# Sharing in the weak lambda-calculus

Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget

INRIA - Rocquencourt,
`Tomasz.Blanc@inria.fr`
`Jean-Jacques.Levy@inria.fr`
`Luc.Maranget@inria.fr`,
`http://moscova.inria.fr/`∼{`tblanc,levy,maranget`}

**Abstract.** Despite decades of research in the λ-calculus, the syntactic properties of the weak λ-calculus did not receive great attention. However, this theory is more relevant for the implementation of programming languages than the usual theory of the strong λ-calculus. In fact, the frameworks of weak explicit substitutions, or computational monads, or λ-calculus with a `let` statement, or super-combinators, were developed for adhoc purposes related to programming language implementation. In this paper, we concentrate on sharing of subterms in a confluent variant of the weak λ-calculus. We introduce a labeling of this calculus that expresses a confluent theory of reductions with sharing, independent of the reduction strategy. We finally state that Wadsworth's evaluation technique with sharing of subterms corresponds to our formal setting.

## 1 Introduction

In the terminology of the λ-calculus, a *strong* calculus validates the following ξ-rule

$$(\xi)\ \frac{M \to N}{\lambda x.M \to \lambda x.N}$$

A *weak* calculus does not validate this rule. One easily shows that the weak λ-calculus is not confluent. In [18], an extension of the weak λ-calculus was introduced. It is strongly inspired from the one of Çağman and Hindley [6] for Combinatory Logic. In this calculus, a restricted version of the ξ-rule is valid; this new ξ′-rule is intuitively defined by

$$(\xi')\ \frac{M \xrightarrow{R} N \quad x \notin R}{\lambda x.M \xrightarrow{R} \lambda x.N}$$

meaning that the ξ-rule is valid when the bound variable $x$ is not free in the redex $R$ contracted between $M$ and $N$ (This rule will be presented in section 2 in a form slightly different from — but equivalent to — the σ-rule used in [18]). The resulting new weak λ-calculus is confluent as shown in [18].

The theory of optimal reductions in the λ-calculus [5] has been extensively studied by Abadi, Asperti, Coppola, Gonthier, Guerrini, Lamping, Lawall, Lévy, Mairson, Martini et al [3, 4, 9, 13, 15, 17]. These authors represent $\lambda$-terms as graphs with shared subcontexts. For instance, in $(\lambda x.xa(xb))(\lambda y.Iy)$ where $I = \lambda x.x$, it is necessary to share the redex $Iy$ independently of the value of $y$. Therefore the subcontext $I[\,]$ has to be shared. But after reduction of the external redex, we get $(\lambda y.Iy)a((\lambda y.Iy)b)$ and further $Ia((\lambda y.Iy)b)$, where the shared subcontext $I[\ ]$ is instantiated with two different terms. Technically, in these graphs, a shared subcontext can be referenced through a *fan-in* node to multiplex incoming arcs from terms using it; and context holes are filled through *fan-out* nodes to demultiplex outgoing arcs pointing to terms filling these holes. For instance, Lamping's graphs [13] operate on fans and *brackets*, which can be decomposed into more elementary fans, brackets and *croissants* obeying to reduction rules directed by the *context semantics* defined in [9].

In the weak $\lambda$-calculus, reductions are not performed under $\lambda$-abstractions. In the above example, the subterm $Iy$ in $(\lambda x.xa(xb))(\lambda y.Iy)$ cannot be reduced since inside the abstraction $\lambda y.Iy$. Thus the subcontext $I[\,]$ needs not be shared. The $\lambda$-terms with sharing can be represented by directed acyclic graphs (*dags*) instead of the (cyclic) Lamping's graph structures required to implement shared subcontexts. In this paper, we present a weak labeled $\lambda$-calculus that expresses a confluent theory of sharing within the weak $\lambda$-calculus corresponding to the shared evaluation strategy by Wadsworth [25] defined with dags in 1971!

The weak $\lambda$-calculus corresponds to runtime systems in functional languages, since runtimes just pass arguments to functions and never compute function bodies, i.e. under $\lambda$-abstractions. At compile-time, inlining or partial evaluation are feasible; but the weak $\lambda$-calculus just corresponds to the execution phase. However a runtime of a functional language usually implements a particular reduction strategy such as call-by-name, call-by-need or call-by-value. We prefer to model these runtimes by a confluent calculus which allows to consider mixed strategies alternating call-by-need and call-by-value. Moreover, a confluent calculus makes independent sharing and reduction strategy, which are two independent concepts. In runtimes of lazy functional languages (Haskell, LML, G-machine) [20, 22], the call-by-need strategy will correspond to a leftmost outermost reduction with sharing in the weak $\lambda$-calculus.

The runtime of a functional program [16] often computes a functional closure, i.e. a pair of a $\lambda$-abstraction (program) and a substitution (environment). The theory of explicit substitutions is related to the notion of closure but does not restrict reduction strategies [1]. This theory is not simple. It uses de Bruijn indices and is not confluent for open terms: Klop's counterexample [11] for surjective pairing can be adapted to the calculus of explicit substitutions [1]. However, confluence (on open terms) can be recovered at the price of either considering a much more complex calculus of explicit substitutions [7], or a theory of weak explicit substitutions as in [8, 18]. In the latter case, a theory of sharing was sketched, through the definition of a weak labeled calculus of explicit substitutions.

Closures and explicit substitutions are not necessary to express sharing of $\lambda$-terms. For instance, call-by-need strategies by Launchbury, Odersky or Ariola et al. [14, 19, 2] use a $\lambda$-calculus with a new `let` construct. However, these calculi have often critical pairs (in the sense of term rewriting) and extra rules to handle the `let` construct. Term rewriting systems (TRS) can also be used by lifting free variables and transforming each $\lambda$-abstraction into the application of several of its free variables to a (super)combinator. In this paper we want to stay with the classical set of $\lambda$-terms and subtheories of the classical $\lambda$-calculus.

In Section 2, we recall the definitions and basic properties of the weak $\lambda$-calculus introduced in [18]. In Section 3, we consider the weak labeled $\lambda$-calculus and prove its confluence. In Section 4, we relate labels and sharing. In Section 5, we discuss several differences between the weak labeled $\lambda$-calculus and dag implementations. In Section 6, we comment on the relation between TRSs and our formal setting. We conclude in Section 7.

## 2 The weak $\lambda$-calculus

The weak $\lambda$-calculus is defined in [18]. As usual, the set of $\lambda$-terms is defined by

$$M, N ::= x \mid MN \mid \lambda x.M$$

and $\beta$-reduction is

$$(\beta) \qquad (\lambda x.M)N \to M[\![x\backslash N]\!]$$

where $M[\![x\backslash N]\!]$ is recursively defined by

$$\begin{aligned} x[\![x\backslash P]\!] &= P \\ y[\![x\backslash P]\!] &= y \quad (x \neq y) \\ (MN)[\![x\backslash P]\!] &= M[\![x\backslash P]\!]\ N[\![x\backslash P]\!] \\ (\lambda y.M)[\![x\backslash P]\!] &= \lambda y.M[\![x\backslash P]\!] \quad (x \neq y,\ y \text{ not free in } P) \end{aligned}$$

We keep $\alpha$-conversion implicit, and freely use renaming of bound variables. In the last case, the substitution must not bind free variables in $P$. Equality on terms is defined up to the renaming of bound variables ($\alpha$-conversion). In the (strong) $\lambda$-calculus, every context is active, since any subterm may be reduced at any time. But in the weak $\lambda$-calculus, without the $\xi$-rule, no reduction inside a $\lambda$-abstraction occurs. The Church-Rosser property (confluence) does not hold in the weak $\lambda$-calculus : when $N \to N'$ we have

$$\begin{tikzcd} (\lambda x.\lambda y.M)N \arrow[r] \arrow[d] & (\lambda x.\lambda y.M)N' \arrow[d] \\ (\lambda y.M[\![x\backslash N]\!]) & (\lambda y.M[\![x\backslash N']\!]) \end{tikzcd}$$

