Axiomatic Sharing-via-Labelling

Thibaut Balabonski

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

Abstract

A judicious use of labelled terms makes it possible to bring together the simplicity of term rewriting and the sharing power of graph rewriting: this has been known for twenty years in the particular case of orthogonal first-order systems. The present paper introduces a concise and easily usable axiomatic presentation of sharing-via-labelling techniques that applies to higher-order term rewriting as well as to non-orthogonal term rewriting. This provides a general framework for the sharing of subterms and keeps the formalism as simple as term rewriting.

1998 ACM Subject Classification I.1.3 Languages and Systems: Evaluation strategies

Keywords and phrases Sharing, Abstract term rewriting, Graphs, Higher order, Non-orthogonality

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

Category Regular Research Paper

1 Introduction

Termgraph rewriting [9, 27] improves on term rewriting by allowing the sharing of subterms, thus preventing some duplications of computation steps. However, graph formalisms are usually far more complex than the corresponding term formalisms. Higher-order graphs [29, 11, 19] in particular require non-trivial correctness criteria and readback procedures.

This paper aims at providing a fully general framework for the sharing of subterms, while keeping the formalism as simple as term rewriting. It substantially generalizes an elegant partial solution originating in L. Maranget's work [25], of which we name two features:

- Graphs are represented by labelled terms.
- Graph reduction is simulated by sequences of term reduction.

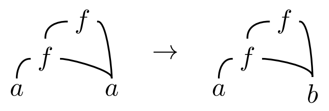
Thus we get a formal setting for graph rewriting using only the well-known term rewriting.

This started 20 years ago with orthogonal first-order rewriting [25], and has then been extended to several weak λ-calculi [10, 7] or orthogonal extensions of these weak calculi [6]. Thanks to its technological simplicity the approach turned out to be useful as an analysis tool for various purposes, in particular by allowing the description of optimal reduction strategies [25] and by allowing simple comparisons between different graph implementations of the λ-calculus [7].

In this line of work, any symbol in a term is given a label which is a metaphorical location. Two subterms with the same label are supposed to be physically equal, stored at the same location in memory or drawn at the same coordinate in a picture (see Example 1). Graph rewriting is then simulated by the *simultaneous* reduction of all the term-redexes corresponding to a unique graph-redex, which means all the term-redexes with a given label.

▶ **Example 1.** The two occurrences of a^δ denote the same node and are reduced simultaneously. Across this paper we use a single arrow $\to$ for the reduction of one redex, and a double arrow $\Rightarrow$ for the simultaneous reduction of several redexes.

$$\begin{array}{rl} & f^\alpha(f^\beta(a^\gamma, a^\delta), a^\delta) \\ \Rightarrow & f^\alpha(f^\beta(a^\gamma, b^\zeta), b^\zeta) \end{array}$$



◀

This correspondence is sound as long as the system preserves a *sharing property* on its labelled terms: whenever two subterms coexist with the same label they are syntactically equal. Note that in this setting, a labelled term t satisfying the sharing property is itself a witness of the correctness of the corresponding graph $\mathcal{G}_t$, and that $\mathcal{G}_t$ is an *acyclic graph.*

While simultaneous reduction in general is a non-trivial extension to one-step rewriting, the particular case considered here is simply definable within usual term rewriting. Indeed we consider the simultaneous reduction of *disjoint* subterms, which we call *parallel reduction.* It is simulated by the sequential reduction of the subterms in any order.

▶ **Example 2.** The parallel reduction of Example 1 can be implemented by:
$f^\alpha(f^\beta(a^\gamma, a^\delta), a^\delta) \quad \to \quad f^\alpha(f^\beta(a^\gamma, b^\zeta), a^\delta) \quad \to \quad f^\alpha(f^\beta(a^\gamma, b^\zeta), b^\zeta)$ ◀

In [25, 10, 6, 7] the labels are generated in an effective way, which helps to define the labelled reduction by simple term rewriting rule schemes. The generation of labels in a reduction is in two steps: first the redex is given a name Ω based on its labels, then new labels are generated using Ω. For instance in Example 2, $\Omega = \delta$ and $\zeta = <\Omega, \zeta_0>$ for some ζ_0 fixed in the rule scheme.

In this method inspired by Lévy's labelling for optimal sharing [23], the name of a redex reflects the past reductions *causally* related to this redex, and is *invariant* from the creation of the redex to its disappearance. These are key ingredients for providing absolute bounds on the sharing of redexes, which is the point of Lévy's optimality theory. In [25, 10, 6, 7], both of these points also seem to play an important role in the preservation of the sharing property. This limits the applicability of these labellings for sharing in several ways:

Orthogonality For the redex names to remain static, no reduction should ever interfere with an existing redex in a way that could alter its name. Hence the restriction to orthogonal systems where no two reduction rules can apply at overlapping sets of positions.

Causal sharing For each system the causal constraints of the previous labellings essentially determine one level of sharing. Hence defining other sharing disciplines requires tampering with the dynamics of the system. It is done in [7] by introducing various fine-tuned notions of weak reduction (systems where reduction in the scope of a binder is constrained).

Weak reduction In higher-order rewriting, the "causal sharing" is typically incompatible with our acyclic graphs, as the optimal sharing of the λ-calculus. Thus the previous approaches apply to higher-order only through a restriction to weak systems that yield compatible sharings. While the restriction does not prevent one from expressing the most common evaluation strategies of the λ-calculus, it seems to rule out any truly higher-order behaviour. Indeed, two common weak λ-calculi have been shown in [21, 7] to be strongly equivalent to orthogonal first-order rewriting systems.

This paper generalizes the previous approaches, in particular by freeing them of their bond to Lévy's optimality theory. It expresses sharing-via-labelling in an abstract rewriting framework encompassing any term rewriting system, be it higher-order and non-orthogonal. In this framework we show that the preservation of sharing needs neither a static identification of redexes, nor a tight relation to causality. We thus break through the previous limitations by giving to redexes a *dynamic* identity. And this makes all the difference!

Our main contribution is an axiomatic framework for the sharing of subterms that is:

Simple: it does not require more formal technology than term rewriting.

Expressive: it can describe a variety of graph algorithms for any term-style rewriting system.

Usable: it relies only on few axioms, and comes with highly adaptable concrete examples.

As a second contribution, we use the axiomatic framework to revisit call-by-need reduction and improve its sharing (Section 5).

Section 2 gives an overview of our axiomatic framework and its main mechanisms, and discusses its expressive power. Section 3 provides all the formal definitions and proofs. Section 4 presents some simple examples, and Section 5 develops a more elaborate example that takes advantage of the specifities of our approach. Other approaches to sharing and to graph reduction are reviewed in Section 6, then Section 7 concludes.

2 Overview of the axiomatic framework

This section gives an informal presentation of our axiomatic framework and discusses its mechanisms. Section 2.1 sets the scene of an abstract term rewriting framework. Section 2.2 presents the mechanisms of our axiomatic labelling on an example, and Section 2.3 discusses the expressive power of the whole framework.

2.1 Principles of the basic formalism

Abstract Rewriting Systems (ARS) [26, 28] give a fully general description of rewriting. In [26] any system is described by only four basic elements: a set $\mathcal{O}$ of objects, a set $\mathcal{R}$ of reductions, and two functions $\mathtt{src} : \mathcal{R} \to \mathcal{O}$ and $\mathtt{tgt} : \mathcal{R} \to \mathcal{O}$ assigning a source object and a target object to each reduction. Hence *ARS* can express rewriting on any kind of objects through any kind of rewriting rules with any kind of rule application conditions.

We introduce *Abstract Term Rewriting Systems (ATRS*, Definition 9*)* as a specialization of *ARS* to term rewriting by taking as set of objects $\mathcal{O}$ a set of terms. Now each reduction r is supposed to apply to a particular subterm of the source, at a position $\mathtt{root}(r)$ called root of r, and to leave the rest unchanged (see Fig. 1).

