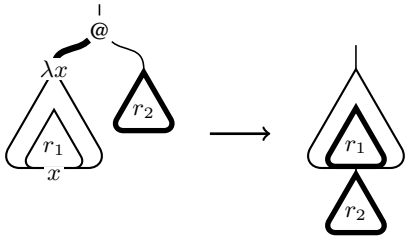
An important feature of weak reduction is that it cannot nest the residuals of disjoint redexes. This fact is formalized in Lemma 1, and will be useful in Section 2.3 to ground the notion of parallel reduction.

Lemma 1 (Disjoint residuals). *Let $\rho : t \to t'$ be a reduction and $r_1, ..., r_n$ (non frozen) redexes of t occurring at disjoint positions. Then the descendants of $r_1, ..., r_n$ also occur at disjoint positions.*

Example 6 shows why Lemma 1 is a feature of weak reduction which is not valid in the plain λ-calculus.

Example 6.

Suppose r_1 and r_2 are two redexes. In the left term r_1 is frozen for any weak β-reduction system.



To prepare the implementation of optimal sharing for weak β-reduction systems in next section (2.2), we give a characterization of redex creation in these systems. Suppose $\rho : t \to_\beta t'$ in a weak β-reduction system. A redex of t' is **created** by ρ if it is not the descendant of a (non frozen) redex of t. The reduction can create redexes in t' at exactly three places:

1. At the root of the contractum, the body of the main λ-abstraction is connected to the context. This can create a new contact between an application and a λ-abstraction.
2. At the places where a substitution occurs, the argument is connected to the body of the main λ-abstraction, or to the context if the body is degenerated.
3. In the prefix of the main λ-abstraction, a previously frozen redex can be "unfrozen" by ρ, as r_1 in Example 6. In other words, a reduction forbidden by the weak restriction in t can be authorized in t'.

2.2 Sharing-via-labelling systems

We define in this section a *labelling* for weak β-reduction systems which characterizes optimal sharing and yields a graph implementation. As in [Lé80], the labels record the past history of a term. This is done in a distributed way since each label remembers only what is relevant to its position. The important point for optimality is that the labels of a redex r are characterized by the past reductions that contributed to the creation of r, which is ensured by the concluding lemmas 2 and 3. These two lemmas also play a key role in the proof of the preservation of the graph structure (so-called sharing property) in Section 2.3.

To embody this *contribution* relation into the labels, we build compound labels of the form $[\Omega, \alpha]$ where a pre-existing label α is *modified* by the *name* Ω of a contributing redex. The *labelled β-reduction* then modifies the labels of the positions where the reduction can contribute to something, following the characterization of redex creation given at the end of the previous subsection. Names and contribution are required to satisfy three axioms which ensure that the name of a redex correctly reflects its contributors.

The labelled terms are formalized as usual CRS terms over a *labelled signature*. Since the labels should not interfere with the normal reduction behaviour, the labelled β-reduction is defined for any possible labelling of the source.

For any countable set $\mathcal{L}$ whose elements are called labels (and written α, β, γ...), a **labelled signature** $\Sigma_{\mathcal{L}}$ is defined as the set $\{f^\alpha | f \in \Sigma, \alpha \in \mathcal{L}\}$. From now on, a **$\mathcal{L}$-labelled term** denotes a CRS term over $\Sigma_{\mathcal{L}}$. In other words, the labels are associated to the symbols, and never directly to the variables or the bindings. Remark that the labels are arbitrary objects: in a concrete definition they can be simple letters as well as richly structured objects.

We write $\lambda^\alpha x.t$ (resp. x^α) as a shorthand for $\lambda^\alpha([x]t)$ (resp. $\epsilon^\alpha(x)$). Write $\tau(t)$ for the label of the root symbol of the labelled term t. Write $\epsilon_k^{\alpha_1 \cdots \alpha_k}(t)$ as a shorthand for $\epsilon^{\alpha_1}(...\epsilon^{\alpha_k}(t))$, where the case $k = 0$ represents t.

Write $|.|$ the (trivial) map from terms over $\Sigma_{\mathcal{L}}$ to terms over Σ that removes the labels of all symbols. By requiring the condition $|\mathcal{P}(t)| = \mathcal{P}(|t|)$, we get a straightforward extension to $\Sigma_{\mathcal{L}}$ of any weak β-reduction system $\mathcal{P}$ over Σ.

Since neither ϵ's nor labels shall interfere with β-reduction, labelled β-redexes allow any number of ϵ occurrences and can be decorated with any labels: for any $\mathcal{L}$, a **$\mathcal{L}$-labelled β-redex** is a $\mathcal{L}$-labelled term of the form $@^\alpha(\epsilon_k^{\beta_1 \cdots \beta_k}(\lambda^\gamma x.t), a)$. Labelled β-reduction is defined later, since it requires an additional notion of sharing-via-labelling systems given below.

We consider sets of labels of the form $\mathcal{L} = \mathcal{V}^{\mathcal{N}}$, generated by the following grammar for any two countable sets $\mathcal{V}$ and $\mathcal{N}$:

$$\mathcal{V}^{\mathcal{N}} \quad ::= \quad \mathcal{V} \mid [\mathcal{N}, \mathcal{V}^{\mathcal{N}}]$$

The label $[\Omega, \alpha]$ denotes "the label α modified by Ω". Write $[\Omega_1 ... \Omega_n, \alpha]$ as a shorthand for $[\Omega_1, ...[\Omega_n, \alpha]]$. Any Ω_i is called a **modifier** of $[\Omega_1 ... \Omega_n, \alpha]$. The *virginal* labels (the labels in $\mathcal{V}$) denote positions that are free from any past history, they will be modified into labels of the form $[\Omega_1 ... \Omega_n, \alpha]$ along the reductions. Let $\mathcal{S}$ be a tuple $\langle \mathcal{P}, \mathcal{N}, \mathcal{V}, \eta, \hookrightarrow \rangle$ where:

- $\mathcal{P}$ is a weak β-reduction system.
- $\mathcal{N}$ is a countable set whose elements are called **(redex) names** (notation Ω).
- $\mathcal{V}$ is a countable set whose elements are called **virginal labels**, with two distinguished elements $\top$ and $\bot$.
- η is a function from $\mathcal{V}^{\mathcal{N}}$-labelled redexes to names.
- $\hookrightarrow$ is a transitive and irreflexive relation on $\mathcal{N}$ called **contribution relation**.

The **terms** considered in $\mathcal{S}$ are the $\mathcal{V}^{\mathcal{N}}$-labelled terms. Call **virginal term** a term whose labels are all virginal and different.

Write $\uparrow\Omega = \{\Omega' \mid \Omega' \hookrightarrow \Omega\}$ the set of all the contributors of the name Ω. Contribution is extended to labels: define $\uparrow\alpha = \emptyset$ if α is a virginal label, and $\uparrow[\Omega,\alpha] = \{\Omega\} \cup \uparrow\Omega \cup \uparrow\alpha$ otherwise. Write $\Omega \hookrightarrow \alpha$ when $\Omega \in \uparrow\alpha$.

$\mathcal{S}$ is a **sharing-via-labelling** system if the following axioms are satisfied:

- (*Redex contributors*)
$$\uparrow\eta(@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.t), a)) = \uparrow\alpha \cup (\bigcup_i \uparrow\beta_i) \cup \uparrow\gamma$$
- (*Name scope*) If $\lambda^\gamma x.p$ is in the codomain of $\mathcal{P}$, and a_t, a_u are any terms and $t_1, ..., t_n, u_1, ..., u_n$ are terms whose free variables are not bound in $\lambda^\gamma x.p$, then
$$\eta(@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.p[t_1, ..., t_n]), a_t)) = \eta(@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.p[u_1, ..., u_n]), a_u))$$
- (*Name equality*) If $\eta(r) = \eta(r')$ then $\tau(r) = \tau(r')$.

The axiom *Redex contributors* states that the name of a redex collects the contributors of the essential parts of the redex, that means its main application, its main λ-abstraction, and all that lays in between these two positions. The axiom *Name scope* states that the name of a redex does not depend of what is deeper than the smallest prefix of the main abstraction, while axiom *Name equality* states that the equality of the names of two redexes implies the equality of their respective root labels.

