Term-graph rewriting via explicit paths

Emilie Balland, Claude Kirchner, Pierre-Etienne Moreau

Term-graph rewriting via explicit paths

Emilie Balland, Claude Kirchner, and Pierre-Etienne Moreau

UHP & LORIA, INRIA & LORIA, and INRIA & LORIA,
BP 101, 54602 Villers-lès-Nancy Cedex France
{Emilie.Balland,Claude.Kirchner,Pierre-Etienne.Moreau}@loria.fr

Abstract. The notion of path is classical in graph theory but not directly used in the term rewriting community. The main idea of this work is to raise the notion of path to the level of first-order terms, *i.e.* paths become *part* of the terms and not just meta-information about them. These paths are represented by sequences of integers (positive or negative) and are interpreted as relative addresses in terms. In this way, paths can also be seen as a generalization of the classical notion of position for the first-order terms and of de Bruijn indexes for the lambda calculus.
In this paper, we define an original framework called Addressed Term Rewriting where paths are used to represent pointers between subterms. Using this approach, any term-graph rewriting systems can be efficiently simulated using a rewrite-based environment.

1 Introduction

The notion of position is of course central as soon as one deals with data structures. Absolute as well as relative positions are at the heart of algorithms manipulating data structures and their appropriate use and representation can lead to very important differences in the complexity behavior and more generally in the expression of the algorithms themselves. A typical example is the notion of de Bruijn indexes for lambda-terms [10] that not only allows for an easier expression of data-structure manipulations, typically substitution, but also completely changes the way the algorithm is designed, because in this case alpha-conversion is useless.

The main idea of this paper is to raise the notion of path to the level of first-order terms, *i.e.* paths become *part* of the terms and not just meta-information about them. Paths are defined as a generalization of positions and denote a relation from a source position to a target one. A main difference with classical positions that specify a subterm *w.r.t.* to a global term is that the source position is not necessarily the root.

The first contribution of the paper is to introduce the notion of addressed terms to ground an extension of term rewriting where paths are used to express relative addresses and thus to provide a natural way to add pointers in classical terms. For instance, the term $f(a, g(a))$ where we want to make explicit the fact that the subterm a is shared, will be represented as the addressed term $f(a, g(\text{-}1 \cdot \text{-}2 \cdot 1))$. The rational term g^ω, solution of the regular equation [9] $x = g(x)$, is represented by the addressed term $g(\text{-}1)$.

Based on the formalization of paths and a notion of rewriting for addressed terms, the main contribution of this paper is to establish a simulation of term-graph rewriting by addressed terms. The main advantage of such an approach is to offer an efficient way to implement in any rule-based language term-graph rewriting features. In fact, since the simulation is completely based on standard first-order terms, this extension is non-intrusive. Indeed this grounds our implementation of explicit sharing in Tom [4].

This new representation of terms generalizes both standard first-order terms as well as de-Bruijn representation of lambda-terms. It requires us to carefully design this new notion of terms and its use to get syntactic correctness (*e.g.* $g(\text{-}1\,.\,\text{-}1)$ makes no sense as such) as well as completeness with respect to the standard notions of term and term graph rewritings. It leads to an original and clean way for representing and transforming graphs in a maximally shared rewrite-based environment, making in particular possible the use of term rewriting strategy [14, 18] for term graph rewriting.

The paper is organized as follows. In Section 2, we formalize the notion of paths and explore some relevant operations and properties. Section 3 focuses on the concept of addressed term rewriting where paths are interpreted as pointers. Section 4 shows the relation between addressed terms and cyclic term-graphs. In this section, paths are considered as a way to identify shared parts of the term. Sections 5 and 6 present related work and conclude.

2 Paths in a term

We assume the reader to be familiar with the basic definitions of first-order terms given, in particular, in [2]. We briefly recall or introduce notations for a few concepts that will be used throughout this paper.

A signature $\mathcal{F}$ is a set of function symbols, each one associated to a natural number by the arity function. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the set of *terms* built from a given finite set $\mathcal{F}$ of function symbols and a denumerable set $\mathcal{X}$ of variables. $\mathcal{S}ymb(t)$ is a partial function from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ to $\mathcal{F}$, which associates to each term t its top-symbol $f \in \mathcal{F}$. The set of variables occurring in a term t is denoted by $\mathcal{V}ar(t)$. If $\mathcal{V}ar(t)$ is empty, t is called a *ground term* and $\mathcal{T}(\mathcal{F})$ is the set of ground terms. A *substitution* σ is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F})$, written, when its domain is finite, $\sigma = \{x_1 \mapsto t_1, \ldots, x_k \mapsto t_k\}$. The codomain is denoted $\mathcal{R}an(\sigma)$

In term rewriting, the concept of positions is used to denote a subterm in a global term (*i.e.* the path from the root to this subterm). They are represented by sequences of positive integers and denoted by greek letters ω, δ. The empty sequence ϵ denotes the position associated to the root, and is called top-position. $\sqsubseteq$ is the classical relation of prefixation between two sequences ($\omega_1 \sqsubseteq \omega_2$ if ω_1 is prefix of ω_2). The subterm of t at position ω is denoted $t_{|\omega}$. The replacement at position ω of the subterm $t_{|\omega}$ by t' is written $t[t']_\omega$. The set of positions of t is denoted $\mathcal{P}os(t)$.

In this section, we define the notion of *path*, which generalizes the notion of position by denoting a path from a subterm to another one, and not only the

path from the root to a given subterm. Negative numbers designate bottom-up displacements whereas positive numbers mean top-down ones.

Definition 2.1 (Path). *The set $\mathcal{P}$ of paths is the monoid $(\mathbb{Z} \setminus \{0\}, .)$ with neutral element ϵ.*

Example 2.1. ϵ, $1 . 3$, and $\text{-}2 . 1 . 3$ are paths. Note that positions are paths only composed of positive integers.

In graph theory, there exists a notion of *path algebras* [6] used to express optimality problems. A path algebra is composed of a set and two operations specific to a given optimality problem. Solving this problem consists in computing with adjacency matrices over the corresponding algebra. Our definition of paths is specific to terms and there is no connection with optimality resolution. Although it would be interesting to instantiate the framework of path algebras with the ideas introduced in this section.

The notion of path is oriented and corresponds to the route from a subterm to another one. For example, considering the term $f(a, b)$, the path $\text{-}1 . 2$ describes how to reach the subterm b starting from a. The negative integer -1 denotes a backward move from a to f, whereas 2 goes from f to b. Given a path, we consider the *inverse*, denoted by $\overline{p}$, and specified by the equations $\overline{\epsilon} = \epsilon$ and $\overline{i . p} = \overline{p} . \text{-}i$. For example, $\overline{\text{-}1 . 2} = \text{-}2 . 1$. We also consider the *subtraction*, denoted by $p_1 - p_2$, that is just an alias for $\overline{p_2} . p_1$. Given two positions s and d (for *source* and *destination*), the result of $d - s$ corresponds to the path connecting s to d. The subtraction is neither associative, nor commutative.

The paths $1 . 2 . \text{-}2$ and 1 should be equivalent because for every source position, the target positions are the same. In the following we define the notion of *canonical form* as the smallest path of this equivalence class. Moreover, when interpreting a negative integer as a backward move from the i-th child to the father, we must ensure that the previous move in the sequence led to the same i-th child. For example, the path $1 . \text{-}2$ cannot be considered as valid because a move downward to its first child is followed by a move backward from its second child. These observations lead us to introduce the notions of canonical forms, well-formed paths, as well as a constant $\bot$ for representing ill-formed paths.