The term $(\lambda y.M[\![x\backslash N]\!])$ is in normal form and cannot be reduced, and the previous diagram does not commute. The problem has been known for a long time in combinatory logic [10], although often kept as a “folk theorem”. In [6], it is specifically stated, and shown as being relevant when translating the $\lambda$-calculus into combinatory logic. In [18], it is proved that confluence is recovered by adding the new inference rule

$$(\sigma)\ \frac{N \to N'}{M[\![x\backslash N]\!] \to M[\![x\backslash N']\!]}\ (M \text{ linear in } x)$$

where the free variable $M$ is linear in $x$ if $x$ has exactly one occurrence in $M$. The use of substitution aims at reflecting the fact that $N$ does not contain variables bound outside $N$. Intuitively, it means that $N$ does not depend on an enclosing $\lambda$-abstraction or its argument. Therefore $N$ may be reduced in such a context. The linearity condition avoids to consider parallel reduction steps.

Here, instead of the $\sigma$-rule, we use a related, more direct approach to recover confluence in the weak $\lambda$-calculus. Let us write $x \notin M$, when $x$ is not a free variable in $M$. Formally:

$$\frac{}{x \notin y}\ (x \neq y) \qquad \frac{x \notin M}{x \notin \lambda y.M} \qquad \frac{x \notin M \quad x \notin N}{x \notin MN}$$

Then our alternative presentation of the weak $\lambda$-calculus adds a new $\xi'$-rule to the classical $\mu$ and $\nu$-rules.

$$(\beta)\ \ R = (\lambda x.M)N \xrightarrow{R} M[\![x\backslash N]\!] \qquad (\nu)\ \frac{M \xrightarrow{R} M'}{MN \xrightarrow{R} M'N} \qquad (w)\ \frac{M \xrightarrow{R} N}{M \to N}$$

$$(\xi')\ \frac{M \xrightarrow{R} M' \quad x \notin R}{\lambda x.M \xrightarrow{R} \lambda x.M'} \qquad (\mu)\ \frac{N \xrightarrow{R} N'}{MN \xrightarrow{R} MN'}$$

The reduction step relation $\xrightarrow{R}$ is annotated with the contracted redex $R$. A $\lambda$-abstraction to the left of the reduction step relation cannot bind a variable in $R$. Therefore $\alpha$-conversion does not change the redex annotating the reduction step relation. Like the $\sigma$-rule, the $\xi'$-rule allows the reduction of a subterm located under a $\lambda$-abstraction if the contracted redex does not contain a variable bound by the $\lambda$-abstraction. Again, we write $\twoheadrightarrow$ for the transitive and reflexive closure of $\to$. So $M \twoheadrightarrow N$ iff $M$ can reduce in several steps (maybe none) to $N$.

**Lemma 1.** *In the weak $\lambda$-calculus, one has*

*(i)* $N \to N' \Rightarrow M[\![x\backslash N]\!] \twoheadrightarrow M[\![x\backslash N']\!]$
*(ii)* $M \to M' \Rightarrow M[\![x\backslash N]\!] \to M'[\![x\backslash N]\!]$
*(iii)* $M[\![x\backslash N]\!][\![y\backslash N']\!] = M[\![y\backslash N']\!][\![x\backslash N[\![y\backslash N']\!]]\!] \quad (x \notin N')$

**Theorem 1.** *The weak $\lambda$-calculus is confluent.*

Proof: Straightforward application of previous lemma. □

In comparison with the strong $\lambda$-calculus, there is an additional way of creating a redex. After contracting $M = (\lambda x.Ix)y$, we obtain a redex $Iy$ which is neither a residual of a redex of $M$ nor a created redex of the strong $\lambda$-calculus. In fact, although the subterm $Ix$ in $M$ is a redex for the strong $\lambda$-calculus, it is not a redex for the weak $\lambda$-calculus: it is *frozen* in $M$ by the occurrence of $x$. The contraction of the enclosing $\lambda$-abstraction *activates* $Iy$.

The weak $\lambda$-calculus enjoys syntactic properties simpler than the strong $\lambda$-calculus. In the weak $\lambda$-calculus, the finite developments theorem [5] is easy to prove, since residuals of disjoint redexes cannot be nested. In the strong $\lambda$-calculus, residuals of two disjoint redexes may be nested. The typical example is: $M = (\lambda x.Ix)(Jy)$ with $I = J = \lambda x.x$. Then, the two disjoint $Ix$ and $Jy$ redexes have nested residuals:

$$(\lambda x.Ix)(Jy) \to I(Jy)$$

But in the weak $\lambda$-calculus this case does not occur since the subterm $Ix$ in $M$ contains the bound variable $x$ and is not considered as a redex of the weak $\lambda$-calculus. In this calculus, residuals of disjoint redexes are disjoint redexes.

This proposition allows now to state another interesting theorem of the weak $\lambda$-calculus, namely Curry's standardization theorem. A standard reduction is usually defined as a reduction contracting redexes in an outside-in and left-to-right way. Precisely a reduction of the form

$$M = M_0 \to M_1 \to \dots M_n = N \quad (n \geq 0)$$

is standard when for all $i$ and $j$ such that $0 \leq i < j < n$, the $R_j$-redex contracted at step $j$ in $M_{j-1}$ is not a residual of a redex external to or to the left of the $R_i$-redex contracted at step $i$ in $M_{i-1}$. We write $M \underset{\text{st}}{\twoheadrightarrow} N$ for the existence of a standard reduction from $M$ to $N$. Notice that the leftmost outermost reduction is a standard reduction (in the usual $\lambda$-calculus), but standard reductions may be more general.

**Theorem 2.** *If $M \twoheadrightarrow M'$, then $M \underset{st}{\twoheadrightarrow} M'$.*

Proof: One follows the proof scheme in [11] or checks the axioms of [21]. The basic step of the proof follows from the observation that when $M \xrightarrow{R} M'$ and $S'$ is a residual of $S$ (in the usual sense of the strong $\lambda$-calculus) in $M$ not inside $R$, then $S$ is a redex of the weak $\lambda$-calculus. Then assume that $M \to M' \to N'$ by contracting $R$ in $M$ and $S'$ in $M'$. Suppose $R$ and $S'$ are not in the standard ordering. Then $S'$ is residual of a redex $S$ in $M$ to the left of or outside $R$. Thus, we know that $S$ is a redex of the weak $\lambda$-calculus and we may contract it getting $N$. By confluence (actually by finite developments), we converge to $N'$ (by a finite development of the residuals of $R$ in $N$). $\square$

A normal form is a term without redex. For instance $I = \lambda x.x$ or $\lambda y.Iy$ are normal forms. Let $\xrightarrow[\text{norm}]{}\!\!\!\!\!\twoheadrightarrow$ be the reduction contracting, at each step, the leftmost outermost redex.