Example 7.

Let $\mathcal{V}$ be any countable set.

1. *The names in $\mathcal{N}_{seq}$ are sequences of labels. Since the names also take part into the definition of the labels, a mutually recursive definition of names and labels is required:*
$$\begin{aligned} \mathcal{N}_{seq} &::= \mathcal{L}; ...; \mathcal{L} \\ \mathcal{L} &::= \mathcal{V}^{\mathcal{N}_{seq}} \end{aligned}$$
Define the name of a redex as
$$\eta_{seq}(@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.t), a)) = \alpha; \beta_1; ...; \beta_k; \gamma$$
The contribution relation is defined by: $\Omega \hookrightarrow_{seq} \alpha_1; ...; \alpha_n$ if and only if there is at least one i such that $\Omega \hookrightarrow_{seq} \alpha_i$.
For $\mathcal{P} \in \{\mathcal{P}_n, \mathcal{P}_{ch}\}$, the system $\langle \mathcal{P}, \mathcal{N}_{seq}, \mathcal{V}, \eta_{seq}, \hookrightarrow_{seq}\rangle$ is a sharing-via-labelling system. The definitions of $\mathcal{N}_{seq}$, η_{seq}, and $\hookrightarrow_{seq}$ correspond to the system presented in [BLM07].
2. *The names in $\mathcal{N}_{ctx}$ are $\mathcal{L}$-labelled contexts, with once again a mutually recursive definition using $\mathcal{L} ::= \mathcal{V}^{\mathcal{N}_{ctx}}$. Define*
$$\eta_{ctx}(@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.t), a)) = @^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.p), [])$$
where $\mathcal{P}(\lambda^\gamma x.t) = \lambda^\gamma x.p$.
Write $\Omega \hookrightarrow_{ctx} c$ if there is a label α in c such that $\Omega \hookrightarrow_{ctx} \alpha$.
For $\langle \mathcal{P}_{ch}, \mathcal{N}_{ctx}, \mathcal{V}, \eta_{ctx}, \hookrightarrow_{ctx}\rangle$ to be a sharing-via-labelling system, the axiom Redex contributors requires that the prefixes satisfy the following property: all the contributors of the labels of a prefix $\lambda^\gamma x.p$ contribute to γ. Fortunately, in this system this property is an invariant of labelled β-reduction (defined below). The system $\langle \mathcal{P}_n, \mathcal{N}_{ctx}, \mathcal{V}, \eta_{ctx}, \hookrightarrow_{ctx}\rangle$ is not a sharing-via-labelling system since it breaks the axiom Name scope.

Labelled β-reduction is defined by a rule scheme which propagates the name of the reduced redex in the reduced term for recording of the contributions. For this, the constructor $[.,.]$ extends to a function on labelled terms: $[\Omega, t]$ is defined as the labelled term t in which all the labels are modified by the function $\alpha \mapsto [\Omega, \alpha]$.

Labelled β-reduction in a sharing-via-labelling system is defined by the rule scheme:
$$@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.p[t_1, ..., t_n]), a) \to_\beta \epsilon^{[\Omega,\top]}([\Omega, p])^{\{x := \epsilon^{[\Omega,\bot]}(a)\}}[t_1, ..., t_n]$$
where $\lambda^\gamma x.p = \mathcal{P}(\lambda^\gamma x.p[t_1, ..., t_n])$ and
where $\Omega = \eta(@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.p[t_1, ..., t_n]), a))$.
The name Ω is added in three areas of the reduced term:

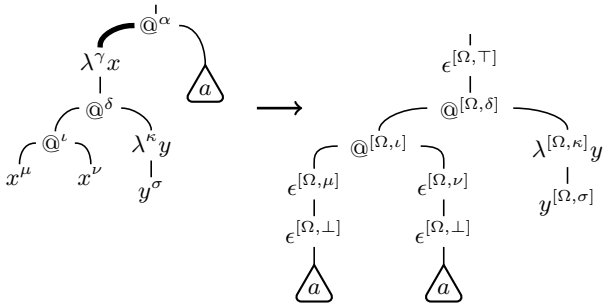
1. At the root of the contractum, with the new label $[\Omega, \top]$.
2. At the places where a substitution occurs, with the new label $[\Omega, \bot]$. Remark in Example 8 how $\top$ and $\bot$ work as parentheses in the syntactic tree of the term.
3. In the prefix of the main abstraction of the redex, as an additional modifier to pre-existing labels.

Remark that these three places follow the three cases of redex creation given in Section 2.1 and that removing the labels in this rule yields exactly the unlabelled β-reduction of Section 2.1.

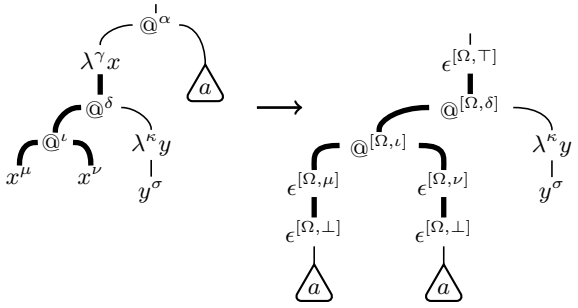
Example 8.

Let $r = @^\alpha(\lambda^\gamma x.@^\delta(@^\iota(x^\mu, x^\nu), \lambda^\kappa y.y^\sigma), a)$ be a β-redex. We reduce r in two sharing-via-labelling systems of Example 7.

1. *In $\langle \mathcal{P}_n, \mathcal{N}_{seq}, \mathcal{V}, \eta_{seq}, \hookrightarrow_{seq}\rangle$ the name of r is the sequence $\Omega = \alpha; \gamma$, and the prefix of the λ-abstraction is its whole body. Then all the labels are modified in the contractum:*



2. *In $\langle \mathcal{P}_{ch}, \mathcal{N}_{ctx}, \mathcal{V}, \eta_{ctx}, \hookrightarrow_{ctx}\rangle$, the name of r is the labelled context $\Omega = @^\alpha(\lambda^\gamma x.@^\delta(@^\iota(x^\mu, x^\nu), []), [])$. The only modified labels are those of the prefix of the λ-abstraction, marked with bold lines below.*



This section ends with the two lemmas 2 and 3, which show how the labels reflect the contribution relation between redexes. In particular, the name of a redex r characterizes the past reductions that led to the creation of r.

Lemma 2 (Redex stability). *Let r be a (non frozen) redex of a term t. If r_d is a descendant of r after a reduction $\rho : t \to_\beta t'$, then r_d is still a (possibly frozen) redex and $\eta(r_d) = \eta(r)$.*

Lemma 3 (Direct contribution). *If a redex of name Ω_c is created by the reduction of a redex of name Ω, then $\Omega \hookrightarrow \Omega_c$.*

2.3 Sharing

This section proves the main property of sharing-via-labelling systems: *parallel labelled reduction* simulates graph reduction (Theorem 1). As in [BLM07], the labelled terms are linked to graphs with the sharing-via-labelling principle seen in the introduction: labels are interpreted as memory locations. The proof of the simulation is then done by ensuring that the two following invariants are preserved by parallel labelled reduction:

1. A term t has the **sharing property**, written $\mathbb{S}(t)$, when any two subterms of t with same label are syntactically equal. This is the main property we want to preserve.
2. A term t has the **maximality property**, written $\mathbb{M}(t)$, when for any (non frozen) redex of name Ω and any subterm of label α in t it is not true that $\Omega \hookrightarrow \alpha$. This property is widely used in the subsequent proofs, it ensures in particular that all the occurrences of a given label are created at the same time (see Example 9). Lemmas 2 and 3 are the cornerstone of the preservation of the maximality property.

Parallel labelled reduction is defined for any term t satisfying the sharing property $\mathbb{S}(t)$: let α be the label of a (non frozen) redex of t. Since $\mathbb{S}(t)$ holds, all the redexes labelled by α are equal and have disjoint positions (however, some may be frozen). The parallel labelled reduction of α, written $t \overset{\alpha}{\Rightarrow}_\beta t'$, is then defined as the simultaneous replacement of all the (non frozen) redexes with label α by their contractum. By lemmas 1 and 2, parallel labelled reduction is well defined as a sequence of single steps, for instance any iterated reduction of the non frozen redexes with label α.

