Admissible Graph Rewriting and Narrowing

Rachid Echahed and **Jean-Christophe Janodet**
Laboratoire LEIBNIZ, IMAG, CNRS
46, avenue Felix Viallet F-38031 Grenoble - France
Rachid.Echahed@imag.fr ; Jean-Christophe.Janodet@imag.fr

Abstract

We address the problem of graph rewriting and narrowing as the underlying operational semantics of rule-based programming languages. We propose new optimal graph rewriting and narrowing strategies in the setting of orthogonal constructor-based graph rewriting systems. For this purpose, we first characterize a subset of graphs, called *admissible graphs*. A graph is admissible if none of its defined operations belongs to a cycle. We then prove the confluence, as well as the confluence modulo bisimilarity (unraveling), of the admissible graph rewriting relation. Afterwards, we define a sequential graph rewriting strategy by using Antoy's definitional trees. We show that the resulting strategy computes only needed redexes and develops optimal derivations w.r.t. the number of steps. Finally, we tackle the graph narrowing relation over admissible graphs and propose a sequential narrowing strategy which computes independent solutions and develops shorter derivations than most general graph narrowing.

1 Introduction

Many efforts have been dedicated to investigate the integration of functional and logic languages during the last decade, see [11] for a survey and [18, 12] for recent language propositions. The underlying operational semantics of such languages is often based on *first-order term* rewriting and narrowing.

In practice, data structures are represented as (cyclic) graphs. There are many reasons that motivate the use of graphs. They actually allow sharing of subexpressions which leads to efficient computations. They also permit to go beyond the processing of first-order terms by handling efficiently real-world data structures. In this paper, we consider *graph* rewriting and narrowing as the underlying operational semantics of rule-based (functional and logic) programming languages.

Using a graph rewriting system (GRS) is not an easy task. Indeed, the classical properties of term rewriting systems (TRS) cannot be lifted without caution to GRSs. One of these properties is confluence. Let us consider the rule $F(a, a, x) \to x$ where a is a constant and x is a variable. This rule, which constitutes an orthogonal TRS, generates a confluent rewrite relation over (finite or infinite) terms whereas it generates a non confluent rewrite relation over graphs [15]. It is well-known that this source of nonconfluency of GRSs comes from the so-called "collapsing rules" in orthogonal GRSs. A rewrite rule is collapsing if its right-hand side is a variable. However, collapsing rules are very often used in programming and thus cannot be prohibited in any programming discipline. Most of access functions are defined by means of collapsing rules, e.g.,

```
car (cons (x,u)) -> x       left-tree (bin-tree (l,x,r)) -> l
cdr (cons (x,u)) -> u      right-tree (bin-tree (l,x,r)) -> r
```

In practice, many programming languages are constructor-based, i.e., operators called *constructors*, which are intended to construct data structures are distinguished from operators called *defined operators* which are defined by means of rewrite rules. In this paper, we follow this discipline and define a new class of GRSs which could be seen as a natural extension to graphs of the well known class of orthogonal constructor-based TRSs. We investigate the graph rewriting and narrowing relations over a particular subset of graphs called *admissible graphs*. An admissible graph is a graph whose cycles do not include defined operators. We give a sufficient (syntactic) condition which ensures that the set of admissible graphs is closed under rewriting. Then, we show the confluence of admissible graph rewriting relation even in the presence of an arbitrary amount of collapsing rules.

It is notorious that finite graphs represent rational terms. Sometimes, one would like to equate two graphs if they represent the same rational term. In this case we say that the two graphs are *bisimilar* (if we refer to the theory of concurrency). We show the confluence modulo bisimilarity of the considered GRSs.

The confluence of a rewrite relation allows to evaluate expressions in a deterministic and efficient way by using rewriting strategies. Such strategies have been well investigated in the setting of finite and infinite orthogonal TRSs (e.g., [19, 13, 16]). In [1], a strategy that computes outermost needed redexes based on definitional trees has been designed in the framework of orthogonal constructor-based TRSs. In this article, we show that Antoy's strategy can be extended to orthogonal constructor-based GRSs with the same nice properties. We particularly prove that the resulting strategy is c-hyper-normalizing on the class of admissible graphs and develops shortest derivations.

Narrowing [14] has been intensively investigated during the last decade in the framework of TRSs. An optimal narrowing strategy is presented in [2]. However, to our knowledge, only few studies considered the extension of narrowing to graphs [9, 21, 10]. In these papers the considered graphs are acyclic and the best known strategy is basic narrowing. In the present paper, we investigate most general narrowing on admissible (cyclic) graphs and establish its soundness and completeness. Furthermore, we extend the strategy presented in [2] to admissible graphs and provide a sequential graph narrowing strategy which computes independent solutions and develops narrowing derivations which are always shorter than most general derivations.