Definition 2.2 (Canonical path and path equivalence). *The canonical form denoted by $(\!|p|\!)$ is obtained by normalizing p with the following rules, for all natural i and path p:*

$$i . \text{-}i \to \epsilon \qquad \text{-}i . i \to \epsilon \qquad i . \text{-}j \to \bot \text{ if } i \neq j$$
$$\epsilon . p \to p \qquad p . \epsilon \to p \qquad \bot . p \to \bot \qquad p . \bot \to \bot$$

Two paths p_1 and p_2 are said equivalent if $(\!|p_1|\!) = (\!|p_2|\!)$.

Property 2.1. Given $p \in \mathcal{P}$, the following equalities hold:

- $\overline{\overline{p}} = p$
- $(\!|p - p|\!) = (\!|\overline{p} . p|\!) = (\!|p . \overline{p}|\!) = \epsilon$.

Let us remark that given two positions ω_1, ω_2, which are sequences of positive integers, we have $(\!|\omega_2 - \omega_1|\!) = p \,.\, \omega'$ where p is a sequence of negative integers and ω' a sequence of positive integers.

Definition 2.3 (Well-formed path). *A path $p \in \mathcal{P}$ is well-formed if $(\!|p|\!) \neq \bot$.*

For example, $1\,.\,2\,.$-2 and $2\,.\,3\,.$-3$\,.$-2$\,.\,1$ are well-formed paths, but $1\,.$-2 is not. Notice that positions can be seen as a subset of well-formed paths because they correspond to paths only composed of positive integers. Recall that the inverse preserves the well-formedness and every subsequence of a well-formed path is also well-formed. On the other hand, the concatenation and thus the subtraction of two well-formed paths do not always lead to a well-formed path: 1 is well-formed, -2 is well-formed, but $1\,.$-2 is not.

From these definitions, we will show how we can extend an algebraic signature with the monoid of paths to obtain addressed terms. The main idea is to represent pointers using paths.

3 Addressed Term Rewriting

In this section, we show how the notion of paths can be used to extend an algebraic signature in order to represent addresses. In this way, we define valid addressed terms (*i.e.* terms with valid pointers) and addressed term rewriting such that the validity is preserved by rewriting.

3.1 Addressed Term

An addressed term is a term whose nodes may be a path, which denotes a reference to another subterm.

Definition 3.1 (Addressed terms). *Given a set of symbols $\mathcal{F}$, a set of variables $\mathcal{X}$, and $\mathcal{P}$ the set of paths, we denote by $\mathcal{T}_a(\mathcal{F}, \mathcal{X})$ the set of* addressed terms *$\mathcal{T}(\mathcal{F} \cup \mathcal{P}, \mathcal{X})$, where the elements of $\mathcal{P}$ are considered as symbols of arity 0 and the sets $\mathcal{F}$,$\mathcal{X}$ and $\mathcal{P}$ are disjoint.*

Example 3.1. For $\mathcal{F} = \{f, g, a\}$, a, g(-1), and $g(f(a,$-2$\,.\,1))$ are *addressed terms*, elements of $\mathcal{T}_a(\mathcal{F})$. Given $\mathcal{F}$ and $\mathcal{X}$, we have $\mathcal{T}(\mathcal{F}, \mathcal{X}) \subset \mathcal{T}_a(\mathcal{F}, \mathcal{X})$. Note that $\bot \notin \mathcal{T}_a(\mathcal{F}, \mathcal{X})$, because $\bot \notin \mathcal{P}$, $\bot \notin \mathcal{F}$, and $\bot \notin \mathcal{X}$.

Definition 3.2 (Address dereferencing). *Given $t \in \mathcal{T}_a(\mathcal{F}, \mathcal{X})$ and $\omega \in \mathcal{P}os(t)$:*

$$\texttt{deref}(t, \omega) = \begin{cases} (\!|\omega \,.\, t_{|\omega}|\!) & \text{if } t_{|\omega} \in \mathcal{P} \\ \omega & \text{otherwise.} \end{cases}$$

The operation $\texttt{deref}(t, \omega)$ returns the position pointed by $t_{|\omega}$ when $t_{|\omega} \in \mathcal{P}$, and ω otherwise. For example, $\texttt{deref}(g(\text{-}1), 1) = \epsilon$, but $\texttt{deref}(g(a), 1) = 1$.

Problems will inevitably occur when $\texttt{deref}(t, \omega)$ returns a position which is not in t ($\texttt{deref}(g(-2), 1)$ for instance). The notion of *valid addressed terms* defines terms which do not have such problems.

Definition 3.3 (Valid addressed terms). *A term $t \in \mathcal{T}_a(\mathcal{F}, \mathcal{X})$ is a* valid addressed term *if $\forall \omega \in \mathcal{P}os(t)$ such that $t_{|\omega} \in \mathcal{P}$ we have:*

- *$t_{|\omega}$ is in canonical form,*
- *$\mathtt{deref}(t, \omega) \in \mathcal{P}os(t)$,*
- *$t_{|\mathtt{deref}(t,\omega)} \notin \mathcal{P} \setminus \{\epsilon\}$.*

We denote by $\mathcal{VT}_a(\mathcal{F}, \mathcal{X})$ the set of valid addressed terms.

We authorize empty paths in valid addressed terms in order to deal with degenerated cycles that can appear when applying collapsing rules, *e.g.* rewrite rules of the form $I(x) \to x$. In the term-rewriting formalism, a fresh constant called black hole and denoted by $\bullet$ is generally introduced [1]. In our context, it is not necessary to introduce a new symbol as the empty path (denoted ϵ) corresponds intuitively to this constant.

Example 3.2. The terms g(-1) and f(-1.2, a) are valid, but $g(3)$ and f(-1.2, -2) are not. Terms reduced to a non-empty address (1, -1.2, *etc.*) are elements of $\mathcal{T}_a(\mathcal{F}, \mathcal{X})$ but are not valid (*i.e.* $\notin \mathcal{VT}_a(\mathcal{F}, \mathcal{X})$). The term $t = f$(-1.1.-1, -2.3) is not valid for two reasons: -1.1.-1 is not in canonical form and $\mathtt{deref}(t, 2) = (\!|2.\text{-}2.3|\!) = 3$ which is not in $\mathcal{P}os(t) = \{\epsilon, 1, 2\}$.

3.2 Addressed term rewriting

Given the notion of addressed terms, we introduce the notion of *addressed term rewriting* which is an extension of term rewriting.

Definition 3.4 (Addressed term rewriting). *Given a rewrite rule $l \to r$ where $l, r \in \mathcal{VT}_a(\mathcal{F}, \mathcal{X})$ and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$, a term $t \in \mathcal{VT}_a(\mathcal{F})$ is rewritten into t' if there exists a position ω and a substitution σ such that $\sigma(l) = t_{|\omega}$ and $\mathcal{R}an(\sigma) \subseteq \mathcal{VT}_a(\mathcal{F}, \mathcal{X})$. t' is defined by $t[\sigma(r)]_\omega$.*

The main difference with term rewriting is that $\mathcal{R}an(\sigma) \subseteq \mathcal{VT}_a(\mathcal{F}, \mathcal{X})$ instead of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Therefore, classical term rewriting is included in addressed term rewriting.

In general, the notion of addressed term rewriting does not imply any condition of validity on the term to be reduced. However, when starting from a valid term and performing only rewrite steps at top position, the notion of validity is preserved.

Theorem 3.1. *The set of valid addressed terms $\mathcal{VT}_a(\mathcal{F}, \mathcal{X})$ is closed under rewriting at top-position.*

Proof. The proof is essentially based on the matching condition. As $\mathcal{R}an(\sigma) \subseteq \mathcal{VT}_a(\mathcal{F}, \mathcal{X})$, it ensures that a variable can only be instantiated by a valid addressed term (*i.e.* closed under its pointers). In particular, a variable can be instantiated neither by a path nor by a term with pointers to the external. As the right hand side r of the rule is valid, so is $\sigma(r)$. □

Since a subterm of a valid term is not necessary valid, Theorem 3.1 cannot be generalized to rewriting at any position. Consider the rule $f(x, y) \to f(y, x)$. When applied to $g(f(a(), \text{-}2\bullet\text{-}1))$ at the position 1, the redex is $f(a(), \text{-}2\bullet\text{-}1)$ and the resulting term is $g(f(\text{-}2\bullet\text{-}1, a()))$, which is not valid.