Theorem 1 (Preservation of sharing). *If $\mathbb{M}(t)$, $\mathbb{S}(t)$ and $t \overset{\alpha_0}{\Longrightarrow}_\beta t'$, then $\mathbb{M}(t')$, and $\mathbb{S}(t')$.*

Proof. (Sketch) Verification of $\mathbb{M}(t')$. Suppose there is a redex r' with name Ω' and a subterm u' with label α' in t' such that $\Omega' \hookrightarrow \alpha'$. If r' is created by the reduction, then Lemma 3 contradicts $\mathbb{M}(t)$. Else, Lemma 2 and axiom *Redex contributors* contradict either $\mathbb{M}(t)$ or axiom *Name equality*.

Verification of $\mathbb{S}(t')$. Let u' and v' be two subterms of t' with same label α'. By case on the origin of both labels α': if one is created and the other is a descendant of a label α' of t, then by $\mathbb{M}(t)$ and Lemma 3 we reach a contradiction, else both have the same origin and evolution (using $\mathbb{S}(t)$). □

Example 9 shows why maximality $\mathbb{M}$ is necessary to the preservation of sharing $\mathbb{S}$.

Example 9.

Consider the system $\langle \mathcal{P}_{ch}, \mathcal{N}_{seq}, \mathcal{V}, \eta_{seq}, \hookrightarrow_{seq} \rangle$ defined in Example 7 and the labelled term $t = @^\alpha(x^{[\gamma;\delta,\iota]}, @^\gamma(\lambda^\delta z.z^\iota, y^\kappa))$. The property $\mathbb{S}(t)$ holds since all the labels of t are different. Remark that t contains a redex $@^\gamma(\lambda^\delta z.z^\iota, y^\kappa)$ of name $\gamma;\delta$. Hence $\mathbb{M}(t)$ does not hold since t contains a label $[\gamma;\delta,\iota]$.
Then $t \overset{\gamma}{\Rightarrow}_\beta @^\alpha(x^{[\gamma;\delta,\iota]}, \epsilon^{[\gamma;\delta,\iota]}(y^\kappa)) = t'$ where the two subterms of t' with label $[\gamma;\delta,\iota]$ are different: $\mathbb{S}(t')$ is falsified.

Finally, labelled terms represent graphs and parallel labelled reduction represents graph reduction.

3. Full laziness

This section shows how the various known implementations of full laziness correspond to several (at least three) different sharing-via-labelling systems (Section 3.1). Thus, they correspond to different graph reductions featuring different amounts of sharing. However, we are going to prove in Section 3.2 that these implementations are reduction-wise equivalent.

Along this section, the signature Σ is restricted to $\{@, \lambda, \epsilon\}$.

3.1 Encodings into sharing-via-labelling systems

The motto of sharing-via-labelling is "labels denote memory locations". What happens to the labels during reduction describes directly what happens to the nodes of the corresponding graph:

- A new label corresponds to a new node. There are two cases:
  - A label of the form $[\Omega, \top]$ or $[\Omega, \bot]$ appears only on ϵ. It represents a new indirection node that contains a pointer leading to the term.
  - Any other $[\Omega, \alpha]$ denotes a new copy of a node labelled α.
- An unchanged label is a node unaffected by the reduction.

The key of the encoding of graph reduction systems or closure-based systems into sharing-via-labelling systems is to modify exactly the labels of what is needed to be copied. Since the rules of a sharing-via-labelling system modify exactly the labels of the prefix of the main λ-abstraction, this amounts to take as prefix of a λ-abstraction exactly what has to be duplicated of its body.

Remark 1. *Our β-rule falsely suggests that an arbitrary number of indirections can be contracted in unit time. The techniques presented in [Jon87] can be used to avoid chains of indirections.*

To describe the encodings of the higher-order approaches to full laziness into sharing-via-labelling systems, we formally define two useful prefixes mentioned in Example 4.

- Call **spine** of a term $\lambda x.t$ the prefix $\lambda x.p$ where p is the prefix of t which contains exactly the positions that are above a free occurrence of x, including the free occurrences of x. Remark that in any weak β-reduction system $\mathcal{P}$, any prefix $\mathcal{P}(\lambda x.t)$ contains the spine of $\lambda x.t$.
- Call **skeleton** of a term $\lambda x.t$ the prefix $\lambda x.p$ where p is the prefix of t containing exactly the positions that are not in a free expression of $\lambda x.t$. As done in [SW05] the skeleton can also be seen as an iterated spine: to get the skeleton of $\lambda x.t$, start with the spine of $\lambda x.t$ and iteratively add to the obtained prefix the spines of all the λ-abstractions that are in the prefix built so far.

The two approaches by C.P. Wadsworth [Wad71] and O. Shivers and M. Wand [SW05] reach fully lazy sharing by two graph implementations in which the duplicated part of an instantiated function is its skeleton. The former uses the definition based on the maximal free expressions while the latter follows the characterization by iterated spine. They are both represented by the weak β-reduction system $\mathcal{P}_{ch}$ such that $\mathcal{P}_{ch}(\lambda x.t)$ is the skeleton of $\lambda x.t$.

In [Ses97], P. Sestoft revises Launchbury's lazy semantics [Lau93] and proposes a fully lazy variant using additional let-bindings: if $\lambda x.p$ is the skeleton of $\lambda x.t$ and $\lambda x.t = \lambda x.p[t_1, ..., t_n]$, then $\lambda x.t$ is replaced by *let* $x_1 = t_1, ..., x_n = t_n$ *in* $\lambda x.p[x_1, ..., x_n]$ with $x_1, ..., x_n$ fresh variables. After this extraction of the maximal free expressions $t_1, ..., t_n$ of $\lambda x.t$, a duplication of the λ-abstraction duplicates the subterm $\lambda x.p[x_1, ..., x_n]$ but does not duplicate the let-parameters $t_1, ..., t_n$. This is again represented by the weak reduction system $\mathcal{P}_{ch}$.

The work by T. Blanc, J.-J. Lévy and L. Maranget [BLM07] already uses a system isomorphic to a sharing-via-labelling system.

Their labelled β-reduction modifies only the labels of the spine of the main λ-abstraction. Thus it corresponds to a weak β-reduction system $\mathcal{P}_{blm}$ where $\mathcal{P}_{blm}(\lambda x.t)$ is the spine of $\lambda x.t$. Moreover their frozen redexes are the redexes containing a free occurrence of a variable bound above: they coincide with those given by $\mathcal{P}_{ch}$.

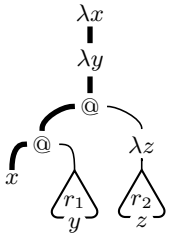
In their call-by-need λ-calculus [AF97], Z. Ariola and M. Felleisen allow the substitution, and thus the duplication, of values. Their fully lazy extension consists in restricting these allowed duplications to a set of fully lazy values: values that do not contain any "free expression". The difference with the previous cases lies in their non-standard definition of free expressions: they use the usual criterion given in Section 1.2 but they exclude the variables and the λ-abstractions. Hence their fully lazy values correspond to the prefixes of the weak β-reduction system $\mathcal{P}_{af}$ such that:

- If $\lambda x.p$ is the skeleton of $\lambda x.t$ and $\lambda x.t = \lambda x.p[t_1, ..., t_n]$ then $\mathcal{P}_{af}(\lambda x.t) = \lambda x.p[\mathcal{P}^{\Box}_{af}(t_1), ..., \mathcal{P}^{\Box}_{af}(t_n)]$.
- $\mathcal{P}^{\Box}_{af}(\lambda x.t) = \mathcal{P}_{af}(\lambda x.t)$.
- $\mathcal{P}^{\Box}_{af}(@(t_1, t_2)) = []$.
- If $\mathcal{P}^{\Box}_{af}(t) = []$ then $\mathcal{P}^{\Box}_{af}(\epsilon(t)) = []$ else $\mathcal{P}^{\Box}_{af}(\epsilon(t)) = \epsilon(\mathcal{P}^{\Box}_{af}(t))$.