**Theorem 3.** *If $M \twoheadrightarrow N$ and $N$ is a normal form, then $M \underset{norm}{\twoheadrightarrow} N$.*

Proof: By persistence of the leftmost outermost redex and the use of the previous theorem. $\square$

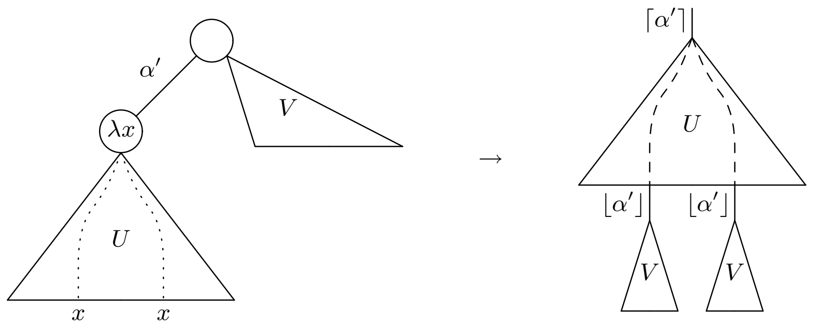

**Fig. 1.** A reduction step in the weak labeled $\lambda$-calculus (dotted lines represent paths from the root of $U$ to occurrences of the free variable $x$ in $U$; dashed lines represent the paths on which diffusion operates).

## 3 The weak labeled $\lambda$-calculus

In this section, a calculus for sharing in the weak $\lambda$-calculus is developed with the help of a confluent, labeled calculus. Labels are used to name subterms; subterms which are copies along reductions keep their labels, but new subterms must have new labels. We want the weak labeled $\lambda$-calculus to be confluent, as sharing and reduction strategy are independent concepts. Therefore an adequate naming scheme should be invariant through reductions equivalent by permutations of reduction steps. The labeled calculus is different from the one used in the labeled calculus of weak explicit substitutions [18], since we have no longer closures and explicit substitutions, and we have to take care of variable binders.

We base our labeling scheme on the labeling of the strong $\lambda$-calculus [17].

| | | |
|---|---|---|
| $U, V ::= \alpha : X$ | | labeled term |
| $X, Y ::= S \mid U$ | | clipped or labeled term |
| $S, T ::= x \mid UV \mid \lambda x.U$ | | clipped term |
| $\alpha, \beta ::= a \mid \lceil\alpha'\rceil \mid \lfloor\alpha'\rfloor \mid [\alpha', \beta] \mid \langle\alpha', \beta\rangle$ | | labels |
| $\alpha', \beta' ::= \alpha_1\alpha_2\cdots\alpha_n \quad (n > 0)$ | | compound labels |

The labeled term $\alpha : X$ is said to have label $\alpha$. Labels can be stacked as in $\alpha_1 : \alpha_2 : \cdots \alpha_n : X$. Compound labels, used in the definition of the $\ell$-reduction, are sequences of labels. An (atomic) label can be a simple letter, or formed by overlining $\lceil\alpha'\rceil$, or underlining $\lfloor\alpha'\rfloor$; it can also be a pair $[\alpha', \beta]$ or $\langle\alpha', \beta\rangle$ of compound label $\alpha'$ and label $\beta$. In the pair with square brackets, we say that $\alpha'$ *tags* $\beta$; for angle brackets, then $\alpha'$ *marks* $\beta$.

The labeled reduction $\ell$-rule is defined as

$$(\ell) \qquad R = (\alpha' \cdot \lambda x.U)V \overset{R}{\rightarrow} \lceil\alpha'\rceil : (\alpha' \textcircled{x} U)[\![x \setminus \lfloor\alpha'\rfloor : V]\!]$$

where

$$\alpha_1\alpha_2\cdots\alpha_n \cdot S = \alpha_1 : \alpha_2 : \cdots \alpha_n : S$$

The *name* of $R$ is $\alpha'$; we write $\mathit{name}(R) = \alpha'$. We assume that substitution has a higher precedence than labeling. Hence $\lceil\alpha'\rceil : U[\![x\backslash V]\!]$ is read as $\lceil\alpha'\rceil : (U[\![x\backslash V]\!])$. As in the strong labeled $\lambda$-calculus, we sandwich the body of the function part of the redex with its name overlined and underlined as shown in figure 1. The diffusion $\alpha' \textcircled{x}\, U$ creates new labels along paths from the root of $U$ to free occurrences of $x$, as illustrated in figure 1. Therefore new labels appeared for every subterm of $U$ containing a free occurrence of $x$. Formally substitution and diffusion are defined as follows:

$$
\begin{aligned}
x[\![x\backslash W]\!] &= W \\
y[\![x\backslash W]\!] &= y \\
(UV)[\![x\backslash W]\!] &= U[\![x\backslash W]\!]\, V[\![x\backslash W]\!] \\
(\lambda y.U)[\![x\backslash W]\!] &= \lambda y.U[\![x\backslash W]\!] \\
(\beta : X)[\![x\backslash W]\!] &= \beta : X[\![x\backslash W]\!] \\[6pt]
\alpha' \textcircled{x} X &= X \quad \text{if } x \notin X \\
\alpha' \textcircled{x} x &= x \\
\alpha' \textcircled{x} UV &= (\alpha' \textcircled{x} U \ \ \alpha' \textcircled{x} V) \quad \text{if } x \in U \\
\alpha' \textcircled{x} UV &= (\langle \alpha', U\rangle \ \ \alpha' \textcircled{x} V) \quad \text{if } x \notin U \text{ and } x \in V \\
\alpha' \textcircled{x} \lambda y.U &= \lambda y.\, \alpha' \textcircled{x} U \quad \text{if } x \in \lambda y.U \\
\alpha' \textcircled{x} \beta : X &= [\alpha', \beta] : \alpha' \textcircled{x} X \quad \text{if } x \in X \\[6pt]
\langle \alpha', \beta : X\rangle &= \langle \alpha', \beta\rangle : X
\end{aligned}
$$

An example of reduction is illustrated in figure 2.

During diffusion, we mark the left subterm of an application when it does not contain the bound variable $x$, since the application may become a redex of which we want to keep the history. Take for instance, the unlabeled $(\lambda x.Ix)y$, where $I = \lambda u.u$. Then $Ix$ is not a redex in $(\lambda x.Ix)y$ since its argument contains $x$, but it becomes the redex $Iy$ after contracting the enclosing redex. We want to mark the name of redex $Iy$ with the name of the enclosing redex that activated it. In the weak labeled $\lambda$-calculus, this example becomes:

$$
\begin{aligned}
& a : (b : (\lambda x.\, c : (d : \lambda u.\, e : u)\ f : x)\ g : y) \\
\rightarrow\ \ & a : \lceil b\rceil : [b, c] : (\langle b, d\rangle : (\lambda u.e : u)\ \ [b, f] : \lfloor b\rfloor : g : y)
\end{aligned}
$$

where the name $\langle b, d\rangle$ of $Iy$ contains the name $b$ of $(\lambda x.Ix)y$, whereas if the diffusion did not mark the left part $I$ of the application $Ix$, the name $d$ of $Iy$ would not have contained the name $b$ of the redex contracted to create $Iy$.

As for the theory of the strong labeled $\lambda$-calculus, the label of a labeled subterm reflects its history. A simple letter stands for a labeled term existing in the initial term (empty history). Overlining $\lceil\alpha'\rceil$, underlining $\lfloor\alpha'\rfloor$, marks $\langle\alpha', \beta\rangle$ and tags $[\alpha', \beta]$ indicate past contraction of a redex of name $\alpha'$. Overlined and underlined labels are kept to relate the weak labeled $\lambda$-calculus with the strong case.