4 Term-Graph Rewriting

There exist several definitions of term-graph rewriting, category-theory oriented [13, 15], equationally oriented [1, 17] or implementation-oriented [5]. The difference between terms and term-graphs is the notion of horizontal and vertical sharing. In this section, we base our work on the equational framework introduced in [1]. This largely used framework allows the definition of possibly cyclic term-graphs, using flattened forms of systems of recursion equations.

Definition 4.1 (System of recursion equations from [1]). *Given $\mathcal{F}$ and $\mathcal{X}$, a system of recursion equations is a list $\{\alpha_1 = t_1, \ldots, \alpha_n = t_n\}$, $\alpha_i \in \mathcal{X}$, $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, where we assume that the first recursion variable α_1 represents the root of the graph. $t_1, \ldots, t_n$ are of the form $f(\beta_1, \ldots, \beta_m)$, $f \in \mathcal{F}$, $\beta_j \in \mathcal{X}$. Multiple definitions of a variable are not allowed and every α_i must be reachable from the root α_1.*

Given a system of recursion equations L, the root is denoted $\mathcal{R}oot(L)$. A variable α is said bound *when it appears in the left-hand side of an equation. Otherwise, α is* free.

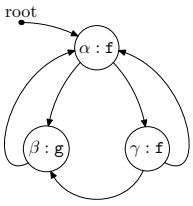
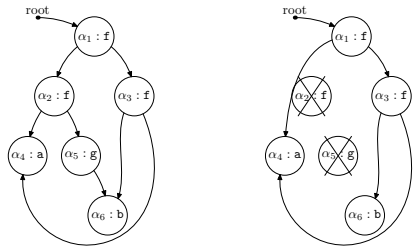


Fig. 1. This cyclic term-graph corresponds to the system of recursion equations $\{\alpha = f(\beta, \gamma), \beta = g(\alpha), \gamma = f(\beta, \alpha)\}$. It contains horizontal sharing: α and γ share the same subterm β; as well as vertical sharing: α is a subterm of both β and γ.

Definition 4.1 corresponds to systems of recursion equations in *flattened form* and ensures the unicity of the representation of a term-graph (modulo renaming of recursion variables). An example of term-graph is given in Figure 1.

Definition 4.2 (Equational term-graph rewriting [1]). *Given a rewrite rule composed of two systems of recursion equations L_1 and L_2 with the same root, a list of recursion equations L is rewritten into L' by the rule $L_1 \to L_2$ if there exists a substitution σ and a bound variable α such that $\sigma(L_1) \subseteq L$ and $\alpha = \mathcal{R}oot(\sigma(L_1))$. $L' = L \setminus \{\alpha = t\} \cup \sigma'(L_2)$ where $\sigma'(L_2)$ denotes $\sigma(L_2)$ in which every bound variable (except the root) has been renamed using a fresh name and after flattening equations and removing unreachable ones. Degenerated cycles, i.e. equations of the form $x = x$ are replaced by $x = \bullet$.*

Example 4.1. Suppose we want to apply the rule $\{\beta_1 = f(\beta_2, \beta_3)\} \to \{\beta_1 = \beta_2\}$, which corresponds to $f(x, y) \to x$, on the following term-graph:



initial subject s final result s'

The initial term-graph is $L = \{\alpha_1 = f(\alpha_2, \alpha_3), \alpha_2 = f(\alpha_4, \alpha_5), \alpha_3 = f(\alpha_6, \alpha_4), \alpha_4 = a,\ \alpha_5 = g(\alpha_6), \alpha_6 = b\}$. When applying the rule at position 1 (*i.e.* on α_2), we have $\sigma = \{\beta_1 \mapsto \alpha_2, \beta_2 \mapsto \alpha_4, \beta_3 \mapsto \alpha_5\}$ and α_2 is the selected bound variable. $\sigma(L_2) = \{\alpha_2 = \alpha_4\}$ (note that renaming with fresh variables is not necessary in this case) and we get $L' = L \setminus \{\alpha_2 = f(\alpha_4, \alpha_5)\} \cup \sigma(L_2)$ as final result. This corresponds to $\{\alpha_1 = f(\alpha_4, \alpha_3), \alpha_3 = f(\alpha_6, \alpha_4), \alpha_4 = a, \alpha_6 = b\}$ after flattening equations and removing unreachable ones.

Let us see how a representation of term-graphs can be obtained from the addressed terms introduced above.

4.1 Addressed term equivalence

In order to simulate term-graphs with addressed terms, we need to establish equivalence classes between valid addressed terms. For example, $f(\text{-}1\,.\,2, a)$ and $f(a, \text{-}2\,.\,1)$ should be equivalent. They both correspond to the term-graph rooted by f whose two children correspond to the shared subterm a. To define the equivalence, we need to introduce three intermediate functions that characterize translation, expansion and sharing.

The first one, called *subterm translation*, is essential in the following. Given a term t and two positions ω_1, ω_2, as illustrated in Figure 2, the translation in t from ω_1 to ω_2, denoted $t_{[\omega_1 \twoheadrightarrow \omega_2]}$, returns the subterm $t_{|\omega_1}$ where the addresses contained in $t_{|\omega_1}$ have been updated as if $t_{|\omega_1}$ was moved to the position ω_2. This first function is only introduced to express the expansion in a concise way.

The operation called *expansion* consists in replacing all the sharing by duplication. For example, the expansion of both $f(\text{-}1\,.\,2, a)$ and $f(a, \text{-}2\,.\,1)$ is $f(a, a)$. Thus two addressed terms are equivalent if their expanded terms are equal. However, this condition alone is not sufficient because the two terms to compare must also have a similar sharing. For example, $f(a, a)$ is not equivalent to $f(a, \text{-}2\,.\,1)$ because a is not explicitly shared in the first one. For this, we introduce a third relation called *sharing* which computes the set of shared positions.

Definition 4.3 (Subterm translation). *Given $t \in \mathcal{VT}_a(\mathcal{F}, \mathcal{X})$ and two positions ω_1, ω_2, we consider the subterm translation $t_{[\omega_1 \twoheadrightarrow \omega_2]}$ defined such that $\forall \delta \in \mathcal{P}os(t_{|\omega_1})$:*

$$t_{[\omega_1 \twoheadrightarrow \omega_2]|\delta} = \begin{cases} (\!|\, \mathtt{deref}(t, \omega_1 \,.\, \delta) - \omega_2 \,.\, \delta \,|\!) & \text{if } t_{|\omega_1 . \delta} \in \mathcal{P} \setminus \{\epsilon\} \\ t_{|\omega_1 . \delta} & \text{otherwise.} \end{cases}$$

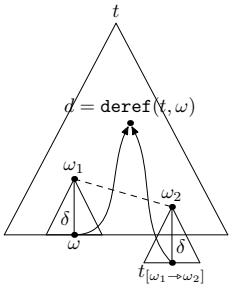


Fig. 2. Given t and ω_1, let us suppose that at position ω the subterm $t_{|\omega_1}$ contains a reference to the subterm d (*i.e.* $d = \texttt{deref}(t, \omega)$). The translation from ω_1 to ω_2 corresponds to an update of $t_{|\omega_1}$ as if it was moved to ω_2. By maintaining the pointers to the referenced terms, the paths stored in $t_{|\omega_1}$ are updated. The result of this operation is the updated subterm $t_{|\omega_1}$. For example, $f(g(\text{-}1\,.\,\text{-}1), a)_{[1 \twoheadrightarrow 2]} = g(\text{-}1\,.\,\text{-}2)$.