Finally, the higher-order approaches [Wad71, AF97, Ses97, SW05, BLM07] to full laziness correspond to three different weak β-reduction systems $\mathcal{P}_{ch}$, $\mathcal{P}_{blm}$ and $\mathcal{P}_{af}$, which we are going to relate in the next section. Example 10 illustrates how the systems $\mathcal{P}_{blm}$ and $\mathcal{P}_{af}$ can yield the same parallel labelled reduction in spite of their differences.

Example 10.

Let $t = \lambda x.\lambda y.@(@(x, r_1), \lambda z.r_2)$ be a term such that r_1, r_2 are two redexes that do not contain any free occurrence of x. Hence a duplication of the spine $\mathcal{P}_{blm}(t)$ of t (marked with bold lines in the picture) do not duplicate these redexes.



If r_1 (resp. r_2) contains a free occurrence of y (resp. z), then both redexes are at least partially contained in $\mathcal{P}_{af}(t)$, and thus duplicated in this system. However, in this case r_1 and r_2 are and remain frozen in both systems, and their label will change before they are unfrozen: the additional duplications are not harmful.

3.2 Equivalence of the parallel labelled reductions

In this section we define a family $\mathcal{W}$ of weak β-reduction systems that contains $\mathcal{P}_{ch}$, $\mathcal{P}_{blm}$ and $\mathcal{P}_{af}$ and we show that all the sharing-via-labelling systems based on the weak β-reduction systems of $\mathcal{W}$ are equivalent, in the sense that they can simulate one another using exactly the same number of shared reduction steps. For this we define a relation $\sim$ on labelled terms which is stable by reduction, and prove that any two terms in relation share the same non-frozen redexes. The relation $\sim$ relates two terms t_1 and t_2 of two different systems $\mathcal{S}_1$ and $\mathcal{S}_2$ whenever t_1 and t_2 can be reached from a common source by two equivalent reduction sequences in $\mathcal{S}_1$ and $\mathcal{S}_2$ (the whole being called a *mirror reduction sequence*).

The definition of $\mathcal{W}$ follows two ideas: the weak β-reduction systems in $\mathcal{W}$ shall enforce CH-weak reduction (for this the third point in particular prevents some applications to be included into a prefix), and all the prefixes are built from spines and skeletons (the first two points, which make the proofs tractable).

A weak β-reduction system $\mathcal{P}$ is in $\mathcal{W}$ if the following additional conditions are satisfied:

- For each abstraction $t = \lambda x.u$, either $\mathcal{P}(t)$ is the spine of t, or $\mathcal{P}(t) = p[\mathcal{P}^{\Box}(t_1), ..., \mathcal{P}^{\Box}(t_n)]$ where p is the skeleton of t and $t = p[t_1, ..., t_n]$.
- For any $\lambda x.t$, $\mathcal{P}^{\Box}(\lambda x.t) = []$ or $\mathcal{P}^{\Box}(\lambda x.t) = \mathcal{P}(\lambda x.t)$.
- $\mathcal{P}^{\Box}(@(t_1, t_2)) = []$.
- If $\mathcal{P}^{\Box}(t) = []$ then $\mathcal{P}^{\Box}(\epsilon(t)) = []$ else $\mathcal{P}^{\Box}(\epsilon(t)) = \epsilon(\mathcal{P}^{\Box}(t))$.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two systems of $\mathcal{W}$. A **mirror reduction sequence** in the systems $\mathcal{P}_1$ and $\mathcal{P}_2$ is a pair (ϱ_1, ϱ_2) such that $\varrho_i = \rho_i^1 ... \rho_i^n$ is a reduction sequence in $\mathcal{P}_i$ and for any $j \in \{1...n\}$ the redexes reduced by the single steps ρ_1^j and ρ_2^j have the same position q_j in the respective source terms. The notion of mirror reduction sequence extends to parallel labelled reduction in two sharing-via-labelling systems based on two (possibly equal) weak β-reduction systems of $\mathcal{W}$.

Let t_1 (resp. t_2) be a term in a sharing-via-labelling system $\mathcal{S}_1$ (resp. $\mathcal{S}_2$). Write $t_1 \sim t_2$ when there is a virginal term t_0 in the intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$ and a mirror reduction sequence (ϱ_1, ϱ_2) in the systems $\mathcal{S}_1$ and $\mathcal{S}_2$ such that ϱ_i is a parallel labelled reduction sequence from t_0 to t_i.

Remark 2. *If $t_1 \sim t_2$ then the two terms have the same set of positions. In particular they contain the same number of ϵ occurrences.*

For any labelled term t, the labels in t induce an equivalence relation on the subterms of t. Say that two subterms u_1 and u_2 of t are **label-equivalent** if and only if $\tau(u_1) = \tau(u_2)$.

Theorem 2 (Sharing equivalence). *If $t_1 \sim t_2$ then the labellings of t_1 and t_2 induce the same label-equivalence on their non frozen applications.*

Proof. (Sketch) We prove two invariants on the pair (t_1, t_2):

1. The equivalence classes of non frozen applications are equal.
2. In each spine, the equivalence classes of applications are equal.

By definition of $t_1 \sim t_2$ there is a parallel labelled mirror reduction sequence (ϱ_1, ϱ_2) leading from a virginal term t_0 to (t_1, t_2). The proof of both invariants is by induction on the number of parallel steps in (ϱ_1, ϱ_2). Remark by Theorem 1 that any intermediate term t in the mirror reduction sequence satisfies $\mathbb{M}(t)$ and $\mathbb{S}(t)$. □

Theorem 2 proves in particular that $\sim$ is a bisimulation: any diagram $\begin{array}{ccc} t & \sim & u \\ {\scriptstyle\beta}\downarrow & & \\ t' & & \end{array}$ or $\begin{array}{ccc} t & \sim & u \\ & & \downarrow{\scriptstyle\beta} \\ & & u' \end{array}$ can be closed as $\begin{array}{ccc} t & \sim & u \\ {\scriptstyle\beta}\downarrow & & \downarrow{\scriptstyle\beta} \\ t' & \sim & u' \end{array}$

This means that in $\mathcal{W}$, two sharing-via-labelling systems generate the same notion of parallel labelled reduction. In other words, their (possibly different) sharings have the same impact on the reduction. This applies in particular to $\mathcal{P}_{ch}$, $\mathcal{P}_{blm}$ and $\mathcal{P}_{af}$, and thus it shows that all the notions of full laziness in [Wad71, AF97, Ses97, SW05, BLM07] define the same reduction spaces.

4. Fully lazy λ-lifting

The primary goal of the present section is to prove that the notion of full laziness defined by fully lazy λ-lifting in [Hug82, Jon87] is equivalent to the unified notion of the previous section. This study of λ-lifting also reveals a strong relationship between optimal sharing in weak λ-calculi [BLM07] and the optimality theory of first-order rewriting [Mar91, Ter03].

In this section fully lazy λ-lifting is ultimately seen as a morphism between two systems: the **source** is the set of the λ-terms

equipped with weak β-reduction, whereas the **target** is the set of first-order terms built with supercombinators equipped with their associated first-order reduction rules. However, the source and the target system are mixed in the intermediate steps. Fully lazy λ-lifting is then defined as an endomorphism of an **object** system combining the source and the target.

Our object system is a CRS over a labelling of the signature Σ defined in Section 2.1. The weak β-reduction is as defined in Section 2. The supercombinators forming the target subsystem are represented by the symbols in $\mathcal{F} = \bigcup_n \mathcal{F}_n$ and their reduction rules are defined in Section 4.1. Fully lazy λ-lifting itself is represented by a set of CRS rules which is proved to be confluent (Lemma 4), to be strongly normalizing (Lemma 5), to preserve one-step reduction (Theorem 3) and to preserve shared reduction (Theorem 4). Since we aim for this last result on shared reduction and since shared reduction is formalized in this paper by the labels, our λ-lifting is defined on labelled terms.