Equivalently, any reduction r can be identified by three other components (see Fig. 2): the redex $\mathtt{dex}(r)$ and reduct $\mathtt{duct}(r)$ are the source-version and target-version of the modified subterm and the context $\mathtt{ctx}(r)$ is what remains and is common to the source and the target.

2.2 Principles of the axiomatic labelling

In this section we derive from scratch a labelling for a higher-order rewriting system. By showing the backstage of this sample construction we present the main ideas of the axiomatic labelling defined in Section 3 (Definition 17).

We consider the fixpoint operator expressed by the rule scheme: $\mu x.t \ \to\ t^{\{x:=\mu x.t\}}$. We analyse the following graph reduction, where the symbol h is duplicated but the expression $g(a)$ is kept shared.

$$\begin{array}{lll} & f(g(a),\ \underline{\mu x.h(g(a),x)}\ ) & (t_1) \\ \to & f(g(a),\ \underline{h(g(a),\mu x.h(g(a),x))}\ ) & (t_2) \end{array}$$

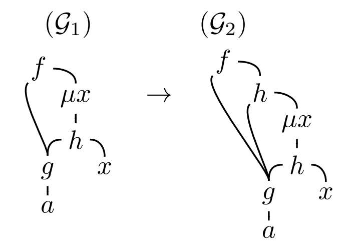


Figure 1 Abstract term reduction, ARS-style: source, target, root.

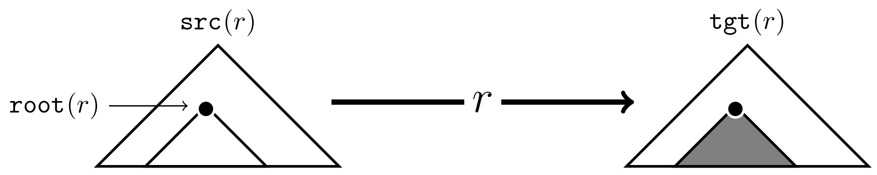
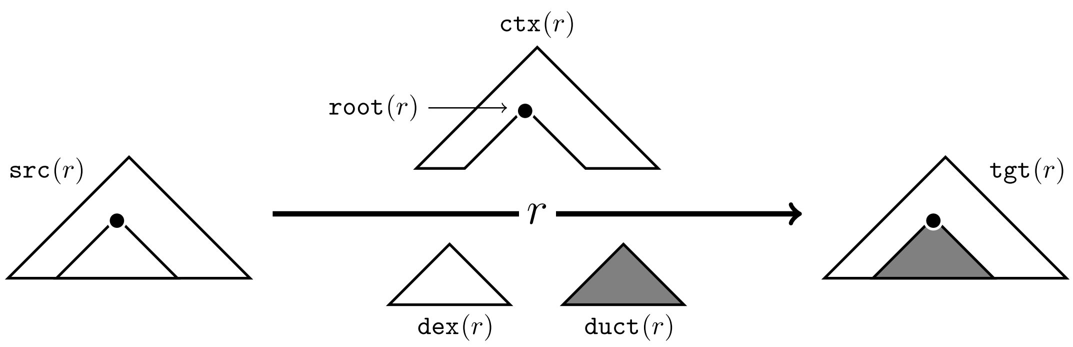
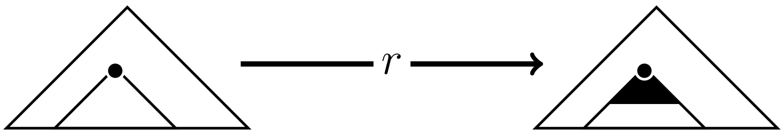


Figure 2 Abstract term reduction, TRS-style: redex, reduct, context.



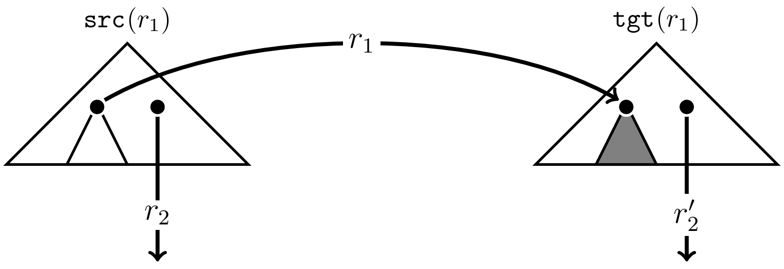
Remark that either triple $(\texttt{src}, \texttt{tgt}, \texttt{root})$ or $(\texttt{dex}, \texttt{duct}, \texttt{ctx})$ is enough to deduce the other informations. For instance, decomposing the source at the root position gives the context and the redex. Conversely, combining the context and the reduct yields the target.

Figure 3 Effect zone



The effect zone, in black, is a connected part of the reduct that contains all that is created or modified by the reduction.

Figure 4 Reductions at disjoint positions



The reduction r_1 cannot suppress the disjoint reduction r_2: some similar r'_2 remains in the target of r_1. An equivalence of reductions (Definition 9) relates r_2 and r'_2.

We want to represent the graph $\mathcal{G}_1$ by a labelling t_1^l of the term t_1 such that each label denotes a node of the graph. We label the symbols f, g and h. In particular the labels of the two occurrences of g have to be equal, and the other labels have to be different. For instance:

$$f^\alpha(g^\beta(a),\ \mu x.h^\gamma(g^\beta(a), x)\ ) \qquad (t_1^l)$$

Now the reduction of t_1^l will act on the labels along the following principles. If labels figuratively represent memory locations, then label generation corresponds to memory allocation. Hence the new labels correspond to creations or to duplications of graph nodes. Call effect zone and write $\texttt{effz}(r)$ the part of the target of a reduction r where new labels are created (see Fig. 3). Intuitively, a smaller effect zone tends to make less duplications.

First, we do not want the reduction to modify its context: the effect zone is contained in the reduct $h(g(a), \mu x.h(g(a), x))$ and the labelled context $f^\alpha(g^\beta(a), \_)$ keeps its labels.

Second, remark that some subterms of the reduct are exact copies of some subterms of the redex. This is the case for instance of $h(g(a), x)$, or of the first occurrence of $g(a)$. We do not need to make new copies of these subterms. Thus they can be kept outside of the effect zone and inherit the labels of their source-side counterparts. Axiom *Effect zone* in Definition 13 checks that any other position is in the effect zone.

Finally, only the outer occurrence of h needs a new label, say δ. Then we can take the following labelling t_2^l of t_2 to represent $\mathcal{G}_2$:

$$f^\alpha(g^\beta(a),\ h^\delta(g^\beta(a), \mu x.h^\gamma(g^\beta(a), x))\ ) \qquad (t_2^l)$$

At this point, we have to generate a new label δ in a way that does not break the sharing property. Preferably, we would like a method that deduces the new labels from the redex and does not require any global information on the context. This fundamental choice of design allows us to define labelled reduction as a usual term rewriting rule scheme.

To avoid unintended clashes, the idea is to blend into the new label δ some information that is characteristic of the reduced redex. As a consequence, two distinct redexes will generate different labels, while two copies of the same redex will naturally have similar evolutions. In our metaphor a safe such characterization of a redex is its physical identity, in other words its label. This choice is possible if and only if the root of the reduced redex is labelled, which is required by an axiom *Redex head label*. In our example, we add a label ω to the symbol μx in t_1^l and get the following corrected labelling of t_1:

$$f^\alpha(g^\beta(a),\ \mu x^\omega.h^\gamma(g^\beta(a), x)\ ) \qquad (t_1^l)$$

Then the label δ can be explicitly built from ω and other information taken in the redex: say for instance that h^δ is the copy of h^γ ordered by a redex of label ω, and write $\delta = [\omega]\gamma$.

Such a generation method is sound if the following last requirement is met: once a redex of label ω is reduced, creating compounds such as $[\omega]\gamma$, no other redex labelled by ω should ever be created. Or better, no new occurrences of the label ω should ever be generated.