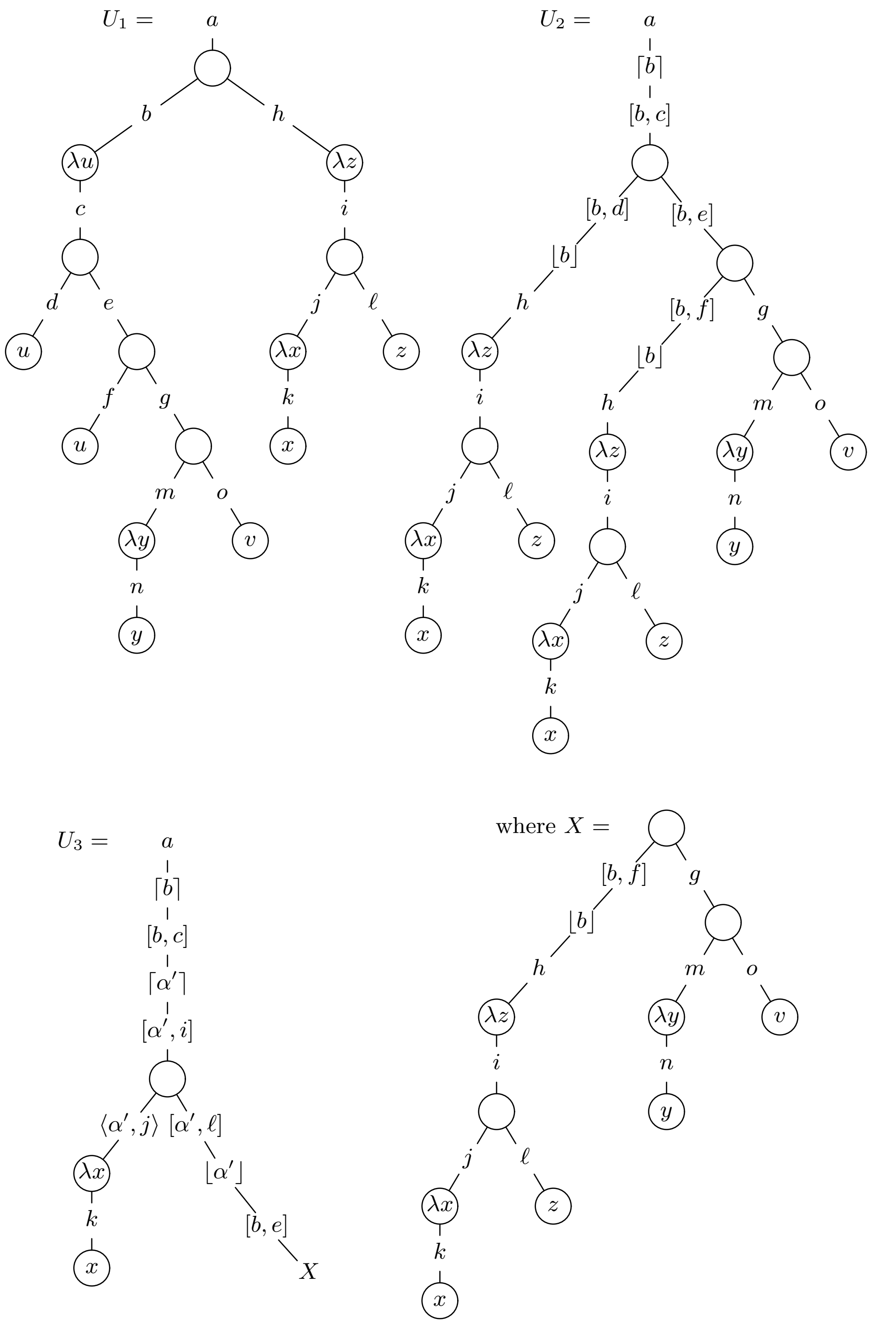

**Fig. 2.** An example of reduction in the weak labeled $\lambda$-calculus where $U_1 \to U_2 \to U_3$ and $\alpha' = [b, d]\lfloor b \rfloor h$. (The redex $(\lambda y.y)v$ could also be contracted in $U_1$, $U_2$ and $U_3$; however $(\lambda x.x)z$ with name $j$ is not a redex in $U_1$ and $U_2$; but its "residual" in $U_3$ is a created redex in $U_3$, therefore its name is marked by the name $\alpha'$ of the contracted redex.)

Free variables are defined as in the unlabeled $\lambda$-calculus. The context rules are defined as follows for labeled reduction:

$$(\nu)\ \frac{U \xrightarrow{R} U'}{UV \xrightarrow{R} U'V} \qquad (\lambda)\ \frac{X \xrightarrow{R} X'}{\alpha : X \xrightarrow{R} \alpha : X'} \qquad (w)\ \frac{X \xrightarrow{R} X'}{X \to X'}$$

$$(\mu)\ \frac{V \xrightarrow{R} V'}{UV \xrightarrow{R} UV'} \qquad (\xi')\ \frac{U \xrightarrow{R} U' \quad x \notin R}{\lambda x.U \xrightarrow{R} \lambda x.U'}$$

We also write $\twoheadrightarrow$ for the transitive and reflexive closure of $\to$.

**Lemma 2.** *If $X \xrightarrow{R} X'$ and $x \notin R$, then $\alpha' ⓧ X \to \alpha' ⓧ X'$*

Proof: We first remark that as $x \notin R$ and $X \xrightarrow{R} X'$, we have $x \in X$ if and only if $x \in X'$. Clearly, $x \notin X$ implies $x \notin X'$ since the set of free variables of the contractum of a redex is a subset of the free variables of the contracted redex. Conversely, a free variable disappears in the contractum iff the redex has all occurrences of this free variable in its argument and the redex erases its argument. But as $x \notin R$, this case is not possible.

Now we proceed by cases on the definition of diffusion and by induction on the size of $X$:

1. If $x \notin X$, then $x \notin X'$ and the lemma is obvious, since $\alpha' ⓧ X = X$ and $\alpha' ⓧ X' = X'$.
2. If $x \in X$, we have again several cases.
   (a) Let $R$ be in $U$ and $X = UV \to U'V = X'$.
      i. If we have $x \in U$, one gets by induction $\alpha' ⓧ U \to \alpha' ⓧ U'$. Then the first remark yields $x \in U'$. And, by the definition of diffusion, we have $\alpha' ⓧ X = (\alpha' ⓧ U\ \alpha' ⓧ V)$ and $\alpha' ⓧ X' = (\alpha' ⓧ U'\ \alpha' ⓧ V)$. By the $\nu$-rule, we obtain $\alpha' ⓧ X \to \alpha' ⓧ X'$.
      ii. If we have $x \notin U$, the first remark yields $x \notin U'$. Then, we have $\alpha' ⓧ X = (\langle \alpha', U\rangle\ \alpha' ⓧ V)$ and $\alpha' ⓧ X' = (\langle \alpha', U'\rangle\ \alpha' ⓧ V)$. By the $\lambda$-rule for context, we get $\langle \alpha', U\rangle \to \langle \alpha', U'\rangle$, and therefore, by the $\nu$-rule $\alpha' ⓧ X \to \alpha' ⓧ X'$.
   (b) If $R$ is in $V$, we have a simpler but similar argument.
   (c) The other cases when $X = \lambda y.U$ and $X = \beta : Y$ are similar.