Section 4.1 introduces an extension of sharing-via-labelling systems and defines fully lazy λ-lifting as a rewriting process. Section 4.2 proves that fully lazy λ-lifting is a bisimulation between the source system and the target system, and Section 4.3 proves that parallel labelled fully lazy λ-lifting preserves optimal sharing.

4.1 Fully lazy λ-lifting systems

This section introduces the fully lazy λ-lifting systems as an extension of the sharing-via-labelling systems. Then, λ-lifting is defined as a CRS reduction in a fully-lazy λ-lifting system. This reduction is confluent and strongly normalizing.

The basic mechanism of λ-lifting is the replacement of a whole prefix by a supercombinator, the prefix and the supercombinator being related by an abstract invertible function called *contractor*. In order to preserve sharing, this replacement has to preserve all the information contained in the labels. Our solution consists in labelling the supercombinator with a structured label containing all the labels of the contracted prefix. Hence the basic operations of λ-lifting are reversible, and we define two inverse transformations (*contraction* and *expansion*) which reversibly relate labelled prefixes and labelled supercombinators.

Fully lazy λ-lifting is related in this paper to the weak β-reduction system $\mathcal{P}_{ch}$ of Example 5, whose definition is reminded below. The ch subscript is omitted along this section.

- $\mathcal{P}(\lambda x.t)$ is the skeleton of $\lambda x.t$.
- $\mathcal{P}^{\Box}(t) = []$ for any term t.

The structured labels used for labelled λ-lifting are tree-shaped, similarly to terms (see Example 11). For any countable set $\mathcal{V}$ (whose elements are called **atomic labels**), the set $\widehat{\mathcal{V}}$ of **tree labels** over $\mathcal{V}$ is defined by the following grammar:

$$\widehat{\mathcal{V}} ::= \Box \mid \top \mid \bot \mid \mathcal{V} \mid \widehat{\mathcal{V}}(\widehat{\mathcal{V}}, ..., \widehat{\mathcal{V}})$$

The label $\Box$ is an "empty" label which is used to denote the lack of label of the empty context [] (see Example 11). The labels in $\widehat{\mathcal{V}}$ are used as virginal labels along this section. A tree label is **well-formed** when no atomic label appears twice in it. For any $\Omega_1, ..., \Omega_n$, a label $[\Omega_1 ... \Omega_n, \alpha]$ with $\alpha \in \widehat{\mathcal{V}}$ is well-formed when α is well-formed.

The **clash** relation is defined on tree labels as: $\alpha \between \beta$ when α and β have an atomic label in common. The notion is used in Section 4.3 to express invariants on labels. For any $\mathcal{V}$ and any $\mathcal{N}$, the clash extends to $\widehat{\mathcal{V}}^{\mathcal{N}}$ as follows: $\alpha \between \beta$ if and only if $\alpha = [\Omega_1 ... \Omega_n, \alpha^*]$ and $\beta = [\Omega_1 ... \Omega_n, \beta^*]$ with $\alpha^*, \beta^* \in \widehat{\mathcal{V}}$ and $\alpha^* \between \beta^*$.

A **contractor** is an injective partial function φ mapping unlabelled n-ary skeletons to unlabelled n-ary function symbols. For any set of atomic labels $\mathcal{V}$ and any set of names $\mathcal{N}$, a contractor φ is extended to $\widehat{\mathcal{V}}^{\mathcal{N}}$-labelled skeletons and $\widehat{\mathcal{V}}^{\mathcal{N}}$-labelled symbols by the following rules.

- If p is labelled with virginal labels, then $\varphi(p) = f^{\alpha}$ such that $f = \varphi(|p|)$ and $\alpha = \|p\|$, where

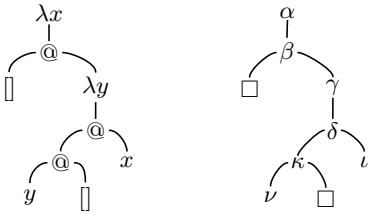
$$\begin{array}{rcl} \|[]\| & = & \Box \\ \|x^{\alpha}\| & = & \alpha \\ \|\epsilon^{\alpha}(t)\| & = & \alpha(\|t\|) \\ \|\lambda^{\alpha}x.t\| & = & \alpha(\|t\|) \\ \|@^{\alpha}(t_1, t_2)\| & = & \alpha(\|t_1\|, \|t_2\|) \\ \|f^{\alpha}(t_1, ..., t_n)\| & = & \alpha(\|t_1\|, ..., \|t_n\|) \end{array}$$

- If p can be decomposed as $p = [\Omega, p']$, then $\varphi(p) = [\Omega, \varphi(p')]$.
- Else, $\varphi(p)$ is undefined.

The extension of φ to labelled skeletons is still injective, hence the labelled φ admits an inverse.

Example 11.

1. *Consider the skeleton* $p = \lambda^{\alpha}x.@^{\beta}([], \lambda^{\gamma}y.@^{\delta}(@^{\kappa}(y^{\nu}, []), x^{\iota}))$. *The collected label is* $\|p\| = \alpha(\beta(\Box, \gamma(\delta(\kappa(\nu, \Box), \iota))))$. *It is interesting to compare the graphical representations of the prefix and the label:*



2. *Suppose* $\varphi(\lambda x.@([], x)) = g$. *Then*
   - $\varphi^{-1}(g^{\alpha(\beta(\Box,\gamma))}) = \lambda^{\alpha}x.@^{\beta}([], x^{\gamma})$
   - $\varphi^{-1}(g^{[\Omega,\alpha(\beta(\Box,\gamma))]}) = \lambda^{[\Omega,\alpha]}x.@^{[\Omega,\beta]}([], x^{[\Omega,\gamma]})$
   - $\varphi^{-1}(g^{\alpha(\beta(\delta(\iota),\gamma))})$ *is undefined because the label has not the same structure as* $\varphi^{-1}(g)$.
   - $\varphi(\lambda^{[\Omega_1,\alpha]}x.@^{[\Omega_2,\beta]}([], x^{[\Omega_2,\gamma]}))$ *is undefined because the labels have different modifiers* Ω_1 *and* Ω_2.

For any set of atomic labels $\mathcal{V}$, for any set of names $\mathcal{N}$ and for any contractor φ, φ-contraction and φ-expansion are two CRS rule schemes on $\widehat{\mathcal{V}}^{\mathcal{N}}$-labelled terms. For any skeleton $\lambda^{\alpha}x.p$ and symbol f^{β} such that $\varphi(\lambda^{\alpha}x.p) = f^{\beta}$ we have the two rules:

(φ-contraction) $\lambda^{\alpha}x.p[Z_1, ..., Z_n] \rightarrow_c f^{\beta}(Z_1, ..., Z_n)$
(φ-expansion) $f^{\beta}(Z_1, ..., Z_n) \rightarrow_e \lambda^{\alpha}x.p[Z_1, ..., Z_n]$

Call an **object redex** a labelled term having one of the two following forms:

(source redex) $@^{\alpha}(\epsilon_k^{\beta_1...\beta_k}(\lambda^{\gamma}x.t), a)$
(target redex) $@^{\alpha}(\epsilon_k^{\beta_1...\beta_k}(f^{\gamma}(t_1, ..., t_n)), a)$

Remark that the set of object redexes is stable by $\rightarrow_c$ and $\rightarrow_e$.
A **fully lazy λ-lifting system** is a tuple $\langle \varphi, \mathcal{N}, \mathcal{V}, \eta, \hookrightarrow \rangle$ such that:

- $\langle \mathcal{P}, \mathcal{N}, \widehat{\mathcal{V}}, \eta, \hookrightarrow \rangle$ is a sharing-via-labelling system.
- φ is a contractor.
- $\mathcal{P}$ commutes with φ-contraction and φ-expansion.
- η is a function from object redexes to names that is stable by φ-contraction and φ-expansion.

In a fully lazy λ-lifting system $\langle \varphi, \mathcal{N}, \mathcal{V}, \eta, \hookrightarrow \rangle$ we consider $\mathcal{L}$-labelled terms with $\mathcal{L} = \widehat{\mathcal{V}}^{\mathcal{N}}$.