Definition 4.4 (Expansion). *Given $t \in \mathcal{VT}_a(\mathcal{F})$, we consider:*

$$\begin{cases} t_0 & = t \\ t_{n+1} & = t_{n[\texttt{deref}(t_n,\omega) \twoheadrightarrow \omega]} \text{ if } \exists \omega \text{ such that } t_{n|\omega} \in \mathcal{P} \setminus \{\epsilon\} \end{cases}$$

The function $\exp(t)$ *is defined as* $\lim_{n\to\infty} t_n$.

Example 4.2. The function `exp` replaces in a valid addressed term every address by the corresponding expanded subterm. $\texttt{exp}(f(\text{-}1\,.\,2, a)) = f(a, a)$ and $\texttt{exp}(g(\text{-}1)) = g(g(g(\ldots)))$. Note that in case of a cycle, the expanded term corresponds to an infinite term. A non-trivial example is $f(g(\text{-}1\,.\,\text{-}1\,.\,2), h(\text{-}1\,.\,\text{-}2\,.\,1))$. In this case, we need to update paths at every application of the rule. As the shared subterms are dependent, the result is infinite. We finally obtain $f(g(h(g(h(\ldots)))), h(g(h(g(\ldots)))))$.

Definition 4.5 (Sharing). *Given $t \in \mathcal{VT}_a(\mathcal{F})$, we consider:*

$$\begin{cases} \textit{share}_0(t) & = \{\{\omega, \textit{deref}(t, \omega)\} \mid \omega \in \mathcal{P}os(t) \text{ and } t_{|\omega} \in \mathcal{P}\} \\ \textit{share}_{n+1}(t) & = \{\{\omega' \,.\, q, q'\} \mid \{\omega, \omega'\}, \{\omega \,.\, q, q'\} \in \textit{share}_n(t)\} \end{cases}$$

The function $\textit{share}(t)$ *is defined as* $\lim_{n\to\infty} \textit{share}_n(t)$.

Example 4.3. The function `share` computes the set of shared position pairs. $\texttt{share}(f(\text{-}1.2, a)) = \{\{1, 2\}\}$ and $\texttt{share}(g(\text{-}1)) = \{\{1, \epsilon\}\}$. A non-trivial example is $f(g(\text{-}1.\text{-}1.2), h(\text{-}1.\text{-}2.1))$. At the first step, $\texttt{share}_0(t) = \{\{1, 2.1\}, \{2, 1.1\}\}$. In this case, it is necessary to close the relation of sharing with prefixes. We finally obtain the infinite set of related positions $\texttt{share}(t) = \{\{1, 2.1\}, \{2, 1.1\}, \ldots, \{1.(1.1)^*, 2.1.(1.1)^*\}, \{2.(1.1)^*, 1.1.(1.1)^*\}\}$ (* denotes the sublist repetition). This infinite result is due to the inter-dependency of the two addresses.

Definition 4.6 (Equivalence). *Two valid addressed terms t_1, t_2 are equivalent (denoted by $t_1 \backsim t_2$) if $\textit{share}(t_1) = \textit{share}(t_2)$ and $\textit{exp}(t_1) = \textit{exp}(t_2)$.*

Note that it is easy to show that $\backsim$ is an equivalence relation (reflexive, symmetric, and transitive).

4.2 Canonical form of addressed terms

For every equivalence class, we define a canonical form using a lexicographic order on positions, denoted $<_\omega$. For example $\epsilon <_\omega 1$ and $1\,.\,2 <_\omega 1\,.\,3$.

Definition 4.7 (Canonical form). *A valid addressed term $t \in \mathcal{VT}_a(\mathcal{F},\mathcal{X})$ is canonical if for every subterm $t_{|\omega} \in \mathcal{P} \setminus \{\epsilon\}$, $\mathtt{deref}(t,\omega) <_\omega \omega$.*
We denote by $\mathcal{T}_g(\mathcal{F},\mathcal{X})$ the set of canonical addressed terms.

Example 4.4. The term $f(a,\text{-}2\,.\,1)$ is canonical but $f(\text{-}1\,.\,2,a)$ is not because $\mathtt{deref}(f(\text{-}1\,.\,2,a),1) = 2$ (the position of the pointed subterm a) is not smaller than 1.

To define the normalization function that returns the canonical form of any valid addressed term, we introduce a swapping function that permutes two subterms and updates all the paths contained in the global term in order to preserve the sharing. First, we translate the two subterms. This translation updates the pointers from the subterms to the external context. To obtain a valid addressed term, we still have to update every pointer from the outside to the subterms.

Definition 4.8 (Swapping). *Given $t \in \mathcal{VT}_a(\mathcal{F},\mathcal{X})$ and two disjoint positions ω_1,ω_2 ($\omega_1 \not\sqsubseteq \omega_2$ and $\omega_2 \not\sqsubseteq \omega_1$), we consider $u = t_{[\omega_1 \to \omega_2]}$, $v = t_{[\omega_2 \to \omega_1]}$, $t' = t[v]_{\omega_1}[u]_{\omega_2}$. The swapping in t of the subterms at position ω_1 and ω_2 is denoted by $t_{[\omega_1 \leftrightarrow \omega_2]}$, and is defined such that $\forall \omega \in \mathcal{P}os(t')$, we have:*

$$t_{[\omega_1 \leftrightarrow \omega_2]|\omega} = \begin{cases} (\!|\omega_2\,.\,\delta - \omega|\!) & \text{if } t'_{|\omega} \in \mathcal{P} \setminus \{\epsilon\} \text{ and } \exists \delta \text{ s.t. } \mathtt{deref}(t',\omega) = \omega_1\,.\,\delta \\ (\!|\omega_1\,.\,\delta - \omega|\!) & \text{if } t'_{|\omega} \in \mathcal{P} \setminus \{\epsilon\} \text{ and } \exists \delta \text{ s.t. } \mathtt{deref}(t',\omega) = \omega_2\,.\,\delta \\ t'_{|\omega} & \text{otherwise.} \end{cases}$$

Example 4.5. $f(a,b)_{[1 \leftrightarrow 2]} = f(b,a)$, $f(g(\text{-}1\,.\,\text{-}1\,.\,2),h(\text{-}1\,.\,\text{-}2\,.\,1))_{[1 \leftrightarrow 2]} = f(h(\text{-}1\,.\,\text{-}1\,.\,2),g(\text{-}1\,.\,\text{-}2\,.\,1))$. A more complex example is the swap of a and b in $t = f(f(a,b),\text{-}2\,.\,1\,.\,2)$. In this case, the address $\text{-}2\,.\,1\,.\,2$ has to be updated because it has to reference b. Thus, the result is $f(f(b,a),\text{-}2\,.\,1\,.\,1)$.

Contrary to the translation, the swapping preserves the notion of validity (Definition 3.3). Note that the complexity of swapping is linear on the size of the term since in the worst case, all the addresses in the term must be updated. We will now introduce a *normalization function* that associates to every valid addressed term its canonical form.

Definition 4.9 (Normalization). *We define $[\![\ ]\!] : \mathcal{VT}_a(\mathcal{F},\mathcal{X}) \to \mathcal{T}_g(\mathcal{F},\mathcal{X})$ such that given $t \in \mathcal{VT}_a(\mathcal{F},\mathcal{X})$, $[\![t]\!]$ is the normal form by the relation $t \to t_{[\omega \leftrightarrow \mathtt{deref}(t,\omega)]}$ if $\omega <_\omega \mathtt{deref}(t,\omega)$.*

Example 4.6. $[\![a]\!] = a$, $[\![f(\text{-}1\,.\,2,a)]\!] = f(a,\text{-}2\,.\,1)$, and $[\![f(g(\text{-}1\,.\,\text{-}1\,.\,2),h(\text{-}1\,.\,\text{-}2\,.\,1))]\!] = f(g(h(\text{-}1\,.\,\text{-}1)),\text{-}2\,.\,1\,.\,1)$

Note that the normalization is linear in the size of the term when the swapping is applied in a leftmost-innermost way.