A first, short-term, precaution is to change the label of μx, which is done by putting μx in the effect zone. This is one requirement of the axiom *Effect zone*: the copies of the root of the redex must be in the effect zone. Also remark that the new label of μx, for instance $[\omega]$, has to be taken different from the new label δ created for h. An axiom *Separation* states that the labels created in the same reduction do not clash.

Then we need to check this property on the long run. We use two new ingredients:

- A strict order $\hookrightarrow$ on labels called *contribution* such that $\alpha \hookrightarrow [\alpha]$ and $\alpha \hookrightarrow [\alpha]\beta$ for any two labels α and β. An axiom *Contribution* expresses that the generated labels progress along the contribution order.

- An *independence* invariant on labelled terms that forbids the existence in a term of two labels α and β such that $\alpha \hookrightarrow \beta$ (*Independence property*, Definition 19).

In this setting we have all the ingredients needed to prove that the sharing property is preserved by parallel reduction (Sharing theorem 21).

2.3 Expressive power

Our framework axiomatizes sharing-via-labelling along three directions: the underlying term systems, the graph algorithms, and the labellings. Each of these levels of abstraction brings its own expressive power. They are reviewed here in reverse order.

The labels play two roles in our framework. As announced they first describe sharing, but they moreover have to contain enough information to allow the generation of fresh labels. The axiom *Contribution* we use for this is strictly less restrictive than its equivalent in the previous approaches. In particular our labels need not contain as much causal information as their predecessors, and our framework covers the previous labellings as well as new and possibly simpler ones that are illustrated across this paper (see Section 4.2). In particular, the sequences of labels and the dummy nodes that create arbitrary long chains of indirections in [10, 7] are no longer necessary. This is possible thanks to an invariant of *independence* that is strictly more general than its predecessors, and that is a key of our technical contribution.

The graph algorithms that can be described in our framework come from the combination of two parameters: first the effect zone specifies the positions for which new nodes are created, then the labelling decides on the positions of the effect zone that are to be shared or not. The first parameter is constrained by axiom *Effect zone* and the second by axiom *Separation*, which are both expressed in terms of equality of subterms. This allows us to describe a variety of graph reduction systems, such as the many variants of call-by-need, lazy, and fully lazy reduction that have been studied on the λ-calculus or similar systems. Some have been reviewed in [7] and encoded in a sharing-via-labelling framework, at the cost of defining finely-tuned reduction strategies. Such studies can be carried on in our new framework without tampering with the dynamics of the systems (see Section 4.2).

Our abstract term rewriting framework makes no assumption on the shape of the reduction rules of the underlying term systems, and few on the application conditions of these rules. The only restriction is the fairly natural one that the reductions affecting disjoint subterms of the same source term do not interfere (see Fig. 4). This allows the sequentialization of parallel reductions as in Example 2 (Lemma 12). This assumption is formalized in an axiom *Residuals* that allows a reduction r_1 to inhibit a reduction r_2 of same source term only if the position of one reduction is a prefix of the position of the other, or in other words one redex is a subterm of the other. Thus, virtually any reasonable term-style rewriting system can be represented. We list below some classes of higher-order systems that are rewriting-theoretically interesting, and moreover particularly relevant to the study of functional programming.

Conditional calculi *Closed reduction strategies* [14] are a family of fragments of the λ-calculus particularly convenient for implementation purposes as they are α-conversion free. They introduce conditions on β-reduction based on the free variables of the redexes.

Non-right-linear systems The fixpoint seen above is such a system, since t appears twice in the right member of $\mu x.t \;\to\; t^{\{x := \mu x.t\}}$.

Non-orthogonal systems Higher-order rewriting is useful to the study of *program transformations*. This may yield non-orthogonal systems as the fully-lazy λ-lifting presented in [7]. Another famous reservoir of non-orthogonal higher-order rewriting systems is the literature on calculi with *explicit substitutions* [1, 20]. Explicit substitutions are an important tool to bridge the gap between the formal definition of β-reduction and its various implementations. Section 5 presents a calculus with `let`-constructs having a comparable flavor.

Non-sequential systems Defining functions by *pattern matching* is a key feature of functional programming. One of its major difficulties is the treatment of matching failure, which is non-sequential in general [18], and similar to the *parallel or* operator.

3 The axiomatic framework

This section defines *Sharing-via-Labelling Systems (SvLS*, Definition 17, Section 3.3*)* and their *parallel labelled reduction* (Definition 18), and then proves that the latter preserves the sharing property (Theorem 21).

The construction follows the principles presented above in three steps: *Abstract Term Rewriting Systems (ATRS*, Definition 9, Section 3.2*)* are the basic formalism for term rewriting. *Marked ATRS (*Definition 13, Section 3.2*)* enrich *ATRS* with a marking of the potential effects of a reduction (the effect zone). *SvLS* are *Marked ATRS* whose terms are partially labelled. We start by some basic and mostly natural definitions on terms.

3.1 Basic syntax

▶ **Definition 3** (Terms)**.** Let $\mathcal{S}$ be a finite or countable set called **signature**. **Terms** over $\mathcal{S}$ are given by the following grammar:

$$t \quad ::= \quad f \mid f(t_1, ..., t_n)$$

where $f \in \mathcal{S}$ and $n \geq 1$. Remark that the symbols in $\mathcal{S}$ have *a priori* variable arities. This can be constrained later. A **context** c is a term with a hole. Write _ for the hole, and $c[t]$ for the term obtained by replacing the hole of c by the term t. ◀

▶ **Example 4.** $@(\lambda x(\_), y)[@(x,y)] = @(\lambda x(@(x,y)), y)$ ◀

▶ **Definition 5** (Positions)**.** A **position** p is a possibly empty sequence of positive integers. Write ϵ the **empty position**, and $p \cdot q$ the **concatenation** of two positions p and q. The concatenation extends to sets as follows: $P \cdot Q = \{p \cdot q \mid p \in P,\ q \in Q\}$. Write $p \cdot Q$ as a shorthand for $\{p\} \cdot Q$. A position p is a **prefix** of a position q, written $p \prec q$, if there is a position p' such that $p \cdot p' = q$. If moreover $p' \neq \epsilon$ then the prefix is **strict**. Two positions p, q are **disjoint** when none is a prefix of the other. An **initial segment** is a set of positions P such that for any positions $q \prec p$, if $p \in P$ then $q \in P$.

The **set of positions** $pos(t)$ of a term t is defined as follows:

$$\begin{array}{rcl} pos(f) & = & \{\epsilon\} \\ pos(f(t_1, ..., t_n)) & = & \{\epsilon\} \cup \bigcup_{1 \leq i \leq n} i \cdot pos(t_i) \end{array}$$

For any term t and position $p \in pos(t)$, write $t(p)$ the **symbol** at position p in t and $t|_p$ the **subterm** of t at position p, defined as follows:

$$\begin{array}{rcl|rcl} f(\epsilon) & = & f & f|_\epsilon & = & f \\ f(t_1, ..., t_n)(\epsilon) & = & f & f(t_1, ..., t_n)|_\epsilon & = & f(t_1, ..., t_n) \\ f(t_1, ..., t_n)(i \cdot p) & = & t_i(p) & f(t_1, ..., t_n)|_{i \cdot p} & = & t_i|_p \end{array}$$ ◀

▶ **Example 6** (λ-terms)**.** Let $\mathcal{X}$ be a countable set of variables. In the signature $\mathcal{S}_\Lambda$ of the λ-calculus we consider the binding λx as one symbol: $\mathcal{S}_\Lambda = \mathcal{X} \cup \{\lambda x \mid x \in \mathcal{X}\} \cup \{@\}$. Let $t = @(\lambda x(@(x,y)), y)$, then $pos(t) = \{\epsilon, 1, 11, 111, 112, 2\}$, $t(1) = \lambda x$ and $t|_{11} = @(x,y)$. ◀

▶ **Definition 7** (Parallel replacement). For any terms t, u and any set of pairwise disjoint positions $P \subseteq pos(t)$, write $t[u]_P$ the **parallel replacement** defined as follows:

$$\begin{array}{rcll} t[u]_\emptyset & = & t & \\ t[u]_{\{\epsilon\}} & = & u & \\ f(t_1, ..., t_n)[u]_P & = & f(t_1[u]_{P_1}, ..., t_n[u]_{P_n}) & \text{where } P = \bigcup_i i \cdot P_i \end{array}$$ ◀

▶ **Example 8** (Parallel replacement). $f(f(a,a),a)[b]_{\{12,2\}} = f(f(a,b),b)$ ◀

3.2 Abstract Term Rewriting Systems (*ATRS*)

We define *Abstract Term Rewriting Systems* as a specialization of *ARS* where the set $\mathcal{O}$ of objects is a set of terms. Each reduction r is associated to a redex $\mathtt{dex}(r)$, a reduct $\mathtt{duct}(r)$, and a context $\mathtt{ctx}(r)$ (Fig. 2).