Example 12 shows a straightforward extension of the function η_{seq} of Example 7 which is not stable by contraction. Example 13 then gives stable variants of the functions η_{seq} and η_{ctx}.

Example 12.

Consider the redex $r = @^\alpha(\lambda^\gamma x.@^\delta(x^\iota, y^\kappa), a)$, which can be φ-contracted to $r' = @^\alpha(f^{\gamma(\delta(\iota,\Box))}(y^\kappa), a)$ for some unary symbol f. Then $\eta_{seq}(r) = \alpha;\gamma$ and $\eta_{seq}(r') = \alpha;\gamma(\delta(\iota,\Box))$: the name is not stable.

Example 13.

1. *Define $\eta'_{seq}(@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.t), a)) = \alpha;\beta_1;\dots;\beta_k;\delta$ with $\gamma = [\Omega_1 \dots \Omega_n, \gamma^*]$ and $\delta = [\Omega_1 \dots \Omega_n, \delta^*]$ where γ^* is virginal and δ^* is the leftmost atomic label of γ^*. This name function is stable since the leftmost atomic label is stable by contraction and expansion.*
2. *If $\mathcal{P}(\lambda x.t) = p$, then define $\eta'_{ctx}(@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.t), a)) = @^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(f^\delta([],\dots,[])),[])$ where f^δ is the unique normal form of $\lambda^\gamma x.p$ by λ-lifting (see definition of λ-lifting and lemmas 4 and 5 below). This name function is stable by normalization.*

We call **source reduction** the β-reduction, whose rule scheme can be simplified:

$$@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.p[Z_1,\dots,Z_n]), Z) \underset{\to_\beta}{} \epsilon^{[\Omega,\top]}([\Omega,p])^{\{x:=\epsilon^{[\Omega,\bot]}(Z)\}}[Z_1,\dots,Z_n]$$

where $\lambda^\gamma x.p$ is a skeleton and
where $\Omega = \eta(@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(\lambda^\gamma x.p[z_1,\dots,z_n]), z))$.

We call **target reduction** the first-order reduction defined by the scheme:

$$@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(f^\gamma(Z_1,\dots,Z_n)), Z) \underset{\to_t}{} \epsilon^{[\Omega,\top]}([\Omega,p])^{\{x:=\epsilon^{[\Omega,\bot]}(Z)\}}[Z_1,\dots,Z_n]$$

where $\varphi^{-1}(f^\gamma) = \lambda^\delta x.p$ and
where $\Omega = \eta(@^\alpha(\epsilon_k^{\beta_1\dots\beta_k}(f^\gamma(z_1,\dots,z_n)), z))$.

Call **object reduction** the union of source reduction and target reduction: $\to_o = \to_\beta \cup \to_t$.

Remark 3. *Target reduction can be decomposed by φ-expanding the function symbol and then applying β-reduction.*

Remark 4. *Consider the system $\mathcal{S} = \langle \mathcal{P}_{ch}, \mathcal{N}_{ctx}, \mathcal{V}, \eta'_{ctx}, \hookrightarrow_{ctx}\rangle$ where $\mathcal{N}_{ctx}$ and $\hookrightarrow_{ctx}$ are defined in Example 7, η'_{ctx} is defined in Example 13, and $\mathcal{V}$ is any countable set. The target reduction of $\mathcal{S}$ is isomorphic to what is called Lévy-labelling in [Ter03, Chap. 8].*

For a clean definition of λ-lifting and for simple proofs of its basic properties we use an extended notion of positions of a term. The so-called *φ-positions* contain the usual syntactic positions but also the positions that are "hidden" in supercombinators: through the contractor φ each symbol represents a prefix, and the positions of these prefixes are taken into account in φ-positions.

The set $\mathcal{Q}(t)$ of φ-positions of t is defined by:

$$\begin{aligned}
\mathcal{Q}(x) &= \{\varepsilon\}\\
\mathcal{Q}([]) &= \{\varepsilon\}\\
\mathcal{Q}(\epsilon(t)) &= \{\varepsilon\} \cup 1.\mathcal{Q}(t)\\
\mathcal{Q}(\lambda^\alpha x.t) &= \{\varepsilon\} \cup 1.\mathcal{Q}(t)\\
\mathcal{Q}(@^\alpha(t_1,t_2)) &= \{\varepsilon\} \cup 1.\mathcal{Q}(t_1) \cup 2.\mathcal{Q}(t_2)\\
\mathcal{Q}(f^\alpha(t_1,\dots,t_n)) &= \{\varepsilon\} \cup 0.\mathcal{Q}(\varphi^{-1}(f^\alpha)) \cup (\bigcup_i i.\mathcal{Q}(t_i))
\end{aligned}$$

The following cases define λ-lifting as a CRS reduction:

- (Reify) If $\lambda^\alpha x.p$ is a skeleton such that $\varphi(\lambda^\alpha x.p) = f^\beta$, then
$$\lambda^\alpha x.p[Z_1,\dots,Z_n] \quad \xrightarrow{\varepsilon}_{lft} \quad f^\beta(Z_1,\dots,Z_n)$$
- (Inside) If $\varphi^{-1}(f^\alpha)[x_1,\dots,x_n] \xrightarrow{q}_{lft} \varphi^{-1}(g^\beta)[x_1,\dots,x_n]$ where $x_1,\dots,x_n$ are fresh variables then
$$f^\alpha(Z_1,\dots,Z_n) \quad \xrightarrow{0.q}_{lft} \quad g^\beta(Z_1,\dots,Z_n)$$
- (Context) If $t \xrightarrow{q_1}_{lft} t'$ and c is a unary context with a hole at position q_2, then
$$c[t] \quad \xrightarrow{q_2.q_1}_{lft} \quad c[t']$$

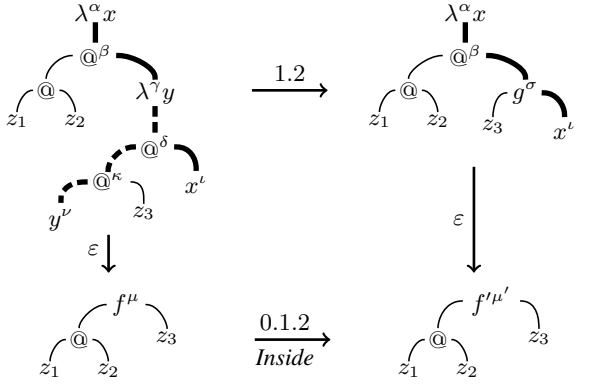
While *Reify* is the main rule of λ-lifting, *Inside* allows us to close confluence diagrams (see Example 14) and to consider λ-lifting as an orthogonal rewriting system (in the sense of [GKK05]). An *Inside* reduction can be seen as a reduction *Reify* inside a symbol. An equivalent of *Inside* naturally appears in [AF97] or [Ses97] as a reduction in the context of a *let ... in ...* construct. Please note that in the rule *Context*, c is an arbitrary context. This means that there is no particular weak restriction here.

Example 14.

Let $t = \lambda^\alpha x.@^\beta(@(z_1,z_2), \lambda^\gamma y.@^\delta(@^\kappa(y^\nu, z_3), x^\iota))$ be a labelled term, where some labels are omitted. Write

$$\begin{aligned}
\varphi^{-1}(f) &= \lambda x.@([], \lambda y.@(@(y,[]),x))\\
\mu &= \alpha(\beta(\Box, \gamma(\delta(\kappa(\nu,\Box),\iota))))\\
\varphi^{-1}(g) &= \lambda y.@(@(y,[]),[])\\
\sigma &= \gamma(\delta(\kappa(\nu,\Box),\Box))\\
\varphi^{-1}(f') &= \lambda x.@([], g([],x))\\
\mu' &= \alpha(\beta(\Box, \sigma(\Box,\iota)))
\end{aligned}$$

The dotted lines denote the skeleton of the abstraction $\lambda^\gamma y$, while the bold lines (solid or dotted) denote the skeleton of the abstraction $\lambda^\alpha x$.