□

**Lemma 3.** *If $U \to U'$ and $X \to X'$, then $X[\![x\backslash U]\!] \twoheadrightarrow X[\![x\backslash U']\!]$ and $X[\![x\backslash U]\!] \to X'[\![x\backslash U]\!]$.*

Proof: Straightforward by induction on the size of $X$. □

**Theorem 4.** *The weak labeled $\lambda$-calculus is confluent.*

Proof: By the Tait–Martin-Lof method, see [5]. The only interesting cases are the two commuting diagrams, when $x \notin R$, and $U \xrightarrow{R} U'$, $V \to V'$:

$$\begin{tikzcd}
(\alpha' \cdot \lambda x.U)V \arrow[r, "R"] \arrow[d] & (\alpha' \cdot \lambda x.U')V \arrow[d] \\
\lceil \alpha' \rceil : (\alpha' \textcircled{$x$} U)\llbracket x \backslash \lfloor \alpha' \rfloor : V \rrbracket \arrow[r] & \lceil \alpha' \rceil : (\alpha' \textcircled{$x$} U')\llbracket x \backslash \lfloor \alpha' \rfloor : V \rrbracket
\end{tikzcd}$$

$$\begin{tikzcd}
(\alpha' \cdot \lambda x.U)V \arrow[r] \arrow[d] & (\alpha' \cdot \lambda x.U)V' \arrow[d] \\
\lceil \alpha' \rceil : (\alpha' \textcircled{$x$} U)\llbracket x \backslash \lfloor \alpha' \rfloor : V \rrbracket \arrow[r, two heads] & \lceil \alpha' \rceil : (\alpha' \textcircled{$x$} U)\llbracket x \backslash \lfloor \alpha' \rfloor : V' \rrbracket
\end{tikzcd}$$

which can be proved with the help of previous lemmas. □

## 4 Sharing of subterms

In the labeled $\lambda$-calculus, the name of a redex records its history. Namely if a redex $S$ is the residual of a redex $R$, their names are equal; and if a redex $S$ is created by the contraction of a redex $R$, the name of $R$ is contained in the name of $S$. We formalize this central property of the weak labeled $\lambda$-calculus with the following $\prec$ relation. We say that $\alpha'$ is strictly smaller than $\beta'$, we write $\alpha' \prec \beta'$, for the following cases:

$$\alpha' \prec \lceil \alpha' \rceil \qquad \alpha' \prec \lfloor \alpha' \rfloor \qquad \alpha' \prec [\alpha', \beta] \qquad \alpha' \prec \langle \alpha', \beta \rangle$$
$$\alpha' \prec \beta_i \;\Rightarrow\; \alpha' \prec \beta_1 \cdots \beta_n \qquad \alpha' \prec \beta' \prec \gamma' \;\Rightarrow\; \alpha' \prec \gamma'$$

Hence the name $\alpha'$ of a redex is smaller than any label where it is overlined or underlined. It is also smaller than a pair of which it is the first component. If it is smaller than some $\beta_i$, it is smaller than a word for which $\beta_i$ is a subcomponent. Moreover we close this relation by transitivity. This relation is clearly a strict ordering. This relation expresses the intuitive interpretation of labels we gave when defining labels: $\alpha' \prec \beta'$ if the contraction of a redex of name $\alpha'$ participates to the "creation" of $\beta'$. We recall that a reduction $X \twoheadrightarrow Y$ creates redex $S$ in $Y$ if $S$ is not a residual (along this reduction) of a redex $R$ in $X$.

**Lemma 4.** *If $X \xrightarrow{R} Y$ and redex $S$ in $Y$ is created in this reduction step, then $name(R) \prec name(S)$.*

Proof: Straightforward by case inspection. □

The goal of this section is to prove that two subterms with the same label are equal in the weak labeled $\lambda$-calculus, provided that we start reductions from a term with distinct letters on its subterms. This will mean that two subterms labeled with the same label can be shared in a dag representation of terms. To prove this property, we first show that several invariants are preserved by reductions.

**Invariant 1** *$\mathcal{Q}(W)$ holds iff we have $\alpha' \not\prec \beta$ for every redex $R$ with name $\alpha'$ and any subterm $\beta : X$ in $W$.*

A *complete labeled reduction* step $U \overset{\alpha'}{\Longrightarrow} V$ is the finite development of all redexes with name $\alpha$ in $U$. We first prove that invariant $\mathcal{Q}$ is preserved by complete labeled reduction steps. Intuitively, invariant $\mathcal{Q}$ means that the names of redexes are maximal. It guarantees that when performing complete labeled reduction steps, the names of the contracted redexes are all distinct.

**Lemma 5.** *If $\mathcal{Q}(W)$ and $W \overset{\gamma'}{\Longrightarrow} W'$, then $\mathcal{Q}(W')$.*

Proof: Let $R$ with name $\alpha'$ be a redex in $W'$. Let $\beta : U$ be a subterm of $W'$ such that $\alpha' \prec \beta$.

1. $R$ is a residual of a redex $R'$ of $W$. Then $\mathit{name}(R) = \mathit{name}(R') = \alpha'$.
   (a) $\beta$ is a label in $W$. Impossible since $\mathcal{Q}(W)$.
   (b) $\beta$ is created by the reduction from $W$ to $W'$.
   By cases on the label's creation:
      i. $\beta = \lceil \gamma' \rceil$. Then $\alpha' \prec \lceil \gamma' \rceil$. There are again two cases:
         A. $\alpha' = \gamma'$. Impossible by definition of $\overset{\gamma'}{\Longrightarrow}$.
         B. $\alpha' \prec \gamma'$. If $\gamma' = \gamma_1 \dots \gamma_n$, there exists $\gamma_i$ such that $\alpha' \prec \gamma_i$. Therefore there is a subterm $\gamma_i : V$ in $W$ with $\alpha' \prec \gamma_i$. This contradicts $\mathcal{Q}(W)$.
      ii. $\beta = \lfloor \gamma' \rfloor$ or $\beta = [\gamma', \beta_1]$ or $\beta = \langle \gamma', \beta_1 \rangle$. These cases are similar to the previous one.
2. $R$ is created by the reduction from $W$ to $W'$. Then $\gamma' \prec \alpha' \prec \beta$.
   (a) $\beta$ is a label in $W$. We have $\gamma' \prec \beta$. Impossible by $\mathcal{Q}(W)$.
   (b) $\beta$ is created by the reduction from $W$ to $W'$.
   By case on the label's creation:
      i. $\beta = \lceil \gamma' \rceil$. Thus $\gamma' \prec \beta$ As $\alpha' \prec \lceil \gamma' \rceil$, we have $\alpha' \preceq \gamma'$ and therefore $\alpha' \prec \alpha'$, since $\gamma' \prec \alpha'$. Contradiction.
      ii. $\beta = \lfloor \gamma' \rfloor$ or $\beta = [\gamma', \beta_1]$ or $\beta = \langle \gamma', \beta_1 \rangle$. These cases are similar to the previous one.

□

To limit the number of cases to inspect, we consider a simple technical invariant on left subterms of application nodes and show that they cannot be labeled by an overlined or an underlined label.

**Invariant 2** *$\mathcal{R}(W)$ holds iff for any clipped subterm $UV$ in $W$, we have either $U = a : X$, or $U = [\alpha', \beta] : X$, or $U = \langle \alpha', \beta \rangle : X$.*

**Lemma 6.** *If $\mathcal{R}(W)$ and $W \to W'$, then $\mathcal{R}(W')$.*

Proof: Let $U'V'$ be a clipped subterm in $W'$. This application node comes from an application node $T = UV$ in $W$. Let $R = (\alpha' \cdot \lambda x.A)B$ be the contracted redex.

1. If $R$ is inside $T$, the only interesting case is when $U$ contains $R$. Then we have $U = \beta : X$, with $R$ inside $X$. Therefore $\beta$ is unchanged by the reduction step, and $\mathcal{R}(W)$ gives the form $a$, $[\gamma', \beta_1]$, or $\langle \gamma', \beta_1 \rangle$ of $\beta$.
2. If $R$ contains $T$ in its argument $B$. Then $U' = U$ and they have same labels.
3. Otherwise $U'$ comes from an $U$ in the function body part $A$ of $R$. Then, if $x \notin U'V'$, then $U' = U$ and this case is again straightforward. Otherwise, there can be a diffusion inside $U'$, but then the label of $U'$ is a tagged or marked pair. In all cases, $\mathcal{R}(W')$ holds.