Axiom *Source & Target* ensures that the source $\mathtt{src}(r)$ and the target $\mathtt{tgt}(r)$ of a reduction r are in $\mathcal{O}$ (which has to be understood as the set of *well-formed* terms). Axiom *Residuals* requires that two reductions concerning disjoint positions do not interfere with each other (Fig. 4). It is formalized using a notion of *equivalence* of redexes characterizing redexes that differ only by their context. Axiom *Residuals* ensures that parallel reduction can be simulated by sequences of single reduction steps (Lemma 12).

▶ **Definition 9** (ATRS). Let $\Sigma = (\mathcal{S}, \mathcal{T}, \mathcal{R}, \mathtt{dex}, \mathtt{duct}, \mathtt{ctx})$ be a tuple where:

- $\mathcal{S}$ is a signature.
- $\mathcal{T}$ is a set of terms over $\mathcal{S}$.
- $\mathcal{R}$ is a countable set whose elements are called **reductions**, and for any $r \in \mathcal{R}$:
  - $\mathtt{dex}(r)$ is a term called **redex** of r.
  - $\mathtt{duct}(r)$ is a term called **reduct** of r.
  - $\mathtt{ctx}(r)$ is a context called **context** of r.

For any $r \in \mathcal{R}$ call **source** (resp. **target**) the term $\mathtt{src}(r) = \mathtt{ctx}(r)[\mathtt{dex}(r)]$ (resp. $\mathtt{tgt}(r) = \mathtt{ctx}(r)[\mathtt{duct}(r)]$) and say $\mathtt{src}(r)$ reduces to $\mathtt{tgt}(r)$. Write $t \to t'$ when t reduces to t' and write $t \twoheadrightarrow t'$ if a sequence of reductions leads from t to t'. For any $r \in \mathcal{R}$ call **root** of r the position $\mathtt{root}(r)$ of the hole of $\mathtt{ctx}(r)$. Two reductions r_1, r_2 are called **equivalent** if $\mathtt{dex}(r_1) = \mathtt{dex}(r_2)$ and $\mathtt{duct}(r_1) = \mathtt{duct}(r_2)$. For any $t \in \mathcal{T}$ and $p \in pos(t)$ write $\mathcal{R}(t,p)$ the set of all the reductions $r \in \mathcal{R}$ such that $\mathtt{src}(r) = t$ and $\mathtt{root}(r) = p$.
Σ is an **Abstract Term Rewriting System** if the two following axioms are satisfied:
Source & Target: For any $r \in \mathcal{R}$, $\mathtt{src}(r) \in \mathcal{T}$ and $\mathtt{tgt}(r) \in \mathcal{T}$.
Residuals: Let $r_1 \in \mathcal{R}$ and $p \in pos(\mathtt{src}(r_1))$ such that $\mathtt{root}(r_1)$ and p are disjoint. Then for any $r_2 \in \mathcal{R}(\mathtt{src}(r_1), p)$ there is $r_2' \in \mathcal{R}(\mathtt{tgt}(r_1), p)$ equivalent to r_2. ◀

In any *ATRS* one can define a notion of parallel reduction that performs several disjoint reductions at one go.

▶ **Definition 10** (Parallel reduction). Let t be a term, and $\{r_1, ..., r_n\}$ a set of reductions of source t whose roots are pairwise disjoint. Write $t' = t[\mathtt{duct}(r_1)]_{\mathtt{root}(r_1)} ... [\mathtt{duct}(r_n)]_{\mathtt{root}(r_n)}$ the term t where each redex has been replaced by its reduct. Say t reduces parallely to t' and write $t \Rightarrow t'$. ◀

▶ **Example 11.** Let r_1 and r_2 be two reductions such that:

$$\begin{array}{rcl|rcl} \mathtt{dex}(r_1) & = & a & \mathtt{src}(r_2) & = & f(f(a,a),a) \\ \mathtt{duct}(r_1) & = & b & \mathtt{tgt}(r_2) & = & f(f(a,a),b) \\ \mathtt{ctx}(r_1) & = & f(f(a,\_),a) & \mathtt{root}(r_2) & = & 2 \end{array}$$

Then $f(f(a,a),a) \Rightarrow f(f(a,b),b)$ by parallel reduction of $\{r_1, r_2\}$. ◀

▶ **Lemma 12** (Simulation of parallel reduction). *In any* ATRS, *if* $t \Rightarrow t'$ *then* $t \twoheadrightarrow t'$.

Proof. By induction on the number of redexes reduced in parallel, with axiom *Residuals*. ◀

An *ATRS* can be extended with *effect zones* identifying in each reduction a part of the reduct containing all the positions that have possibly been created or modified by the reduction (Fig. 3). Axiom *Effect zone* states this property by contraposition: any subterm of the reduct which does not intersect the effect zone is syntactically equal to a strict subterm of the redex. This forms a *Marked ATRS*.

▶ **Definition 13** (Marked ATRS). Let $\Sigma = (\mathcal{S}, \mathcal{T}, \mathcal{R}, \texttt{dex}, \texttt{duct}, \texttt{ctx}, \texttt{effz})$ be a tuple where:

- $(\mathcal{S}, \mathcal{T}, \mathcal{R}, \texttt{dex}, \texttt{duct}, \texttt{ctx})$ is an *ATRS*.
- For any $r \in \mathcal{R}$, $\texttt{effz}(r)$ is an initial segment of $pos(\texttt{duct}(r))$, called **effect zone** of r.

Σ is a **Marked *ATRS*** if the following axiom is satisfied:

Effect zone: For any $r \in \mathcal{R}$ and $p \in pos(\texttt{duct}(r)) \setminus \texttt{effz}(r)$ there is $q \in pos(\texttt{dex}(r))$ such that $q \neq \epsilon$ and $\texttt{dex}(r)|_q = \texttt{duct}(r)|_p$. ◀

The condition $q \neq \epsilon$ ensures that any copy of the root symbol of the redex is in the effect zone, which forces the generation of new labels in the next section. This is required for the preservation of the so-called *independence* property (Definition 19, Lemma 20).

▶ **Example 14.** Let r be a reduction such that $\texttt{dex}(r) = g(f(a, b))$ and $\texttt{duct}(r) = h(g(f(a, b)), a)$. Any initial segment of $\{\epsilon, 1, 11, 111, 112, 2\}$ that includes $\{\epsilon, 1\}$ is an admissible effect zone for r. In particular, so are $\{\epsilon, 1\}$, $\{\epsilon, 1, 2\}$, and $\{\epsilon, 1, 11, 111, 112, 2\}$. See Section 4.2 for the impact of the choice of an effect zone. ◀

3.3 Sharing-via-Labelling Systems (*SvLS*)

We define *Sharing-via-Labelling Systems* as a specialization of *Marked ATRS* where the signature contains possibly labelled symbols and where the labels are partially ordered by a contribution relation $\hookrightarrow$.

Axiom *Redex head label* requires the head symbol of the redex of any reduction to be labelled. Axiom *Separation* prevents the labels of an effect zone from clashing. Axiom *Contribution* controls the progression of the labels along $\hookrightarrow$.