Remark 5. *The reduction Reify is a straightforward use of contraction $\to_c$, and the reduction Inside can be factorized using contraction $\to_c$ and expansion $\to_e$: if $t \to_{lft} t'$ by Inside, then there are u and u' such that $t \to_e u \to_c u' \to_c t'$.*

Lemma 4 (Diamond property). *If $t \to_{lft} u$ and $t \to_{lft} v$ with $u \neq v$ then there is w such that $u \to_{lft} w$ and $v \to_{lft} w$.*

Lemma 5 (Termination). *The system $\to_{lft}$ is strongly normalizing.*

Lemma 4 implies that $\to_{lft}$ is confluent, and that all the $\to_{lft}$-sequences between two given terms have the same length [Ter03]. An immediate corollary of lemmas 4 and 5 is that any term has a unique $\to_{lft}$-normal form.

4.2 Fully-lazy λ-lifting as a bisimulation

This section proves that $\to_{lft}$ does not only transform terms but also reduction sequences, and that the transformation operates forward as well as backward. Formally, the reflexive-transitive closure of the relation $\to_{lft}$ (written $\twoheadrightarrow_{lft}$) is a bisimulation between $\to_o$ and itself. Thanks to the CRS formalism which allows reasoning on the intermediate steps of the transformation, the proof can be reduced to a one-step simulation property (Lemma 6).

Lemma 6 (One-step simulation).

- *If* $t' \ {}_o\!\leftarrow t \to_{lft} u$ *then there is* u' *such that* $t' \twoheadrightarrow_{lft} u' \ {}_{lft}\!\leftarrow u$.
- *It* $t \to_{lft} u \to_o u'$ *then there is* t' *such that* $t \to_o t' \twoheadrightarrow_{lft} u'$.

Proof. (Sketch) By case on the relative positions of the two redexes. Most cases use the stability of η and the commutation of $\mathcal{P}$ with $\to_{lft}$. The diagram is closed by $t' \twoheadrightarrow_{lft} u'$ because a $\to_o$-reduction can destroy or duplicate the considered $\to_{lft}$-redex. □

Lemma 6 yields as immediate corollary:

Theorem 3 (Bisimulation). *The relation* $\twoheadrightarrow_{lft}$ *is a bisimulation between the reduction* $\to_o$ *and itself, which means that any diagram*

$$\begin{array}{ccc} t & \overset{lft}{\longrightarrow\!\!\!\!\rightarrow} & u \\ {\scriptstyle o}\downarrow & & \\ t' & & \end{array} \quad \text{or} \quad \begin{array}{ccc} t & \overset{lft}{\longrightarrow\!\!\!\!\rightarrow} & u \\ & & \downarrow{\scriptstyle o} \\ & & u' \end{array} \quad \text{can be closed as} \quad \begin{array}{ccc} t & \overset{lft}{\longrightarrow\!\!\!\!\rightarrow} & u \\ {\scriptstyle o}\downarrow & & \downarrow{\scriptstyle o} \\ t' & \underset{lft}{\longrightarrow\!\!\!\!\rightarrow} & u' \end{array}$$

This bisimulation is a strong property for λ-lifting: it associates a progressive transformation of reduction sequences to the progressive transformation of terms. Moreover, there is a bijection between the single steps of the image and the antecedent reduction sequences. In the next section, we show that this holds also for parallel labelled reduction.

4.3 Fully-lazy λ-lifting as a graph bisimulation

For $r \in \{\beta, t, o, c, e, lft\}$, write $\overset{\alpha}{\Rightarrow}_r$ the simultaneous reduction of all the $\to_r$-redexes of label α in a term satisfying the sharing property $\mathbb{S}$. This last section shows that $\Rightarrow_{lft}$ preserves sharing: the same subterms are shared in the source and in the target of this reduction. This allows us to conclude that the full lazinesses of [Wad71] and [Jon87] are bisimilar.

In this section we use again the invariants $\mathbb{S}$ and $\mathbb{M}$ introduced in Section 2.3. In our new setting we extend $\mathbb{M}$ to any object redex. We moreover use the following three invariants:

3. Say a term t has the **independent labelling property**, written $\mathbb{I}(t)$, when all the labels of t are well-formed and there is no pair of labels (α, β) of t such that $\alpha \neq \beta$ and $\alpha \wr \beta$. This property is useful to ensure that contraction and expansion do not break the sharing property.
4. Say a term t has the **tuned skeletons property**, written $\mathbb{T}(t)$, when every skeleton p in t is of the form $[\Omega_1 ... \Omega_n, p^*]$ where p^* has only virginal labels. This property allows applying a contractor φ to any prefix, and rules out the last case of Example 11.
5. Say a term t has the **harmonious binding property**, written $\mathbb{H}(t)$, when any two variables with the same label are either both free or both bound by abstractions bearing the same label. This property strengthens the sharing property: when both are present, two whole skeletons are shared whenever one of their nodes is shared.

Finally, say a term t has the $\mathbb{SMIITH}$ property (written $\mathbb{SMIITH}(t)$) when the five properties are satisfied by t as well as by $\varphi^{-1}(f^\alpha)$ for any f^α appearing in t (or recursively in the antecedent by φ of a symbol).

Lemma 7. *For any* $r \in \{\beta, t, o, c, e, lft\}$ *and any two terms* t, t', *if* $\mathbb{SMIITH}(t)$ *and* $t \overset{\alpha_0}{\Rightarrow}_r t'$ *then* $\mathbb{SMIITH}(t')$.

Proof. (Sketch) By remarks 3 and 5 we need only consider the cases where $r \in \{\beta, c, e\}$. As in Theorem 1 the proof is by case analysis on the origin of the considered labels. □

Theorem 4 (Preservation of optimal sharing). *Let* t *be a term such that* $\mathbb{SMIITH}(t)$. *Suppose* $t \overset{\alpha_0}{\Rightarrow}_{lft} t'$. *Let* u' *(resp.* v'*) be a subterm of* t', *with ancestor* u *(resp.* v*) in* t. *Then* $\tau(u') = \tau(v')$ *if and only if* $\tau(u) = \tau(v)$.

Proof. Suppose $\tau(u') = \tau(v') = \alpha'$. Case on the origin of α':

- If $\tau(u) = \tau(v) = \alpha'$, it's over.
- If $\tau(u) = \alpha'$ and $\tau(v) \neq \alpha'$, then $\tau(v) = \alpha_0$ such that $\alpha_0 = [\Omega_1 ... \Omega_n, \alpha_0^*]$ and $\alpha' = [\Omega_1 ... \Omega_n, \alpha^*]$ with α_0^* appearing into α^*, which implies $\alpha' \wr \alpha_0$ and contradicts $\mathbb{I}(t)$.
- If $\tau(u) \neq \alpha'$ and $\tau(v) \neq \alpha'$ then $\tau(u) = \tau(v) = \alpha_0$.

Suppose $\tau(u) = \tau(v) = \alpha$. If $\alpha \neq \alpha_0$, then $\tau(u') = \tau(u) = \tau(v) = \tau(v')$. Else $u = v$, $u \overset{\alpha_0}{\Longrightarrow}_{lft} u'$, $v \overset{\alpha_0}{\Longrightarrow}_{lft} v'$ and $u' = v'$. In particular $\tau(u') = \tau(v')$. □

Finally, write $t \approx t'$ is t and t' are $\Rightarrow_{lft}$-convertible. We deduce that

$$\text{any} \ \begin{array}{ccc} t & \approx & u \\ {\scriptstyle o}\Downarrow & & \\ t' & & \end{array} \quad \text{or} \quad \begin{array}{ccc} t & \approx & u \\ & & \Downarrow{\scriptstyle o} \\ & & u' \end{array} \quad \text{can be closed as} \quad \begin{array}{ccc} t & \approx & u \\ {\scriptstyle o}\Downarrow & & \Downarrow{\scriptstyle o} \\ t' & \approx & u' \end{array}$$

This implies that the implementations of full laziness in [Wad71] and [Jon87] define the same reduction space, which also correspond to optimal sharing along [Mar91, Ter03] for the first-order system defined by the target reduction $\to_t$, and to optimal sharing along [BLM07] for the CH-weak reduction of the λ-calculus.