Proposition 4.1. *The normalization function is convergent.*

Proof. The proof is in the Appendix A. □

Proposition 4.2. *$\forall t \in \mathcal{VT}_a(\mathcal{F},\mathcal{X})$, t and $[\![t]\!]$ are equivalent.*

Proof. Given $t \in \mathcal{VT}_a(\mathcal{F},\mathcal{X})$, every rewriting step of normalization preserves the two functions $\texttt{share}(t)$ and $\texttt{exp}(t)$. In fact, the swapping between a pointer and its corresponding pointed subterm preserves the sharing and as only addresses are updated the expansion is the same. □

Proposition 4.3. *$\forall t_1, t_2 \in \mathcal{VT}_a(\mathcal{F},\mathcal{X})$, we have: $[\![t_1]\!] \backsim [\![t_2]\!] \Leftrightarrow [\![t_1]\!] = [\![t_2]\!]$.*

Proof. First, the proof that $[\![t_1]\!] = [\![t_2]\!] \Rightarrow [\![t_1]\!] \backsim [\![t_2]\!]$ is trivial because as $\backsim$ is an equivalence relation and thus is reflexive. Secondly, we prove that $[\![t_1]\!] \backsim [\![t_2]\!] \Rightarrow [\![t_1]\!] = [\![t_2]\!]$. Suppose they are not equal, it means that there exists a shared subterm at a position ω addressed at a position ω' in $[\![t_1]\!]$ and the contrary in $[\![t_2]\!]$. As $[\![t_1]\!], [\![t_2]\!]$ are canonical, it implies that $\omega <_\omega \omega'$ and $\omega' <_\omega \omega$ which is impossible due to the total order on positions. So $[\![t_1]\!] = [\![t_2]\!]$. □

Theorem 4.1. *$\forall t_1, t_2 \in \mathcal{VT}_a(\mathcal{F},\mathcal{X})$, we have: $t_1 \backsim t_2 \Leftrightarrow [\![t_1]\!] = [\![t_2]\!]$*

Proof. First, we will prove that given $t_1, t_2 \in \mathcal{VT}_a(\mathcal{F},\mathcal{X})$, the equivalence of t_1 and t_2 implies that $[\![t_1]\!] = [\![t_2]\!]$. Due to the proposition 4.2 and the transitivity of the equivalence, $[\![t_1]\!] \backsim [\![t_2]\!]$. Due to the proposition 4.3, $[\![t_1]\!] = [\![t_2]\!]$. Now, we will prove that $[\![t_1]\!] = [\![t_2]\!]$ implies that $t_1 \backsim t_2$. Due to the proposition 4.2, $t_1 \backsim [\![t_1]\!]$ and $t_2 \backsim [\![t_2]\!]$. As $[\![t_1]\!] = [\![t_2]\!]$ and because of the equivalence transitivity, $t_1 \backsim t_2$. □

In practice, to verify that two terms are equivalent we will compare their canonical forms. It is simpler and more realistic than computing $\texttt{exp}(t)$ and $\texttt{share}(t)$, which moreover can be infinite.

4.3 Term-graph rewriting using canonical addressed terms

We introduce an efficient algorithm for implementing term-graph rewriting using canonical addressed terms. The idea is to use syntactic pattern matching and a newly defined rewrite step that preserves shared parts of the redex as in Definition 4.2, page 6.

Definition 4.10 (Rewriting algorithm). *Given $t \in \mathcal{T}_g(\mathcal{F})$ and $l, r \in \mathcal{T}_g(\mathcal{F},\mathcal{X})$:*

- *we denote by ω_{xl} the position of the variable x in l. In case of non-linearity, we choose the smallest position where x appears in l.*
- *t is rewritten into t' by the rule $l \to r$ if:*
  1. *there exists a position ω and a substitution σ such that $\sigma(l) = t_{|\omega}$,*
  2. *$t' = [\![\langle \dot{t}, \dot{r} \rangle_{[1.\omega \leftrightarrow 2]}]\!]_{|1}$, where*
     $\langle \_, \_ \rangle$ is a fresh binary symbol,
     $\dot{t}$ correponds to t where every path towards the position $1.\omega$ has been replaced by a path towards the position 2,
     $\dot{r}$ is the ground term corresponding to r in which any occurrence of a variable x (whose position is denoted by ω_{xr}) is replaced by the path $(\!|\texttt{deref}((1.\omega.\omega_{xl}), \langle \dot{t}, r \rangle) - (2.\omega_{xr})|\!)$.

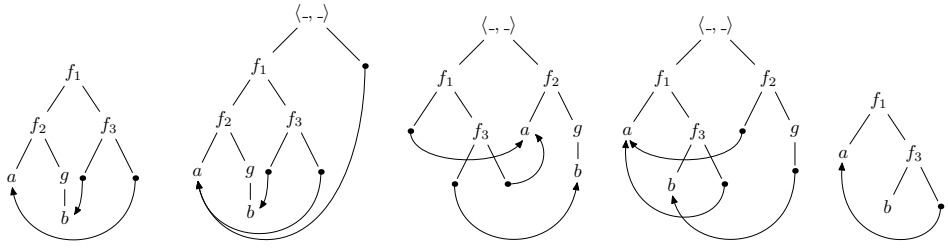
In this algorithm, no substitution is explicitly computed. Instead, the right hand side of the rule is instantiated by replacing every variable by paths to their corresponding subterm in $t_{|\omega}$. By swapping the redex and the right hand side in $\langle \dot{t}, \dot{r} \rangle_{[1.\omega \leftrightarrow 2]}$, the substitution is automatically applied. The main subtlety of the algorithm is to rebalance the whole term using normalization. All the shared subterms in $t_{|\omega}$ that must be conserved are moved to the term and the unused part is left in the right part of $\langle \dot{t}, \dot{r} \rangle_{[1.\omega \leftrightarrow 2]}$. At the end, the result of the rewrite step corresponds exactly to the left child of $[\![\langle \dot{t}, \dot{r} \rangle_{[1.\omega \leftrightarrow 2]}]\!]$.

The complexity of this algorithm is linear in the size of the subject because the complexity of the swapping and the normalization are linear.

Example 4.7. Suppose we want to apply the rule $f(x, y) \to f(y, x)$ on the subject $t = f(a, b)$. The rule is applied at top-position, therefore we have $\omega_{xl} = 1$, $\omega_{yl} = 2$, $\omega_{xr} = 2$, $\omega_{yr} = 1$, $\dot{t} = f(a, b)$ and $\dot{r} = f(\langle\!|(1 \,.\, \epsilon \,.\, \omega_{yl}) - (2 \,.\, \omega_{yr})|\!\rangle, \langle\!|(1 \,.\, \epsilon \,.\, \omega_{xl}) - (2 \,.\, \omega_{xr})|\!\rangle) = f(\text{-}1 \,.\, \text{-}2 \,.\, 1 \,.\, 2, \text{-}2 \,.\, \text{-}2 \,.\, 1 \,.\, 1)$. Starting from $\langle f(a, b), \dot{r} \rangle$, we get $\langle f(a, b), \dot{r} \rangle_{[1 \leftrightarrow 2]} = \langle f(\text{-}1 \,.\, \text{-}1 \,.\, 2 \,.\, 2, \text{-}2 \,.\, \text{-}1 \,.\, 2 \,.\, 1), f(a, b) \rangle$. When computing the canonical form, we obtain $[\![\langle f(a, b), \dot{r} \rangle_{[1 \leftrightarrow 2]}]\!] = \langle f(b, a), f(\text{-}1 \,.\, \text{-}2 \,.\, 1 \,.\, 1, \text{-}2 \,.\, \text{-}2 \,.\, 1 \,.\, 2) \rangle$. Finally, the result is $[\![\langle f(a, b), \dot{r} \rangle_{[1 \leftrightarrow 2]}]\!]_{|1} = f(b, a)$ as expected.