□

Marked labels of the form $\langle \alpha', \beta \rangle$ record histories of reductions. A subterm with a marked label indicates that the application node just above has been activated by a redex with name $\alpha'$. Although marked labels are necessary for recording histories of reduction (see lemma 5), these labels have no impact on the definition of sharing. The following sharing relation $\simeq$ means that two labels are equal by erasing marks (in marked labels). It is defined inductively by:

$$\begin{array}{lll}
a \simeq a & \lceil \alpha' \rceil \simeq \lceil \alpha' \rceil & \lfloor \alpha' \rfloor \simeq \lfloor \alpha' \rfloor \\
\beta \simeq \gamma \;\Rightarrow\; [\alpha', \beta] \simeq [\alpha', \gamma] & \beta \simeq \gamma \;\Rightarrow\; \langle \alpha', \beta \rangle \simeq \langle \alpha', \gamma \rangle & \\
\beta \simeq \gamma \;\Rightarrow\; \beta \simeq \langle \alpha', \gamma \rangle & \beta \simeq \gamma \;\Rightarrow\; \langle \alpha', \beta \rangle \simeq \gamma &
\end{array}$$

Thus, if $\alpha \simeq \beta$, the labeled terms $\alpha : X$ and $\beta : Y$ are shared. This is expressed by the following invariant.

**Invariant 3** $\mathcal{P}(W)$ *holds iff, for any pair of subterms* $\alpha : X$ *and* $\beta : Y$ *such that* $\alpha \simeq \beta$, *we have* $X = Y$.

To prove this invariant, we need two extra invariants. The first one states that labels of free and bound variables cannot be equal up to the sharing relation.

**Invariant 4** $\mathcal{S}(W)$ *holds iff, for any pair of subterms* $\alpha : x$ *and* $\beta : y$ *such that* $\alpha \simeq \beta$, *we have* $x$ *free in* $W$ *iff* $y$ *free in* $W$.

**Lemma 7.** *If* $\mathcal{S}(W)$ *and* $W \to W'$, *then* $\mathcal{S}(W')$.

Proof: Firstly, a free (resp. bound) variable $x$ in $W'$ can only come from a free (resp. bound) variable $x$ in $W$. Secondly, a labeled subterm $\alpha : x$ in $W'$ can only come from a labeled subterm $\alpha_1 : x$ in $W$ where $\alpha \simeq \alpha_1$. Combining these two remarks gives the full proof. □

In the weak labeled $\lambda$-calculus, the name of a redex is the compound label found on the left part of an application towards the corresponding abstraction. In a dag representation of terms, we only share subterms. Hence both application and abstraction nodes of a shared redex are shared. This means that for a given redex name $\alpha'$, there must be sharing of the application subterm making the redex. In the terminology of the weak labeled $\lambda$-calculus, it means that we share the atomic label of the application node. To be more precise, we state the following invariant.

**Invariant 5** *$\mathcal{T}(W)$ holds iff, for any application subterms $\beta : (\alpha : X)U$ and $\gamma : (\alpha : Y)V$, we have $\beta \simeq \gamma$.*

**Lemma 8.** *If $\mathcal{P}(W) \wedge \mathcal{Q}(W) \wedge \mathcal{R}(W) \wedge \mathcal{T}(W)$ and $W \overset{\gamma'}{\Longrightarrow} W'$, then $\mathcal{T}(W')$.*

Proof: Take $\beta : (\alpha : X')U'$ and $\gamma : (\alpha : Y')V'$ in $W'$.

1. $\alpha$ exists in $W$. Then $\gamma' \not\leq \alpha$ since $\mathcal{Q}(W)$. Therefore, there are two labeled terms $U_1 = \beta_1 : (\alpha : X)U$ and $V_1 = \gamma_1 : (\alpha : Y)V$ in $W$, with a possible diffusion marking $U_1$ and/or $V_1$. Therefore, $\beta_1 \simeq \beta$ and $\gamma_1 \simeq \gamma$. By $\mathcal{T}(W)$, we have $\beta_1 \simeq \gamma_1$. Thus $\beta \simeq \gamma$.
2. $\alpha = [\gamma', \alpha_1]$. Then $\beta = [\gamma', \beta_1]$. Then $\gamma = [\gamma', \gamma_1]$, since both applications received a single diffusion, since all $\gamma'$ redexes are disjoint by $\mathcal{P}(W)$. In $W$, the corresponding applications are $\beta_1 : (\alpha_1 : XU)$ and $\gamma_1 : (\alpha_1 : YV)$. By $\mathcal{T}(W)$, we get $\beta_1 \simeq \gamma_1$. Thus $\beta \simeq \gamma$.
3. $\alpha = \langle \gamma', \alpha_1 \rangle$. Case similar to previous one.
4. Other cases, impossible since $\mathcal{R}(W)$.
   □

**Lemma 9.** *If $\mathcal{P}(W) \wedge \mathcal{Q}(W) \wedge \mathcal{R}(W) \wedge \mathcal{S}(W) \wedge \mathcal{T}(W)$ and $W \overset{\gamma'}{\Longrightarrow} W'$, then $\mathcal{P}(W')$.*

Proof: By $\mathcal{T}(W)$, the labels of all redexes with name $\gamma'$ have the same value $\gamma$ modulo $\simeq$. Therefore, by $\mathcal{P}(W)$, all redexes with name $\gamma'$ are the same $R = (\gamma' \cdot \lambda x.A)B$, and their contractums are identical $R' = \lceil \gamma' \rceil : (\gamma' \circledast A)[\![x \backslash \lfloor \gamma' \rfloor : B]\!]$. As these redexes are identical, they are all disjoint. Conversely, every subterm $R$ is a redex by $\mathcal{S}(W)$ if at least one $R$ is a redex in $W$.

Let $U' = \alpha : X'$ and $V' = \beta : Y'$ two subterms of $W'$ such that $\alpha \simeq \beta$.

1. Both $U'$ and $V'$ come from subterms $U = \alpha : X$ and $V = \beta : Y$ in $W$, with same label as $U'$ and $V'$. By $\mathcal{P}(W)$, we know that $X = Y$. There are two cases.
   (a) $U'$ contains a contractum. Then either $U$ contains $x$ replaced by $\lfloor \gamma' \rfloor : B$. Either $U$ contains $R$. The first alternative is impossible since then $R$ would contain $R$ in its argument. Since every $R$ in $X$ and $Y$ is a redex in $W$, then $X = Y$, $X \overset{\gamma'}{\Longrightarrow} X'$ and $Y \overset{\gamma'}{\Longrightarrow} Y'$. Thus $X' = Y'$.
   (b) $U'$ does not contain a contractum of $R$.
      i. $U'$ is disjoint from the contractums. Thus $U$ is a subterm of $W$ disjoint from redexes $R$. Then $U = U'$.
      ii. $U'$ is in the argument part of a contractum. Thus, $U$ is a subterm of $W$ in the argument part of a redex $R$.
      iii. $U'$ is in the body part of the contractum. Thus, $U$ is a subterm of $W$ in the function body part of a redex $R$, but $U$ does not contain the variable $x$ bound in $R$, since otherwise its label $\alpha$ would not be equal to the one of $U'$.

In any of these cases, $U' = \alpha : X' = U = \alpha : X$. Therefore $X' = X$. Moreover, as $X = Y$, then $V = \beta : X$ cannot contain $R$ (since X would do) or an occurrence of the bound variable of the function part in $R$, which means that $Y = Y'$. Thus $X' = Y'$.