5. Conclusion

Sharing, and in particular fully lazy sharing, is described and implemented by different technical tools including graphs, closures, and program transformations. As a consequence, the many definitions of fully lazy sharing [Wad71, Jon87, Ses97, AF97, SW05, BLM07] are sometimes hardly comparable. Yet they all intend to implement the same basic ideas.

This paper unifies all these views of full laziness. To achieve this we define an axiomatic framework of sharing-via-labelling systems, in which the various approaches can be expressed. Then we prove that all the resulting systems are bisimilar, in the sense that they have isomorphic reduction spaces.

In particular, by linking [BLM07] to other definitions of full laziness, we confirm the intuition of its authors that fully lazy sharing gives an optimal sharing for the weak λ-calculus of [ÇH98].

Last but not least, we show that weak reduction in λ-calculus can be expressed in orthogonal first-order rewriting by means of fully lazy λ-lifting, with a one-to-one correspondence between their reduction steps. This remarkable last property makes our first-order formulation really different from the formulations that use de Bruijn indices or explicit substitutions [Tur79, Mar91]. Moreover, our transformation preserves optimal sharing and expresses fully lazy sharing as optimal sharing for the target first-order system.

5.1 Related work

Related approaches to the efficient implementation of functional programming languages include in particular the study of optimal reduction and the attempts to implement it. The study of optimal reduction [Lé80, Mar91, GK96, vO96, BLM07] traditionally uses three equivalent characterizations called labelling, extraction, and

zig-zag. This paper extends the labelling-based characterization of optimality to weak β-reduction systems.

The possibility of a straightforward implementation in graphs of the label-based characterization of optimality has for long been known to be a feature of first-order rewriting [Mar91] that did not hold in the λ-calculus. However, more recently this feature has been observed in a weak restriction of the λ-calculus [BLM07]. The present paper confirms this observation by showing that it holds in any weak β-reduction system, and explains this first-order behaviour of weak reduction by expliciting a link between weak β-reduction systems and first-order rewriting.

Two kinds of implementations are known to perform less shared β-reduction steps than some fully lazy implementations. In particular partial evaluation [HG91] has been compared to [Jon87], and optimality [Lé80] is born as an idealization of [Wad71]. Hence this paper shows that partial evaluation [HG91] as well as any implementation of optimality [AG98, PQ07] is able to perform less shared β-reduction steps than any implementation of full laziness.

On the other hand, a global comparison between all these approaches is still missing. In particular, the following facts keep the question open for now: the number of shared β-reduction steps in a fully lazy system is polynomially related to the actual cost of performing the reduction on a Turing machine (simple extension of [LM09]), but this does not hold with optimal sharing [AM98].

5.2 Future work

Fully lazy λ-lifting is shown to be a powerful tool to give a faithful first-order account of higher-order systems. This phenomenon prompts us to carry on investigations in at least two directions.

- Generalization of λ-lifting to any weak β-reduction system satisfying our axioms. Two interesting and challenging examples would be the plain λ-lifting of [Joh85], in which the prefixes are not stable by reduction, and a new notion of λ-lifting based on $\mathcal{P}_{blm}$, which would turn the spines into supercombinators. Since the spines can bind variables in their holes, some function symbols would also bind some variables in their arguments.
- Generalization of sharing and λ-lifting to higher-order rewriting, which includes richer systems that rewrite functions, such as proof assistants and compilers. As far as the author is aware, general higher-order rewriting knows no notion of weak reduction, sharing-via-labelling, or λ-lifting. However, the present work seems to be abstract enough to be generalized to higher-order frameworks such as combinatory reduction systems.

Acknowledgments

The author would like to thank Delia Kesner for her continuous support, Vincent van Oostrom for several discussions and for a mysterious but insightful remark that triggered this investigation, and Roberto di Cosmo, Olivier Danvy, John Field and the anonymous reviewers for numerous helpful comments and suggestions.

References

[AF97] Z.M. Ariola and M. Felleisen. The call-by-need lambda calculus. *J. Funct. Program.*, 7(3):265–301, 1997.

[AFM+95] Z.M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. In *POPL*, pages 233–246, 1995.

[AG98] A. Asperti and S. Guerrini. *The optimal implementation of functional programming languages*. Cambridge University Press, 1998.

[AM98] A. Asperti and H.G. Mairson. Parallel Beta Reduction is not Elementary Recursive. In *POPL*, pages 303-315, 1998.

[Bal11] T. Balabonski. A Unified Approach to Fully Lazy Sharing. `http://hal.archives-ouvertes.fr/hal-00637048/`. Rapport technique PPS, 2011.

[BKKS87] H.P. Barendregt, R. Kennaway, J-W. Klop, and M. Ronan Sleep. Needed reduction and spine strategies for the lambda calculus. *Inf. Comput.*, 75(3):191–231, 1987.

[BLM07] T. Blanc, J.-J. Lévy, and L. Maranget. Sharing in the Weak Lambda-Calculus Revisited. In *Reflections on Type Theory, Lambda Calculus and the Mind*, 2007.

[Bru03] H.J.S. Bruggink. Residuals in higher-order rewriting. In *RTA*, pages 123–137, 2003.

[ÇH98] N. Çağman and J. R. Hindley. Combinatory weak reduction in lambda calculus. *TCS*, 198(1-2):239–247, 1998.

[DMMZ10] O. Danvy, K. Millikin, J. Munk, and I. Zerny. Defunctionalized Interpreters for Call-by-Need Evaluation. In *FLOPS*, pages 240–256, 2010.

[DLLL05] D. Dougherty, P. Lescanne, L. Liquori, and F. Lang. Addressed Term Rewriting Systems: Syntax, Semantics, and Pragmatics: Extended Abstract. *ENTCS*, 127(5):57–82, 2005.

[GK96] J. Glauert and Z. Khasidashvili. Relative normalization in deterministic residual structures. In *CAAP*, pages 180–195, 1996.

[GKK05] J.R.W. Glauert, D. Kesner, and Z. Khasidashvili. Expression reduction systems and extensions: An overview. In *Processes, Terms and Cycles*, pages 496–553, 2005.

[HG91] C.K. Holst and D.K. Gomard. Partial evaluation is fuller laziness. In *PEPM*, pages 223–233, 1991.

[Hug82] R.J.M. Hughes. Super combinators: A new implementation method for applicative languages. In *LFP*, pages 1–10, 1982.

[Joh85] T. Johnsson. Lambda lifting: Transforming programs to recursive equations. In *FPCA*, pages 190–203, 1985.

[Jon87] S. Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice-Hall, Inc., 1987.

[KvOvR93] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theor. Comput. Sci.*, 121(1&2):279–308, 1993.

[Lam90] J. Lamping. An algorithm for optimal lambda calculus reduction. In *POPL*, pages 16–30, 1990.

[Lau93] J. Launchbury. A natural semantics for lazy evaluation. In *POPL*, pages 144–154, 1993.

[LM09] U. Dal Lago and S. Martini. On constructor rewrite systems and the lambda-calculus. In *ICALP (2)*, pages 163–174, 2009.

[Lé80] J.-J. 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] L. Maranget. Optimal Derivations in Weak Lambda-calculi and in Orthogonal Terms Rewriting Systems. In *POPL*, pages 255–269, 1991.

[PQ07] M. Pedicini and F. Quaglia. Pelcr: Parallel environment for optimal lambda-calculus reduction. *ACM Trans. Comput. Logic*, 8, July 2007.

[Ses97] P. Sestoft. Deriving a lazy abstract machine. *J. Funct. Program.*, 7(3):231–264, 1997.

[SW05] O. Shivers and M. Wand. Bottom-up β-reduction: Uplinks and λ-DAGs. In *ESOP*, pages 217–232, 2005.

[Ter03] Terese. *Term Rewriting Systems*. Cambridge Univ.Press, 2003.

[Tur79] D.A. Turner. A new implementation technique for applicative languages. In *Softw., Pract. Exper.*, 9(1):31–49, 1979.

[vO96] V. van Oostrom. Higher-order families. In *RTA*, pages 392–407, 1996.

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

[Yos94] N. Yoshida. Optimal reduction in weak-λ-calculus with shared environments. *J. of Computer Software*, 11(5):2–20, 1994.