The following example illustrates how collapsing rules are handled. Suppose we want to apply the rule $f(x) \to x$ to the subject $t = f(\text{-}1)$. Since -1 is a path to the top of the redex, we have $\dot{t} = f(\text{-}1 \,.\, \text{-}1 \,.\, 2)$. In a second step $\dot{r}$ is evaluated to $\dot{r} = \langle\!|\texttt{deref}((1 \,.\, 1), \langle f(\text{-}1 \,.\, \text{-}1 \,.\, 2), x \rangle) - 2|\!\rangle = \langle\!|2 - 2|\!\rangle = \epsilon$ and we get $\langle f(\text{-}1 \,.\, \text{-}1 \,.\, 2), \epsilon \rangle_{[1 \leftrightarrow 2]} = \langle \epsilon, f(\text{-}1 \,.\, \text{-}2 \,.\, 1) \rangle$ which is already normalized. The result of the rewrite step is $\langle \epsilon, f(\text{-}1 \,.\, \text{-}2 \,.\, 1) \rangle_{|1} = \epsilon$ in accordance with [1].

By applying the rule $f(x, y) \to x$ to the first subterm, this last example shows what happens to pointers to subterms that disappear:



In this example, the subterm $g(b)$ is not preserved by the rule, therefore the reference to b is replaced by a copy of the subterm.

Theorem 4.2. *The set of canonical terms $\mathcal{T}_g(\mathcal{F}, \mathcal{X})$ is closed under rewriting.*

Proof. Given a term $t \in \mathcal{T}_g(\mathcal{F}, \mathcal{X})$, we have to prove that $[\![\langle \dot{t}, \dot{r} \rangle_{[1.\omega \leftrightarrow 2]}]\!]_{|1} \in \mathcal{T}_g(\mathcal{F}, \mathcal{X})$. The ground term $\dot{r}$ is not a valid addressed term in itself because of the addresses that replace variables. The term $\dot{t}$ is neither valid because some paths can have been replaced by paths to the position 2. But, $\langle \dot{t}, \dot{r} \rangle$ is valid because unvalid paths in $\dot{t}$ are now referencing the top position of $\dot{r}$ and invalid paths

in $\dot{r}$ are referencing subterms of t. Since the swapping function preserves the notion of validity, the normal form $[\![\langle \dot{t}, \dot{r} \rangle_{[1.\omega \leftrightarrow 2]}]\!]$ exists. Thanks to the property of canonical form (pointers only from the right to the left), we can conclude that $[\![\langle \dot{t}, \dot{r} \rangle_{[1.\omega \leftrightarrow 2]}]\!]_{|1} \in \mathcal{T}_g(\mathcal{F}, \mathcal{X})$. □

As the main result of the paper we show in the next section that every term-graph can be represented by a canonical addressed term and that term-graph rewriting (Definition 4.2) can be simulated by the algorithm introduced in Definition 4.10.

4.4 Simulation of term-graph rewriting

We introduce a function ϕ that associates to every system of recursion equations its canonical representation in $\mathcal{T}_g(\mathcal{F}, \mathcal{X})$. For this purpose, we need an intermediate representation between systems of equations and addressed terms. The representation is called *labelled terms* and is defined in [16]. In this representation, $l : t$ means that the term t is labelled by l. Given a list of recursion equations L, a recursion variable α bound in L, and a set of recursion variables Λ, we consider the function $\psi(L, \alpha, \Lambda)$, which returns a pair, and is defined as follows:

$$\psi(L,\alpha,\Lambda) = \begin{cases} (\alpha : f(\beta'_1,\ldots,\beta'_m), \Lambda'_m) \text{ if } & \alpha \notin \Lambda \\ \text{where } & \alpha = f(\beta_1,\ldots,\beta_m) \in L \\ & (\beta'_1, \Lambda'_1) = \psi(L, \beta_1, \Lambda \cup \{\alpha\}) \\ & \vdots \\ & (\beta'_m, \Lambda'_m) = \psi(L, \beta_m, \Lambda'_{m-1}) \\ (\alpha, \Lambda) \text{ otherwise.} & \end{cases}$$

Example 4.8. Considering the example given Figure 1, we have $L = \{\alpha = f(\beta,\gamma), \beta = g(\alpha), \gamma = f(\beta,\alpha)\}$. $\psi(L,\alpha,\emptyset)$ is equal to the pair $(\alpha : f(\beta'_1,\beta'_2), \Lambda'_2)$ where $\beta'_1 = \beta : g(\alpha)$, $\Lambda'_1 = \{\alpha,\beta\}$, $\beta'_2 = \gamma : f(\beta,\alpha)$, $\Lambda'_2 = \{\alpha,\beta,\gamma\}$. In other words, the intermediate *labelled term* representation of L is $\alpha : f(\beta : g(\alpha), \gamma : f(\beta,\alpha))$.

By abuse of notation, we denote by $\psi(L)$ the projection on the first field $\pi_1(\psi(L,\ \mathcal{R}oot(L),\ \emptyset))$. With this representation of a system of equations, we identify where the subterm must be attached (labelled subterm) and where we must put addresses. Given a labelled term t, we associate to each label α_i in t a position ω_{α_i} corresponding to the subterm labelled by this variable. For example, in $\alpha : f(\beta : g(\alpha), \gamma : f(\beta,\alpha))$, $\omega_\alpha = \epsilon$, $\omega_\beta = 1$ and $\omega_\gamma = 2$.

Definition 4.11. *Given a list of recursion equations L, we define its representation $\phi(L) \in \mathcal{T}_g(\mathcal{F}, \mathcal{X})$ such that $\forall \omega \in \mathcal{P}os(\psi(L))$:*

$$\mathcal{S}ymb(\phi(L)_{|\omega}) = \begin{cases} f & \text{if } \psi(L)_{|\omega} = \alpha : f(\ldots) \\ (\!|\omega_\alpha - \omega|\!) & \text{if } \psi(L)_{|\omega} = \alpha \text{ and } \alpha \text{ is bound in } L \\ \alpha & \text{otherwise} \end{cases}$$

ϕ can be naturally extended to rules: $\phi(L_1 \to L_2) = \phi(L_1) \to \phi(L_2)$.

Example 4.9. Continuing the previous example, we have $\psi(L) = \alpha : f(\beta : g(\alpha), \gamma : f(\beta, \alpha))$, $\omega_\alpha = \epsilon$, $\omega_\beta = 1$, and $\omega_\gamma = 2$. Therefore, the system of recursion equation is represented by the canonical addressed term $\phi(L) = f(g(\text{-}1 \centerdot \text{-}1), f(\text{-}1 \centerdot \text{-}2 \centerdot 1, \text{-}2 \centerdot \text{-}2))$.

Theorem 4.3 (Rewrite step simulation). *Given G a system of recursion equations and $L \to R$ a rewrite rule denoted by r, we have:*

$$G \to_r G' \Rightarrow \phi(G) \to_{\phi(r)} \phi(G')$$

Proof. The proof is in the Appendix B. □

Acyclic term graphs are widely used to obtain efficient term rewriting implementation [12]. In our context, Theorem 4.3 shows how simulating term-graph rewriting using term-rewriting and provides an efficient technique for easily extending rule-based languages with term-graphs and more generally with a notion of pointers.

5 Related work

The notion of term graph has been intensively studied in the literature, in particular in [5], where a restricted form of *acyclic* term-graph has been used to represent terms with sharing. This approach leads to efficient implementations of a term rewriting engine, as in Clean, Elan, or Maude.