▶ **Definition 15** (Labelled signature). Let $\mathcal{S}$ be a signature, and $\mathcal{L}$ be a countable set whose elements are called **labels**. The **labelled signature** $\mathcal{S}[\mathcal{L}]$ is a signature defined by $\mathcal{S}[\mathcal{L}] = \mathcal{S} \cup \{f^\alpha | f \in \mathcal{S}, \alpha \in \mathcal{L}\}$. Let t be a term over a labelled signature $\mathcal{S}[\mathcal{L}]$, and $p \in pos(t)$. If $t(p) = f^\alpha$ with $f \in \mathcal{S}$ and $\alpha \in \mathcal{L}$, then write $\tau_p(t) = \alpha$ the **label** at position p in t. Else $\tau_p(t)$ is undefined. From now on, by writing $\tau_p(t)$ we suppose the label is defined. ◀

The sharing property is the main invariant we want to preserve:

▶ **Definition 16** (Sharing property). A term t on a labelled signature $\mathcal{S}[\mathcal{L}]$ has the **sharing property**, written $\mathbb{S}(t)$, when for any positions $p, q \in pos(t)$ if $\tau_p(t) = \tau_q(t)$ then $t|_p = t|_q$. ◀

▶ **Definition 17** (SvLS). Let $(\mathcal{L}, \hookrightarrow, \Sigma)$ be a tuple where:

- $\mathcal{L}$ is a countable set of labels.
- $\hookrightarrow$ is an irreflexive and transitive relation on $\mathcal{L}$ called **contribution relation**.
- $\Sigma = (\mathcal{S}[\mathcal{L}], \mathcal{T}, \mathcal{R}, \texttt{dex}, \texttt{duct}, \texttt{ctx}, \texttt{effz})$ is a *Marked ATRS* over a signature $\mathcal{S}[\mathcal{L}]$.

For any t, if there is $p \in pos(t)$ such that $\tau_p(t) = \alpha$ then we say α **occurs** in t. If moreover $t = \texttt{duct}(r)$ for some $r \in \mathcal{R}$ and $p \in \texttt{effz}(r)$ then we say α **occurs** in $\texttt{effz}(r)$.

$(\mathcal{L}, \hookrightarrow, \Sigma)$ is a **Sharing-via-Labelling System** if the following axioms are satisfied:

Redex head label: For any reduction $r \in \mathcal{R}$, the head label $\tau_\epsilon(\mathtt{dex}(r))$ is defined. Write $\tau(r)$ as a shorthand for $\tau_\epsilon(\mathtt{dex}(r))$.

Separation: For any $r \in \mathcal{R}$ such that $\mathbb{S}(\mathtt{dex}(r))$, for any $p, q \in \mathtt{effz}(r)$, if $\tau_p(\mathtt{duct}(r)) = \tau_q(\mathtt{duct}(r))$, then $\mathtt{duct}(r)|_p = \mathtt{duct}(r)|_q$.

Contribution: For any $r \in \mathcal{R}$ and $\beta \in \mathcal{L}$ occurring in $\mathtt{effz}(r)$ one has $\tau(r) \hookrightarrow \beta$. Moreover for any $\alpha \in \mathcal{L}$, if $\alpha \neq \tau(r)$ and $\alpha \hookrightarrow \beta$ then there is $\gamma \in \mathcal{L}$ in $\mathtt{dex}(r)$ such that $\alpha \hookrightarrow \gamma$.

A label $\alpha \in \mathcal{L}$ is **initial** if there is no $\beta \in \mathcal{L}$ such that $\beta \hookrightarrow \alpha$. A term t is **initial** if all its labels are initial and different. ◀

Remark that $\to$ does not preserve the sharing property: only the parallel labelled reduction defined below and the corresponding reduction sequences have a graphical meaning. The parallel labelled reduction simulating graph reduction is the simultaneous reduction of all the redexes with a given label.

▶ **Definition 18** (Parallel labelled reduction). Let t be a term in a *SvLS* such that $\mathbb{S}(t)$, and α be a label in t. Write $P_\alpha = \{q \in pos(t) \mid \tau_q(t) = \alpha\}$. By $\mathbb{S}(t)$ remark that the positions in P_α are pairwise disjoint. Let $r \in \mathcal{R}$ with $\tau(r) = \alpha$ such that for any $p \in P_\alpha$ there is $r_p \in \mathcal{R}(t, p)$ equivalent to r. Then t reduces parallely to $t' = t[\mathtt{duct}(r)]_{P_\alpha}$ (Definition 10), which is written $t \Rightarrow_\alpha t'$ and called parallel labelled reduction. ◀

The proof of the preservation of the sharing property requires an invariant of $\Rightarrow_\alpha$:

▶ **Definition 19** (Independence property). A term t in a *SvLS* has the **independence property**, written $\mathbb{I}(t)$, when there are no two labels α and β in t such that $\alpha \hookrightarrow \beta$. ◀

▶ **Lemma 20** (Independence). *In any* SvLS*, if* $\mathbb{S}(t)$*,* $\mathbb{I}(t)$ *and* $t \Rightarrow_{\alpha_0} t'$*, then* $\mathbb{I}(t')$*.*

Proof. By definition, there is a reduction r such that $\tau(r) = \alpha_0$ and the parallel reduction uses $\mathtt{dex}(r)$ and $\mathtt{duct}(r)$ as unique redex and reduct. Let α, β be two labels in t' s.t. $\alpha \hookrightarrow \beta$.

- If β occurs in t, then by $\mathbb{I}(t)$ there is no occurrence of α in t. Hence α occurs in $\mathtt{effz}(r)$ and by axiom *Contribution* $\alpha_0 \hookrightarrow \alpha$. By transitivity $\alpha_0 \hookrightarrow \beta$, which contradicts $\mathbb{I}(t)$.
- If there is no occurrence of β in t, then β occurs in $\mathtt{effz}(r)$.
  - If α occurs in t, then case on $\alpha \hookrightarrow \beta$ with axiom *Contribution*:
    * Either $\alpha = \alpha_0$, which implies that α_0 occurs in t'. Then by definition of parallel labelled reduction α_0 occurs in $\mathtt{duct}(r)$, and by axiom *Effect zone* α_0 occurs in $\mathtt{effz}(r)$. Hence by axiom *Contribution* $\alpha_0 \hookrightarrow \alpha_0$, which contradicts irreflexivity.
    * Or there is γ in $\mathtt{dex}(r)$ such that $\alpha \hookrightarrow \gamma$, which contradicts $\mathbb{I}(t)$.
  - If there is no occurrence of α in t, then α occurs in $\mathtt{effz}(r)$ and in particular $\alpha_0 \hookrightarrow \alpha$. Moreover $\alpha \neq \alpha_0$, hence by axiom *Contribution* there is γ in $\mathtt{dex}(r)$ s.t. $\alpha \hookrightarrow \gamma$. By transitivity $\alpha_0 \hookrightarrow \gamma$, which contradicts $\mathbb{I}(t)$. ◀

▶ **Theorem 21** (Sharing). *In any* SvLS*, if* $\mathbb{S}(t)$*,* $\mathbb{I}(t)$ *and* $t \Rightarrow_{\alpha_0} t'$*, then* $\mathbb{S}(t')$*.*

Proof. As above there is r with $\tau(r) = \alpha_0$ such that the parallel reduction uses $\mathtt{dex}(r)$ and $\mathtt{duct}(r)$ as unique redex and reduct. Let $p', q' \in pos(t')$. Suppose $\tau_{p'}(t') = \tau_{q'}(t') = \alpha$.

- If none of p', q' is in the effect zone of a reduced redex, then by axiom *Effect zone* there is p (resp. q) in $pos(t)$ such that $t(p) = t'(p')$ (resp. $t(q) = t'(q')$). In particular $\tau_p(t) = \tau_q(t) = \alpha$, and $t|_p = t|_q$ by $\mathbb{S}(t)$. Write $P = \{p_1 \mid \tau_{p_1}(t|_p) = \alpha_0\}$ and $Q = \{q_1 \mid \tau_{q_1}(t|_q) = \alpha_0\}$ and remark that $P = Q$. By definition $t'|_{p'} = (t|_p)[\mathtt{duct}(t)]_P = (t|_p)[\mathtt{duct}(r)]_Q = t'|_{q'}$.
- If both p' and q' are in the effect zone of a reduced redex, then by unicity of $\mathtt{duct}(r)$ and by axiom *Separation* $t'|_{p'} = t'|_{q'}$.