2. Both labels $\alpha$ and $\beta$ are new in $W'$. Thus $\gamma' \prec \alpha$ and $\gamma' \prec \beta$.
   (a) $\alpha = [\gamma', \alpha_1]$. As $\alpha \simeq \beta$, we have $\beta = [\gamma', \beta_1]$ and $\alpha_1 \simeq \beta_1$.
   In $W$, there exist two labeled subterms $\alpha_1 : X_1$ and $\beta_1 : Y_1$ such that $\alpha : X' = (\gamma' ⓧ \alpha_1 : X_1)[\![x \backslash \lfloor \gamma' \rfloor : B]\!]$ and $\beta : Y' = (\gamma' ⓧ \beta_1 : Y_1)[\![x \backslash \lfloor \gamma' \rfloor : B]\!]$. By $\mathcal{P}(W)$, we have $X_1 = Y_1$ which implies $X' = Y'$.
   (b) $\alpha = \lceil \gamma' \rceil$. By $\mathcal{Q}(W)$, this label is absent from $W$. It is created at the top of contractums of $\gamma'$-redexes, which are all equal by $\mathcal{P}(W)$.
   (c) $\alpha = \lfloor \gamma' \rfloor$. By $\mathcal{Q}(W)$, this label is absent from $W$. It is created at the top of copies of arguments of $R$ in the contractums. These copies are all equal.
   (d) $\alpha = \langle \gamma', \alpha_1 \rangle$. As $\alpha \simeq \beta$ and $\beta$ is new, we can only have $\beta = \langle \gamma', \beta_1 \rangle$ and $\alpha_1 \simeq \beta_1$. Both $U'$ and $V'$ come from $U$ and $V$ to the left of an application after a diffusion, when $x \notin U$ and $x \notin V$. So $U = \alpha_1 : X$ and $V = \beta_1 : Y$. Moreover $X = X'$ and $Y = Y'$, since $x \notin X$ and $x \notin Y$. By $\mathcal{P}(W)$, we get $X = Y$. Hence $X' = Y'$.
3. $\alpha$ is created, but $\beta$ already exists in $W$. By $\mathcal{Q}(W)$, we know that $\gamma' \not\prec \beta$. As $\alpha \simeq \beta$, we can only have $\alpha = \langle \gamma', \alpha_1 \rangle$ and $\alpha_1 \simeq \beta$. Therefore, there is a term $U = \alpha_1 : X$ to the left of an application subterm in $W$, with $x \notin X$, and $U' = \langle \gamma', \alpha_1 \rangle : X'$, with $X = X'$. As $V' = \beta : Y'$ comes from $\beta : Y$ in $W$, we know by $\mathcal{P}(W)$ that $X = Y$. If $Y$ is modified from $W$ to $W'$, this can only be because $Y$ contains a $\gamma'$-redex, which is impossible since $X$ is contained in a $\gamma'$-redex; or because it has been substituted after a diffusion, but then it is unchanged since $x \notin X = Y$. Therefore $Y = Y'$, and $X' = X = Y = Y'$.
4. $\alpha$ exists in $W$, but $\beta$ is created. Analogous to previous case.

□

We can now state our sharing theorem, characterizing a dag implementation for evaluating terms in the weak $\lambda$-calculus. We need to have a notation for terms without sharing.

**Notation 1** *Let Init$(U)$ holds when every subterm of U is labeled with a distinct letter.*

**Theorem 5.** *Let Init$(U)$ and $U \Longrightarrow\!\!\!\!\!\!\!\!\!\!\Longrightarrow V$, then $\mathcal{P}(V)$.*

Proof: We first notice that, if *Init*$(U)$, then $\mathcal{P}(U) \wedge \mathcal{Q}(U) \wedge \mathcal{R}(U) \wedge \mathcal{S}(U) \wedge \mathcal{T}(U)$. Thanks to previous lemmas, this invariant remains valid along reduction steps $\Longrightarrow\!\!\!\!\!\!\!\!\!\!\Longrightarrow$. □

## 5 Dag implementation

The weak labeled $\lambda$-calculus provides a formal setting for a dag implementation of the weak $\lambda$-calculus. If we start from a term labeled with distinct letters, each label identifies the address of a shared subterm in the dag implementation. There is a subtlety: first components of marked labels must be skipped, because they are only relevant for storing the history of redexes. The goal of our labeled calculus is to coincide with the dag implementation in Wadsworth [25].

Wadsworth has two implementations with sharing. The first one shares the argument of the contracted redex at each reduction step. When there are $n$ occurrences of $x$ in $A$, instead of having $n$ copies of the argument $B$ in the contractum of $(\lambda x.A)B$, Wadsworth proposes to replace $x$ by a reference to a single subterm $B$. However his method demands to copy the function part $\lambda x.A$ of the contracted redex if the reference counter of $\lambda x.A$ is strictly greater than 1. This disallows the substitution of $x$ in other subterms using $A$. In his second dag implementation, Wadsworth notices that it is unnecessary to copy subterms in $A$ that do not contain occurrences of the free variable $x$. This variant of the dag implementation avoids unnecessary copies.

However finding if a term does not contain an occurrence of the $x$ variable is a global operation. The occurrences of $x$ in a redex $(\lambda x.A)B$ cannot disappear in the weak $\lambda$-calculus, since there cannot be redexes containing $x$ inside $A$. Therefore, one may compile any abstraction $\lambda x.A$ in the initial term to mark each subterm of $A$ with free variables bound outside $\lambda x.A$. This approach is taken by supercombinators and $\lambda$-lifting [22], where each abstraction is $\lambda$-lifted into a combinator to which are applied the free variables contained in this abstraction. But the calculus of supercombinators adds extra reduction rules, and is not a subcalculus of the $\lambda$-calculus. More precisely, Peyton-Jones and Hughes introduced the idea of *full lazyness* similar to the diffusion operator, by creating combinators abstracting over *maximal free subterms*, and it would be interesting to connect our sharing theory and their compilation scheme.

Shivers and Wand [23] provide a more realistic implementation where, in the abstract tree corresponding to any abstraction $\lambda x.A$, the top node points to the list of occurrences of the variable $x$ bound in this abstraction. In their representation of terms, any subterm points to the subterm directly containing it. Thus, terms are represented by graphs with a double-linked connection between vertices (i.e. nodes of the abstract trees). This representation facilitates access to terms and subterms in the usual top-down way, but makes also possible bottom-up traversals of terms from binders of abstractions through the bound occurrences of variables towards the root of an abstraction, or the root of a term. Therefore during a reduction step contracting redex $(\lambda x.A)B$, one first duplicates the paths from top of $A$ to $x$ in a bottom-up traversal from every occurrence of $x$ towards the top node of $A$. We thus copy nodes corresponding to subterms containing $x$ by performing local operations. However, as on the path from an occurrence of $x$ to the top node of $A$, one may encounter an other binder, it is also necessary to make a recursive call to duplicate paths to its bound variable. This method is an efficient way of implementing the second method in Wadsworth [25].

A closer look at our labeled $\lambda$-calculus demonstrates several differences between Shivers and Wand's method and our calculus. The creation of nodes along paths to occurrences of the bound variable $x$ corresponds to the new tagged labels created by diffusion. But we can meet other binders $\lambda y$ on these paths. Then we do not copy the paths from such a $\lambda y$ to the free variable $y$ during diffusion. Therefore we can have several binders for the same occurrence of the bound variable $y$. However no confusion occurs in our calculus, since sharing is virtual and is only represented through labels. But this situation in an actual shared implementation is unsafe, since it would hardly respect alpha renaming. Extending the diffusion of our calculus could address this problem but it remains to prove that the calculus is still confluent, which looks feasible. Such a modification of the weak $\lambda$-calculus would be one way of providing a formal theory to the implementation in [23].