We are in a dual situation: we already have a very efficient term rewriting engine. This implementation is based on the notion of *maximally shared terms*, internally represented by acyclic graphs. Such a representation is purely functional and does not allow any side effect. This constraint makes the implementation of graphs and term-graphs difficult. The contribution of the paper is to provide a solution to represent and transform graphs in a functional environment. In addition to be efficient (linear complexity), a main advantage of using a classical underlying term representation is to make possible the reuse of the notion of term rewriting strategy [14, 18] which allows to control how rules are applied.

The extension to cyclic term-graph rewriting has been studied and in particular linked up with rational term rewriting [7, 8]. Especially, a mapping from cyclic term-graph rewriting to rational parallel term rewriting can be defined. In this context, it is often difficult to deal with graph homomorphism. In this work, we choose to simulate term-graph rewriting as defined in [1] and to favor practical aspects. As a consequence, the term-graphs $g(\text{-}1)$ and $g(g(\text{-}1 \centerdot \text{-}1))$ are not equivalent even if they both represent the same infinite term g^ω.

Moreover, the set of valid addressed terms where addresses are not interpreted as sharing but as oriented pointers (Section 3) is to our knowledge a new approach that can be interesting to study and simulate object-oriented languages [11]. For example, it could be used to model garbage collection algorithms.

In the context of term-graph rewriting, an original approach is the formalism presented in [11] where the right-hand side of the rules consists in a set of actions on the pointers. The work presented in this paper is a first step towards an implementation of such a formalism.

5.1 Integration in a rewrite environment: the Tom language

As canonical addressed terms are terms, it is possible to extend in a non-intrusive way any rule-based language in order to support efficient term-graph rewriting. Actually, the presented formalism is implemented in Tom and thus directly available at `http://tom.loria.fr`.

For now, it is possible to automatically generate from a signature the extended version for addressed terms where the normalization is integrated. As the Tom terms are implemented with maximal sharing, so are the term-graphs. This part of the implementation is presented in [3]. All the operations on paths have been also implemented. Thus users can define a system of term-graph rules and it is automatically compiled in a basic Tom strategy based on the rewriting algorithm (Definition 4.10). These term-graph rewriting rules can then be integrated in a more complex strategy using Tom strategy combinators. As a consequence, all Tom features are available for term-graphs.

6 Conclusion

We have generalized the notion of term positions with *term paths* that are closely related to the notion of path in graphs and to the concept of de Bruijn indices [10]. By extending a signature with paths we obtained a new kind of rewriting called *addressed term rewriting* where terms can contain pointers. Since this extension is close to term rewriting (the main difference is the condition on the substitution's codomain), classical known results about confluence and termination can be reused in this context. This will be useful to model the semantics of imperative programming languages for instance.

In the second part of the paper, we introduced *canonical addressed terms* to represent term-graphs. As in the case of de Bruijn indices, the interest of this representation is to avoid problems of alpha conversion, compared to representations with labels or variables. Another advantage is that contrary to a recursion equation, the hierarchical structure of the term-graph is explicit because of its term representation. The last contribution of the paper is an efficient algorithm that simulates cyclic term-graph rewriting using canonical addressed terms. Thanks to pointers, the substitution can be applied in an unusual way, using swapping between the redex and the right-hand side of the rule.

To conclude, this formalism opens promising perspectives in terms of program transformation and code analysis. To the best of our knowledge, the integration in the Tom language constitutes actually one of the most active and maintained implementation of term-graph rewriting and thus provides a solid platform to experiment graph transformations in a concise and expressive way.

Acknowledgments: We would like to thank Rachid Echahed for fruitful exchanges that greatly contributed to improve the paper. We also thank Joe Wells and the members of the Protheo team for advices and remarks about premilinary versions of this work.

References

1. Z. M. Ariola and J. W. Klop. Equational term graph rewriting. *Fundamenta Informaticae*, 26(3/4):207–240, 1996.
2. F. Baader and T. Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998.
3. E. Balland and P. Brauner. Term-graph rewriting in tom using relative positions. In *International Workshop on Computing with Terms and Graphs -TERMGRAPH*, 2007.
4. E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles. Tom: Piggybacking rewriting on java. In *RTA*, volume 4533 of *LNCS*. Springer-Verlag, 2007.
5. H. P. Barendregt, M. van Eekelen, J. Glauert, J. Kennaway, M. Plasmeijer, and M. Sleep. Term graph rewriting. In *PARLE Parallel Architectures and Languages Europe*, volume 259 of *LNCS*, pages 141–158, Eindhoven, 1987. Springer-Verlag.
6. B. Carré, editor. *Graphs and Networks*. Clarendon Press, Oxford, 1979.
7. A. Corradini and F. Drewes. (cyclic) term graph rewriting is adequate for rational parallel term rewriting. Technical Report TR-97-14, Dipartimento di Informatica, Pisa, Italy, 1997.
8. A. Corradini and F. Gadducci. Rational term rewriting. In *FoSSaCS*, volume 1378 of *LNCS*, pages 156–171, 1998.
9. B. Courcelle. Fundamental properties of infinite trees. *TCS*, 25:95–169, 1983.
10. N. G. de Bruijn. Lambda calculus notation with nameless dummies. a tool for automatic formula manipulation with application to the church-rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.
11. R. Echahed and N. Peltier. Narrowing data-structures with pointers. In *ICGT*, volume 4178 of *LNCS*, pages 92–106, 2006.
12. B. Hoffmann and D. Plump. Implementing term rewriting by jungle evaluation. *RAIRO: Theoretical Informatics and Applications*, 25, 1991.
13. R. Kennaway. On graph rewritings. *TCS*, 52(1-2):37–58, 1987.
14. C. Kirchner. Strategic rewriting. In *International Workshop on Reduction Strategies in Rewriting and Programming - WRS*, volume 124 of *ENTCS*, 2004.
15. M. Löwe. Algebraic approach to single-pushout graph transformation. *TCS*, 109(1–2):181–224, 1993.
16. E. Ohlebusch. Implementing conditional term rewriting by graph rewriting. *TCS*, 262(1):311–331, 2001.
17. D. Plump. *Handbook of Graph Grammars and Computing by Graph Transformation*, chapter Term graph rewriting, pages 3–61. World Scientific Publishing, 1999.
18. E. Visser. Stratego: A language for program transformation based on rewriting strategies. System description of Stratego 0.5. In *RTA*, volume 2051 of *LNCS*, pages 357–361, 2001.

A Proof of Proposition 4.1

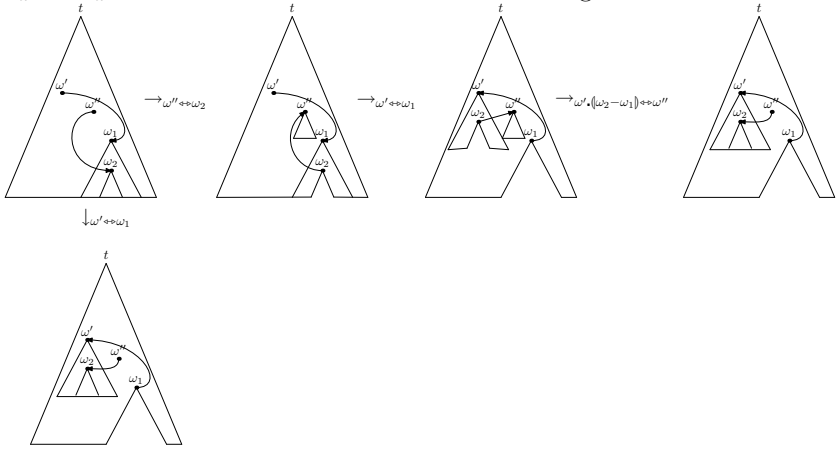
Theorem. *The normalization function is convergent.*

We have to prove termination and confluence. The termination property is trivial because of the total order on positions and of the fact that valid addressed terms do not contain pointers to pointers. Consider the multiset $\{\texttt{deref}(t,\omega) \mid t_{|\omega} \in \mathcal{P}\}$ composed of the pointed positions and the order on mutisets (a multiset N is smaller than a multiset M if it is obtained by replacing finitely many elements of M by smaller elements). This multiset decreases strictly at each rewriting step. Indeed, after applying the rule at a given position, $t_{|\omega} \notin \mathcal{P}$ and $t_{|\texttt{deref}(t,\omega)} \in \mathcal{P}$. Now $\texttt{deref}(t,\texttt{deref}(t,\omega)) = \omega$ and $\omega <_\omega \texttt{deref}(t,\omega)$. Concerning the other updated addresses, $\texttt{deref}(t,\omega)$ is either smaller (pointers to the swapped subterm or to subterms of it) or remains unchanging (pointers inside the subterm). Finally, the cardinality of the multiset has not evolved and each replaced element has been substituted by a smaller one so the termination is ensured.