- Else, wlog. p' is in the effect zone of a reduced redex and $\alpha_0 \hookrightarrow \alpha$ (axiom *Contribution*) and q' is not in the effect zone of a reduced redex and α occurs in t (axiom *Effect zone*). This contradicts $\mathbb{I}(t)$. ◀

Remark that any initial term t_i satisfies $\mathbb{S}(t_i)$ and $\mathbb{I}(t_i)$. Then Independence lemma 20 and Sharing theorem 21 have an immediate corollary:

▶ **Corollary 22** (Long-term sharing). *In any* SvLS*, if t_i is an initial term and there is a sequence of parallel labelled reductions from t_i to a term t, then $\mathbb{S}(t)$.* ◀

4 First examples

This section presents some simple *Sharing-via-Labelling Systems* and gives useful definitions to handle binders: Subsection 4.1 formalizes the example of Section 2.2, then Subsection 4.2 shows three sharing disciplines for the (unrestricted) λ-calulus.

4.1 The fixpoint operator, formally

Let $\mathcal{X}$ and L be countable sets of variables and initial labels. We consider the signature $\{a, f, g, h\}$ used in Section 2.2. Define:

$$\begin{array}{llcll} \text{Labels } (\mathcal{L}): & \alpha & ::= & \mathfrak{a} \mid [\alpha] \mid [\alpha]\alpha & \mathfrak{a} \in L \\ \text{Terms } (\mathcal{T}_\mu): & t & ::= & x \mid a \mid \mu x^\alpha.t \mid F^\alpha(t_1, ..., t_n) & x \in \mathcal{X},\ F \in \{f, g, h\} \end{array}$$

Free variables $fv(t)$ and bound variables $bv(t)$ of a term t are defined as usual, remembering that $\mu x^\alpha.u$ binds x in u. For any term t and variable x, call **spine** of x in t the initial segment of $pos(t)$ whose elements are the strict prefixes of the positions of the free occurrences of x in t. These are exactly the positions of the subterms of t that would be affected by a substitution of x. By abuse of language, the "labels of a spine" denote the labels at the positions of a spine. The **labelled substitution** $t^{\{x:=\omega,u\}}$ substitutes u for x in t and modifies the labels of the spine of x in t using the label ω. If $x \notin fv(t)$ then $t^{\{x:=\omega,u\}} = t$, else:

$$\begin{array}{rcll} x^{\{x:=\omega,u\}} & = & u & \\ (\mu y^\alpha.t)^{\{x:=\omega,u\}} & = & \mu y^{[\omega]\alpha}.t^{\{x:=\omega,u\}} & y \notin fv(u) \\ (F^\alpha(t_1, ..., t_n))^{\{x:=\omega,u\}} & = & F^{[\omega]\alpha}(t_1^{\{x:=\omega,u\}}, ..., t_n^{\{x:=\omega,u\}}) & F \in \{f, g, h\} \end{array}$$

Remark that this substitution without capture may be undefined in general. See Section 4.2. Define the reduction rule $(\mu)\ :\ \mu x^\omega.t\ \to\ t^{\{x:=\omega,\mu x^{[\omega]}.t\}}$. The corresponding reduction relation is the smallest congruence on terms generated by (μ).

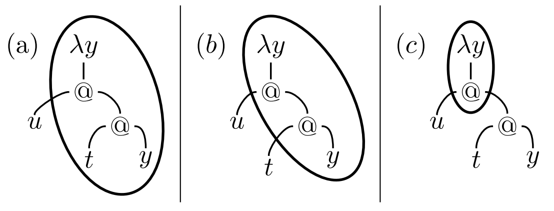
Axiomatic checking The application of the rule (μ) to a term $\mu x^\omega.t$ in any context c defines an obvious reduction r in redex-reduct-context format in a way that satisfies the *ATRS* axioms. Write $t' = t^{\{x:=\omega,\mu x^{[\omega]}.t\}}$. The effect zone of r is the union of the spine of x in t and the occurrences of $\mu x^{[\omega]}$. Hence the positions in t' that are outside of the effect zone are subterms of t, which verifies the axiom *Effect zone*.

Define $\hookrightarrow$ as the least transitive relation such that $\alpha \hookrightarrow [\alpha]$ and $\alpha \hookrightarrow [\alpha]\beta$ for any $\alpha, \beta \in \mathcal{L}$. It is irreflexive. Axiom *Redex head label* is satisfied since all occurrences of μ are labelled. Axiom *Contribution* is satisfied since all labels in the effect zone have the form $[\omega]$ or $[\omega]\alpha$ with ω the label of the redex.

Axiom *Separation*: suppose $\mathbb{S}(\mu x^\omega.t)$, and α occurs twice in the effect zone, at two positions p and q of the reduct. If $\alpha = [\omega]$ then $t'|_p = t'|_q = \mu x^{[\omega]}.t$. Else $\alpha = [\omega]\beta$ for some β occurring at positions p and q in t. Hence, since $\mathbb{S}(t)$ by hypothesis, $t|_p = t|_q$, and $t'|_p = t|_p^{\{x:=\omega,\mu x^{[\omega]}.t\}} = t|_q^{\{x:=\omega,\mu x^{[\omega]}.t\}} = t'|_q$.

4.2 The λ-calculus: β- and α-reduction

We consider the λ-calculus, with the usual notions of free and bound variables, and the notions of spine and labelled substitution adapted from above. We also use L, $\mathcal{L}$ and $\hookrightarrow$ as above, and we label applications and abstractions. Let $(\lambda x.\lambda y.x(ty))u \to \lambda y.u(ty)$ be a reduction where t, u are two terms in which neither x nor y appears free. Three of the possible effect zones are circled below. The effect zone is arbitrary as soon as it contains the spine of x in $\lambda y.x(ty)$, which is circled in case (c).



(a) duplicating the whole body of the function is usual lazy evaluation; (b) avoiding the copy of the free expressions (t here) is Wadsworth's algorithm [29]; (c) duplicating only the spine of x is found in [10].

While these examples have been originally defined with weak reduction strategies, more general strategies are known to allow shorter reductions [17], thus we consider here general β-reduction. As a consequence α-conversion is sometimes needed to avoid blocked terms.

Remark that in our framework only the symbols of the effect zone are allowed to be modified, hence any renaming operation has to be contained in the effect zone. In cases (a) and (b) there is no problem: the effect zone is always large enough. In case (c) there is a solution: add a reduction rule for renaming, with appropriate creations of labels.

$$\text{Renaming:} \qquad \lambda x^{\alpha}.t \quad \to \quad \lambda y^{[\alpha]}.t^{\{x:=\alpha,y\}} \qquad y \notin \mathit{fv}(t) \cup \mathit{bv}(t)$$

The effect zone of this reduction is $\epsilon \cdot P$ where P is the spine of x in t. Checking the axioms for this new rule as well as for β-reduction is similar to Section 4.1. Remark that renaming can also be inlined in β, with an appropriate extension of the effect zone.

The need for this new rule and the duplications it implies comes from a new behaviour brought by case (c): sharing-via-labelling allows the sharing of open subterms bound by different binders. Note that this is not allowed in usual higher-order termgraphs [11, 19]. Section 5 develops this point and illustrates how one can take advantage of it.

5 A significant example

This section presents a labelling of a calculus inspired by the *call-by-need λ-calculus* [3]. It emphasizes how the graphs represented by *SvLS* are more general than usual higher-order termgraphs, and takes advantage of this point to present a reduction strategy performing a more-than-fully-lazy sharing.