There are other subtle differences between our labeled $\lambda$-calculus and standard dag implementations. One is the introduction of new nodes, since, in a reduction step, we create new nodes, with $\lceil \alpha \rceil : U$ and $\lfloor \alpha \rfloor : U$. These nodes guarantee that we do not lose the history of the creation of redexes along a reduction. However, they do not seem necessary with respect to sharing. Similarly, it is quite debatable to have a label on a bound variable. We have labeled variables since the structure looks then more regular. But a calculus without these labels seems also feasible, without destroying sharing.

We can notice that the weak labeled $\lambda$-calculus makes a confluent theory of dag implementation. Usually, confluence proofs with graphs are rather delicate unless strong restrictions as in interaction nets [12]. But in the latter case, the theory only considers $\Longrightarrow$ reduction steps. Here we can reason about terms with induction as in the classical $\lambda$-calculus. This is what we gain with labeled calculi, since we have a textual representation of dags by handling the node address, the atomic label, and the $\lambda$-term.

Finally, the call-by-need strategies $\underset{\text{norm}}{\Longrightarrow}$ of the labeled calculus of weak explicit substitutions correspond to complete labeled reduction steps in the weak labeled $\lambda$-calculus. If $Init(U)$ holds in the initial labeled term, one can show that the number of steps to get a normal form with this reduction is always minimal. The proof technique follows the one for the classical $\lambda$-calculus [17].

## 6 Relation with Term Rewriting Systems

One referee pointed out that the weak $\lambda$-calculus could be considered as a first-order term rewriting system (TRS) [24]. He mentioned that, for instance, the pattern of the weak $\beta$-redex $(\lambda x.Ix(\lambda y.Ixy))S$ is $(\lambda x.Z_1x(\lambda y.Z_2xZ_3))X$ giving rise to the first-order rule $(\lambda x.Z_1x(\lambda y.Z_2xZ_3))X \rightarrow Z_1X(\lambda y.Z_2XZ_3)$, where any terms can be substituted for the meta variables $Z_1$, $Z_2$, $Z_3$ and $X$. One could then substitute $I$ for both $Z_1$ and $Z_2$, $y$ for $Z_3$ and $S$ for $X$ to yield the step $(\lambda x.Ix(\lambda y.Ixy))S \rightarrow IS(\lambda y.ISy)$. With this remark, most of the properties of residuals hold for TRSs and thus might look more natural. However, the corresponding labeled TRS (see [24, 20]) differs from our weak labeled $\lambda$-calculus. As mentioned in the introduction, we think that our calculus has the advantage of staying in a subtheory of the classical $\lambda$-calculus.

## 7 Conclusion

In this paper, we recalled the definitions and basic properties of the weak $\lambda$-calculus introduced in [18]. This calculus follows the standard presentations of the $\lambda$-calculus, since the classical (strong) $\lambda$-calculus is an extension of it.

We presented a weak labeled $\lambda$-calculus which is a confluent theory of sharing for the weak $\lambda$-calculus. We stated that it corresponds to the representation of terms in Wadsworth's dag implementations but a very precise connection has still to be formalized.

The correspondence between the labeled $\lambda$-calculus and the calculus of sharing with explicit substitutions could also be studied as in [18]. This would make a bridge with the call-by-need strategies used in the evaluation of functional programs. One can also relate supercombinators and the shared structures for evaluating them in lazy languages to our calculus. A similar approach with respect to frameworks with a `let` statement could also be studied, although these research directions are not focused on basic syntactic properties such as confluence. (Notice that our calculus has no critical pairs.)

The theory of optimal reductions inside the weak $\lambda$-calculus is missing in this article, although one might easily guess it following the one of the classical $\lambda$-calculus. It remains to achieve it.

Finally, many of the results of this paper were considered as folk theorems, rather easy to prove. We hope to have shown that some of the proofs deserve attention. In fact, some of them are not easy at all. This paper also showed that the theory of the weak $\lambda$-calculus still needs more research.

## Acknowledgements

We have benefited from numerous comments by the referees.

## References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 6(2):299–327, 1996.
2. Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call-by-need lambda calculus. In *Proc. 22nd ACM Symposium on Principles of Programming Languages*, pages 2330–246, January 1995.
3. A. Asperti, P. Coppola, and S. Martini. (Optimal) duplication is not elementary recursive. *Information and Computation*, 193/1:21–56, 2004.
4. A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages.* Cambridge University Press, 1999.
5. H. P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics.* North-Holland, 1981.
6. N. Çağman and J. R. Hindley. Combinatory weak reduction in lambda calculus. *Theoretical Computer Science*, 198:239–249, 1998.
7. P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, 1996.

8. M. Fernández and I. Mackie. Closed reductions in the lambda-calculus. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic*, volume 1683 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 1999.
9. G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proc. of the 19th Conference on Principles of Programming Languages*, pages 15–26. ACM Press, 1992.
10. J. R. Hindley. Combinatory reductions and lambda reductions compared. *Zeit. Math. Logik*, 23:169–180, 1977.
11. J. W. Klop. *Combinatory Reduction Systems.* PhD thesis, Rijksuniversiteit Utrecht, 1980.
12. Y. Lafont. Interaction nets. In *Principles of Programming Languages*, pages 95–108. ACM Press, 1990.
13. J. Lamping. An algorithm for optimal lambda-calculus reduction. In *Proceedings of the 17th Annual ACM Symposium on Principles of Programming Languages*, pages 16–30, 1990.
14. J. Launchbury. A natural semantics for lazy evaluation. In *Proc. of the 1993 conference on Principles of Programming Languages*, pages 144–154. ACM Press, 1993.
15. J. L. Lawall and H. G. Mairson. On the global dynamics of optimal graph reduction. In *ACM International Conference on Functional Programming*, pages 188–195, 1997.
16. X. Leroy. The ZINC experiment: an economical implementation of the ML language. Technical report 117, INRIA, 1990.
17. J.-J. Lévy. *Réductions correctes et optimales dans le lambda-calcul.* PhD thesis, Univ. of Paris 7, Paris, 1978.
18. J.-J. Lévy and L. Maranget. Explicit substitutions and programming languages. In *Proc. 19th Conference on the Foundations of Software Technology and Theoretical Computer Science, Electronic Notes in Theoretical Computer Science*, volume 1738, pages 181–200, 1999.
19. J. Maraist, M. Odersky, D. N. Turner, and P. Wadler. Call-by-name, call-by-value, call-by-need, and the linear lambda calculus. In *Electronic Notes in Theoretical Computer Science*, pages 41–62, March 1995.
20. L. Maranget. *La stratégie paresseuse.* PhD thesis, Univ. of Paris 7, Paris, 1992.
21. P.-A. Melliès. *Description Abstraite des Systèmes de Réécriture.* PhD thesis, Univ. of Paris 7, december 1996.
22. S. L. Peyton-Jones. *The implementation of Functional Programming Languages.* Prentice-Hall, 1987.
23. O. Shivers and M. Wand. Bottom-up beta-substitution: uplinks and lambda-dags. Technical report, BRICS RS-04-38, DAIMI, Department of Computer Science, University of Århus, Århus, Denmark, 2004.
24. Terese. *Term Rewriting Systems.* Cambridge University Press, 2003.
25. C. P. Wadsworth. *Semantics and pragmatics of the lambda-calculus.* PhD thesis, Oxford University, 1971.