Moreover, we will show that for the same reason, the normal form is unique. Suppose there are two distinct positions ω' and ω'' where the rule can be applied. We denote $\omega_1 = \texttt{deref}(t,\omega')$ and $\omega_2 = \texttt{deref}(t,\omega'')$. We distinguish several cases depending on the order between positions and the prefixation (noted $\sqsubseteq$). Some combinations of conditions are impossible because the rule can be applied only if $\omega' <_\omega \omega_1$ and $\omega'' <_\omega \omega_2$. Moreover, each prefixation $\omega \sqsubseteq \beta$ implies that $\omega <_\omega \beta$. If we consider that $\omega' <_\omega \omega''$, there are eight cases to consider:

$\omega'' < \omega_1 < \omega_2 \wedge \begin{cases} \omega_1 \sqsubseteq \omega_2 \\ \omega_1 \not\sqsubseteq \omega_2 \end{cases}$	$\omega'' < \omega_2 < \omega_1 \wedge \begin{cases} \omega_2 \sqsubseteq \omega_1 \\ \omega_2 \not\sqsubseteq \omega_1 \end{cases}$
$\omega' < \omega_1 < \omega'' < \omega_2 \wedge \begin{cases} \omega_1 \sqsubseteq \omega'' \wedge \omega_1 \sqsubseteq \omega_2 \\ \omega_1 \sqsubseteq \omega'' \wedge \omega_1 \not\sqsubseteq \omega_2 \\ \omega_1 \not\sqsubseteq \omega'' \wedge \omega_1 \not\sqsubseteq \omega_2 \end{cases}$	$\omega_1 = \omega_2$

For each case, we can prove local confluence. We will only detail the first case $\omega'' <_\omega \omega_1 <_\omega \omega_2 \wedge \omega_1 \sqsubseteq \omega_2$. The other cases are analogous.



If we start by swapping ω' and ω_1, as $\omega_1 \sqsubseteq \omega_2$, the subterm at position ω_2 is translated to the position $\omega' \centerdot (\!|\omega_2 - \omega_1|\!)$ and its pointer at position ω'' is updated. Now $\texttt{deref}(t, \omega'') = \omega' \centerdot (\!|\omega_2 - \omega_1|\!)$ and as $\omega' <_\omega \omega''$, $\texttt{deref}(t, \omega'') <_\omega \omega''$, we do not need a second swapping.

If we start by swapping ω'' and ω_2, ω' is not updated and $\omega' <_\omega \texttt{deref}(t, \omega') = \omega_1$. So we need to swap ω' and ω_1 and then translate the pointer at position ω_2 to the position $\omega' \centerdot (\!|\omega_2 - \omega_1|\!)$ which does not respect the order on positions $(\omega' \centerdot (\!|\omega_2 - \omega_1|\!) <_\omega \texttt{deref}(t, \omega' \centerdot (\!|\omega_2 - \omega_1|\!)) = \omega'')$. We need a third swapping between ω'' and $\omega' \centerdot (\!|\omega_2 - \omega_1|\!)$.

As the normalization terminates, local confluence implies convergence. □

B Proof of Theorem 4.3

Theorem. *Given G a system of recursion equations and $L \to R$ a rewrite rule denoted by r, we want to prove that: $G \to_r G' \Rightarrow \phi(G) \to_{\phi(r)} \phi(G')$.*

Applying $L \to R$ (where $\mathcal{R}oot(R) = \mathcal{R}oot(L)$) to G means there exists a substitution σ and an equation $\alpha = t_\alpha \in G$, such that $\sigma(L) \subseteq G$ and $\alpha = \mathcal{R}oot(\sigma(L))$. According to Definition 4.2, $G' = G \setminus \{\alpha = t_\alpha\} \cup \sigma'(R)$, where $\sigma'(R)$ denotes $\sigma(R)$ in which every bound variable (except the root α) has been renamed using a fresh name. We now consider the intermediate *labelled term* representation $\psi(G)_{|\omega_\alpha}$, where ω_α is the position of the subterm $\alpha : t_\alpha$ in $\psi(G)$.

We can show that $\exists \sigma' \mid \sigma'(\psi(L)) = \psi(G)_{|\omega_\alpha}$. Indeed, the substitution σ' can be expressed from σ. First, we restrict the domain of σ' to free recursion variables of L because only these variables correspond to variables in $\psi(L)$. Then, $\forall \beta \in \mathcal{D}om(\sigma')$, $\sigma'(\beta) = \sigma(\beta) : t_{\sigma(G)}$ if $\sigma(G)_{|\omega_\alpha \centerdot \omega'} = \sigma(\beta) : t_{\sigma(G)}$ where ω' is the position of β in L. $\sigma'(\beta) = \sigma(\beta)$ otherwise. Thus, from Definition 4.11 and as $\exists \sigma' \mid \sigma'(\psi(L)) = \psi(G)_{|\omega_\alpha}$, $\exists \theta \mid \theta(\phi(L)) = \phi(G)_{|\omega_\alpha}$. Actually, θ can be deduced from σ'. The idea is to replace recursion variables by pointers when necessary (*i.e.* for each β such that $\sigma'(\beta) = \alpha_i$).

Therefore, due to Definitions 4.2 and 4.10, if $L \to R$ can be applied to G at position ω_α, then $\phi(L \to R)$ can be applied to $\phi(G)$ at position ω_α. We denote by t' the result of such application, and we must show that $t' = \phi(G')$. First, we will express $\psi(G')$ as a function of $\psi(G)$. Recall that in a term-graph, the equations which are not reachable from the root are automatically removed. In our case, since the equation $\{\alpha = t_\alpha\}$ has been removed when building G', the equations corresponding to variables only reachable from α are no longer there in G'. These equations correspond to subterms of the term-graph $\sigma(L)$ no longer shared in G'.

We call u the labelled term $\psi(G)$ after applying the substitution (*i.e.* $u = \psi(G)[\sigma'(\psi(R))]_{\omega_\alpha}$). To construct $\psi(G')$ from u, subterms of $\psi(\sigma(L))$ still shared have to be copied. In fact, in u, $\psi(\sigma(G))$ has disappeared so contrary to G', we do not need to remove unreachable subterms but we have to copy subterms which are still shared. Formally, $\psi(L)$ corresponds to u where every recursion variable α bounded in $\sigma(L)$ (*i.e.* $\alpha = t_\alpha \in \sigma(L)$) and free in u have been replaced by t_α.

According to Definition 4.10, $t' = [\![\langle \dot{t}, \dot{v} \rangle_{[1.\omega \leftrightarrow 2]}]\!]_{|1}$, with $v = \phi(R)$. After swapping, $\langle \dot{t}, \dot{v} \rangle_{[1.\omega \leftrightarrow 2]_{|1}}$ corresponds to u. In fact, due to the order on the positions, every subterm shared between the redex and the rest of the term is copied in $\langle \dot{t}, \dot{v} \rangle_{[1.\omega \leftrightarrow 2]_{|1}}$ and thus we obtain the correspondence with $\psi(G')$. Therefore, thanks to Definition 4.11, we can show that $t' = \phi(G')$. □