In the call-by-need λ-calculus $\lambda_{\texttt{let}}$, the β-rule is replaced by the introduction of a `let`-construct that delays the substitution of the argument: $(\lambda x.t)u \;\to\; \texttt{let}\ x = u\ \texttt{in}\ t$.
From the point of view of the argument u, this keeps only one shared occurrence of u for possibly numerous corresponding occurrences of x in t.

Now, in our context the labels can take care of this sharing already: the substitution of u does not need modifying the labels of u. This safeguard allows us to relax most of the original rules of $\lambda_{\texttt{let}}$ and use the delay introduced by `let` in a new way. Indeed, in `let` $x = u$ `in` t, from the point of view of t, the `let` makes the value of x unspecified. This allows us to share multiple copies of t bound by different `let`-constructs with different arguments.

Let $\mathcal{X}$, L, $\mathcal{L}$ and $\hookrightarrow$ be as defined above. We define below the labelled terms, where applications and `let`-constructs are labelled (labelling λ-abstractions is possible but not needed). The answers represent λ-abstractions under `let`-constructs. Evaluation focuses are contexts that guide call-by-need shared reduction by pointing one of the redexes to be reduced parallely.

$$\begin{array}{llcll}
\text{Terms } (\mathcal{T}_{\texttt{let}}): & t & ::= & x \mid \lambda x.t \mid @^{\alpha}(t,t) \mid \texttt{let}^{\alpha}\ x = t\ \texttt{in}\ t & x \in \mathcal{X} \\
\text{Answers } (\mathcal{A}_{\texttt{let}}): & A & ::= & \lambda x.t \mid \texttt{let}^{\alpha}\ x = t\ \texttt{in}\ A & x \in \mathcal{X} \\
\text{Eval. focus } (\mathcal{E}_{\texttt{let}}): & E & ::= & \_ \mid @^{\alpha}(E,t) \mid \texttt{let}^{\alpha}\ x = t\ \texttt{in}\ E & x \in \mathcal{X}
\end{array}$$

The application is explicited by a binary constructor @ to ease the reading of labelled terms. Free and bound variables are defined as usual, remembering that both $\lambda x.t$ and `let` $x = u$ `in` t bind x in t. Spines and labelled substitutions are adapted from above.

Two main rules (Intro) and (Subst) respectively introduce a `let`-construct to delay the substitution of a β-reduction and perform substitution when the focus of the reduction encounters a variable. The (Scope) rule prevents `let`-constructs from hiding β-redexes. A (GC) rule is added to lighten the examples, and (Renaming) is as discussed in Section 4.2. The system is not orthogonal: (GC) and (Renaming) introduce several critical pairs. The reduction relation is the smallest congruence on terms generated by the rules.

$$\begin{array}{lrcll}
\text{Intro:} & @^{\alpha}(\lambda x.t, u) & \to & \texttt{let}^{[\alpha]}\ x = u\ \texttt{in}\ t & \\
\text{Subst:} & \texttt{let}^{\alpha}\ x = t\ \texttt{in}\ E[x] & \to & \texttt{let}^{[\alpha]}\ x = t\ \texttt{in}\ E[y]^{\{y := \alpha, t\}} & y \text{ fresh} \\
\text{Scope:} & @^{\alpha}(\texttt{let}^{\beta}\ x = t\ \texttt{in}\ A,\ u) & \to & \texttt{let}^{[\alpha]\beta}\ x = t\ \texttt{in}\ @^{[\alpha]}(A, u) & x \notin fv(u) \\
\text{GC:} & \texttt{let}^{\alpha}\ x = t\ \texttt{in}\ u & \to & u & x \notin fv(u) \\
\text{Renaming:} & \texttt{let}^{\alpha}\ x = t\ \texttt{in}\ u & \to & \texttt{let}^{[\alpha]}\ y = t\ \texttt{in}\ u^{\{x := \alpha, y\}} & y \notin fv(u) \cup bv(u)
\end{array}$$

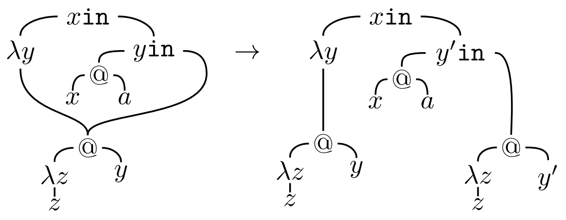
Remark that in rule (Subst) only one occurrence of the variable is substituted and the labels are modified along the substitution.

In the sequence below we follow the underlined subterm $\underline{@^{\delta}(\lambda z.z, y)}$. In usual call-by-need the two copies are separated from step (1), as the value of the first occurrence of x is required. Even with the fully lazy version given in [2], only the duplication of $\lambda z.z$ would be spared, which would not help here. In this example however, the two copies remain shared thanks to the label δ (and later $[\delta]$). They are separated only at the end of step (5). Inbetween, steps (3), (4) and (5) evaluate in parallel the two copies. As a consequence, the function substituted for the second occurrence of x in step (6) is already fully evaluated. The interesting new behaviour observed here is the sharing of two subterms occurring at very different positions: one is buried under a λ-abstraction while the other stands in the evaluation focus.

$$\begin{array}{rll}
 & @^{\alpha}(\ \lambda x.@^{\beta}(x, @^{\gamma}(x,a)),\ \lambda y.\underline{@^{\delta}(\lambda z.z, y)}\ ) & \\
\to_I & \texttt{let}^{[\alpha]}\ x = \lambda y.\underline{@^{\delta}(\lambda z.z, y)}\ \texttt{in}\ @^{\beta}(x, @^{\gamma}(x,a)) & \\
\to_{Su} & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{@^{\delta}(\lambda z.z, y)}\ \texttt{in}\ @^{[[\alpha]]\beta}(\lambda y.\underline{@^{\delta}(\lambda z.z, y)}, @^{\gamma}(x,a)) & (1) \\
\to_I & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{@^{\delta}(\lambda z.z, y)}\ \texttt{in}\ \texttt{let}^{[[[\alpha]]\beta]}\ y = @^{\gamma}(x,a)\ \texttt{in}\ \underline{@^{\delta}(\lambda z.z, y)} & (2) \\
\Rightarrow_I & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{\texttt{let}^{[\delta]}\ z = y\ \texttt{in}\ z}\ \texttt{in}\ \texttt{let}^{[[[\alpha]]\beta]}\ y = @^{\gamma}(x,a)\ \texttt{in}\ \underline{\texttt{let}^{[\delta]}\ z = y\ \texttt{in}\ z} & (3) \\
\Rightarrow_{Su} & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{\texttt{let}^{[\delta]}\ z = y\ \texttt{in}\ y}\ \texttt{in}\ \texttt{let}^{[[[\alpha]]\beta]}\ y = @^{\gamma}(x,a)\ \texttt{in}\ \underline{\texttt{let}^{[\delta]}\ z = y\ \texttt{in}\ y} & (4) \\
\Rightarrow_{GC} & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{y}\ \texttt{in}\ \texttt{let}^{[[[\alpha]]\beta]}\ y = @^{\gamma}(x,a)\ \texttt{in}\ \underline{y} & (5) \\
\to^{Su}_{GC} & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{y}\ \texttt{in}\ @^{\gamma}(x,a) & \\
\to^{Su}_{GC} & @^{[[[\alpha]]\gamma]}(\lambda y.\underline{y})a & (6)
\end{array}$$

The labelled term obtained at step (2) and its corresponding graph are represented below. They show how sharing-via-labelling allows the sharing of open subterms affected by different bindings. This kind of graph is usually called *non-admissible* [29], but in our case the labelled term clearly shows that there is no binding ambiguity, and thus that using this graph is possible. The point is: in this setting α-conversion cannot be silent. Indeed, renaming one occurrence of y requires breaking the sharing and modifying one of the occurrences of the label δ.

$$\begin{array}{rl} & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{@^{\delta}(\lambda z.z, y)} \\ & \texttt{in let}^{[[[\alpha]]\beta]}\ y = @^{\gamma}(x, a) \\ & \quad \texttt{in } \underline{@^{\delta}(\lambda z.z, y)} \\ \\ \to & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{@^{\delta}(\lambda z.z, y)} \\ & \texttt{in let}^{[[[[\alpha]]\beta]]}\ y = @^{\gamma}(x, a) \\ & \quad \texttt{in } \underline{@^{[[[[\alpha]]\beta]]\delta}(\lambda z.z, y)} \end{array}$$



In this case the "physical identity" of one copy of the shared redex $@^{\delta}(\lambda z.z, y)$ is modified, which illustrates the *dynamic* identification mentioned in introduction.

Axiomatic checking Define a reduction as the application of any rule in any context. The axioms of *ATRS* are satisfied. Define the effect zone of a reduction as the set of the positions where the labels are changed. Remark that the collapsing rule (GC) has an empty effect zone, while the effect zones of (Subst.) and (Renaming) correspond to spines. The axiom *Effect zone* is satisfied. The axioms of *SvLS* are satisfied as in Section 4.1.

▶ Remark. The labelled substitution extends implicitly the right-hand-sides of the rules, hence this system does not fit directly in usual (higher-order) term rewriting frameworks. This is fixed as soon as the rule schemes explicitly mention the spines, as it is done in [7]. ◀

6 Related work

Sharing and graph rewriting have already been described in many different ways, some of which are reviewed here. We compare them with our framework along three directions:
Expressive power: the amount of compatible rewriting systems and their features.
Sharing power: the level of sharing achieved on compatible systems.
Formal complexity: the simplicity or complexity of defining and handling the system.

Adressed TRS D. Dougherty et al. [13] propose labelled first-order systems with a similar sharing power. They allow non-orthogonality, and also cycles representing infinite terms, which we do not here. They do not however mention higher-order systems. Their technology is slightly more complex than term rewriting: the new labels are taken globally fresh, thus the reduction of a redex depends on its global context and parallel reduction is not decomposed.

Term graphs Term graphs [9, 27, 19, 8] provide sharing facilities for any first-order or higher-order system, with an equivalent sharing power in many cases, and also allow cycles [8]. They have a far higher formal complexity, since they require subtle correctness criteria and notions of morphisms of graphs with binders. They forbid the sharing of open subterms affected by different binders, which limits the reduction strategies with sharing (Section 5).

Call by need The call-by-need λ-calculus [3] and its fully lazy version [2] give a simple and elegant account of Wadsworth's graph algorithm for the evaluation of the λ-calculus [29]. N. Yoshida [30] also develops similar ideas by using explicit substitutions in place of `let`-constructs. These works also manage to use only terms and one-step term reduction. On the other hand they require structural rules or structural equations. The latter point however has been recently improved in [12]. Concerning the expressive power, these works are a study of the particular case of the λ-calculus, which is orthogonal.

Optimal sharing The optimal implementations of higher-order systems [23, 22, 4, 5] perform a better sharing than our framework, and apply to a large class of orthogonal higher-order systems, through Interaction Systems [5]. They do not apply however to all orthogonal higher-order systems, and do not apply at all to non-orthogonal systems. They also achieve optimal sharing at the cost of a far greater formal complexity.

Interaction nets Interaction nets can implement wide classes of first-order and higher-order rewriting systems [15, 16, 24]. They are known in particular for the suboptimal yet powerful sharing they realize in systems like [24]. The limit of their expressive power is their sequential nature. Also their atomization of every operation makes the link with the term specifications technically very demanding.

7 Conclusion

The axiomatic sharing-via-labelling systems presented in this paper provide a general framework for the sharing of subterms in term rewriting. This very expressive framework applies to higher-order and non-orthogonal rewriting systems and covers a wide range of shared reduction algorithms based on acyclic graphs. The formalism is simpler than most of the other approaches to shared rewriting since it is definable within the realm of term rewriting.

The framework generalizes the previous label-based approaches to sharing inspired by optimality theory [25, 10, 6, 7] by dropping the causal content of the labels. This makes it possible to eliminate their restriction to weak reduction as well as their need for indirection nodes, and to define more efficient shared reduction strategies. Section 5 in particular shows this, illustrating the power of ingredients as simple as term rewriting and acyclic sharing.

This work opens new perspectives for rewriting theory and functional programming:

Study of general term rewriting The *Abstract Term Rewriting Systems (ATRS)* are an intermediate formalism between fully abstract rewriting frameworks and concrete term rewriting frameworks. This granularity makes it possible to grasp specific properties of term rewriting that are common to all term frameworks.

Specification of graph rewriting Sharing-via-labelling provides a simple approach to graph rewriting, which is now applicable on a large scale.

Analysis of functional programming Our work provide a homogeneous framework for comparing shared implementations of functional programming languages, in the line of [7].

Implementation of functional programming Sharing-via-labelling provides means to imagine and express new graph algorithms and new reduction strategies for the implementation of functional programming languages, as illustrated in Section 5.

Acknowledgements I would like to thank Delia Kesner for her good advice and constant support, as well as the anonymous referees for their helpful comments and suggestions.

References

1 M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.

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

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

4 A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages.* Cambridge University Press, 1998.

5 A. Asperti and C. Laneve. Interaction systems ii: The practice of optimal reductions. *TCS*, 159(2):191–244, 1996.

6 T. Balabonski. Optimality for dynamic patterns. In *PPDP'10*, pages 231–242, 2010.

7 T. Balabonski. A unified approach to fully lazy sharing. In *POPL'12*, 2012.

8 P. Baldan, C. Bertolissi, H. Cirstea, and C. Kirchner. A rewriting calculus for cyclic higher-order term graphs. *Mathematical Structures in Computer Science*, 17(3):363–406, 2007.

9 H. P. Barendregt, M. C. J. D. van Eekelen, J. R. W. Glauert, R. Kennaway, M. J. Plasmeijer, and M. R. Sleep. Term graph rewriting. In *PARLE (2)*, pages 141–158, 1987.

10 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.

11 S. Blom. *Term Graph Rewriting - Syntax and Semantics.* Ph.D. thesis, 2001.

12 S. Chang and M. Felleisen. The call-by-need lambda calculus, revisited. In *ESOP*, 2012.

13 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.

14 M. Fernández, I.Mackie, and F.-R. Sinot. Closed reduction: explicit substitutions without alpha-conversion. *MSCS*, 15(2):343–381, 2005.

15 M. Fernández and I. Mackie. Interaction nets and term-rewriting systems. *TCS*, 190(1):3–39, 1998.

16 M. Fernández, I. Mackie, S. Sato, and M. Walker. Recursive functions with pattern matching in interaction nets. *ENTCS*, 253(4):55–71, 2009.

17 C.K. Holst and D.K. Gomard. Partial evaluation is fuller laziness. *SIGPLAN Not.*, 26(9):223–233, 1991.

18 C.B. Jay and D. Kesner. First-class patterns. *J. Funct. Program.*, 19(2):191–225, 2009.

19 W. Kahl. Relational treatment of term graphs with bound variables. *Logic Journal of the IGPL*, 6(2):259–303, 1998.

20 D. Kesner. The theory of calculi with explicit substitutions revisited. In *CSL*, pages 238–252, 2007.

21 U. Dal Lago and S. Martini. On constructor rewrite systems and the lambda-calculus. In *ICALP'09*, pages 163–174, 2009.

22 J. Lamping. An algorithm for optimal lambda calculus reduction. In *POPL*, 1990.

23 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.

24 I. Mackie. Efficient lambda-evaluation with interaction nets. In *RTA*, pages 155–169, 2004.

25 L. Maranget. Optimal Derivations in Weak Lambda-calculi and in Orthogonal Terms Rewriting Systems. In *POPL'91*, pages 255–269, 1991.

26 P.-A. Melliès. *Description Abstraite des Systèmes de Réécriture.* Ph.D. thesis, 1996.

27 D. Plump. Term graph rewriting. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 3–61, 1999.

28 Terese. *Term Rewriting Systems.* Cambridge University Press, 2003.

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

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