The rest of the paper is organized as follows. We recall briefly some preliminaries on graphs in the next section. In Section 3, we introduce the framework of constructor-based GRSs and exhibits the results concerning confluence and confluence modulo bisimilarity. We define our rewriting strategy in Section 4 and list its properties. Section 5 introduces admissible graph narrowing and states the completeness of most-general graph narrowing. In Section 6, we define a sequential narrowing strategy and list its properties. Due to lack of space, some detailed definitions and all the proofs are omitted from this paper. Precise definitions and proofs can be found in [7, 6].

2 Preliminaries

Many different notations are used in the literature to investigate graph rewriting [8, 22]. The aim of this section is to recall briefly some key definitions in order to make easier the understanding of the paper. We are mostly consistent with [5].

A *many-sorted signature* $\Sigma = \langle S, \Omega \rangle$ consists of a set S of sorts and an S-indexed family of sets of operation symbols $\Omega = \uplus_{s \in S} \Omega_s$ with $\Omega_s = \uplus_{(w,s) \in S^* \times S} \Omega_{w,s}$. We shall write $f : s_1 \dots s_n \to s$ whenever $f \in \Omega_{s_1 \dots s_n, s}$ and say that f is of *sort* s and *rank* $s_1 \dots s_n$. We consider a graph as a set of nodes and edges between the nodes. Each node is labeled with an operation symbol or a variable. Let $\mathcal{X} = \uplus_{s \in S} \mathcal{X}_s$ be an S-indexed family of countable sets of *variables* and $\mathcal{N} = \uplus_{s \in S} \mathcal{N}_s$ an S-indexed family of countable sets of *nodes*. We assume that $\mathcal{X}$ and $\mathcal{N}$ are fixed throughout the rest of the paper.

A *graph* g over $\langle \Sigma, \mathcal{N}, \mathcal{X} \rangle$ is a tuple $g = \langle \mathcal{N}_g, \mathcal{L}_g, \mathcal{S}_g, \mathcal{R}\text{oots}_g \rangle$ such that $\mathcal{N}_g$ is a set of nodes, $\mathcal{L}_g : \mathcal{N}_g \to \Omega \cup \mathcal{X}$ is a *labeling function* which maps to every node of g an operation symbol or a variable, $\mathcal{S}_g$ is a *successor function* which maps to every node of g a (possibly empty) string of nodes and $\mathcal{R}\text{oots}_g$ is a set of distinguished nodes of g, called its *roots*. We also assume three conditions of well definedness. (i) Graphs are well typed : a node n is of the same sort as its label $\mathcal{L}_g(n)$, and its successors $\mathcal{S}_g(n)$ are compatible with the rank of $\mathcal{L}_g(n)$. (ii) Graphs are connected : for all nodes $n \in \mathcal{N}_g$, there exist a root $r \in \mathcal{R}\text{oots}_g$ and a path from r to n. (iii) Let $\mathcal{V}(g)$ be the set of variables of g. For all $x \in \mathcal{V}(g)$, there exists one and only one node $n \in \mathcal{N}_g$ such that $\mathcal{L}_g(n) = x$.

A *term graph* is a (possibly cyclic) graph with one root denoted $\mathcal{R}\text{oot}_g$. Two term graphs g_1 and g_2 are *bisimilar*, denoted $g_1 \doteq g_2$, iff they represent the same (infinite) tree when one unravels them [3]. We write $g_1 \sim g_2$ whenever the term graphs g_1 and g_2 are equal up to renaming of nodes.

As the formal definition of graphs is not useful to give examples, we introduce a linear notation [5]. In the following grammar, the variable A

(resp. n) ranges over the set $\Omega \cup \mathcal{X}$ (resp. $\mathcal{N}$) :

$$
\begin{array}{lcl}
\textsc{Graph} & ::= & \textsc{Node} \mid \textsc{Node} + \textsc{Graph} \\
\textsc{Node} & ::= & n{:}A(\textsc{Node},\ldots,\textsc{Node}) \mid n
\end{array}
$$

The set of roots of a graph defined with a linear expression contains the first node of the expression and all the nodes appearing just after a $+$.

Example 2.1 We define a first graph G which represents a cyclic list alternating the expressions `s(0)+s(0)` and `s(0)`. The term graph G is given by (i) $\mathcal{N}_G = \{$`n1`, …, `n5`$\}$, (ii) $\mathcal{R}\mathrm{oot}_G =$ `n1`, (iii) $\mathcal{L}_G$ is defined by $\mathcal{L}_G($`n1`$) = \mathcal{L}_G($`n5`$) =$ `c`, $\mathcal{L}_G($`n2`$) =$ `+`, $\mathcal{L}_G($`n3`$) =$ `s` and $\mathcal{L}_G($`n4`$) =$ `0` and (iv) $\mathcal{S}_G$ is defined by $\mathcal{S}_G($`n1`$) =$ `n2.n5` , $\mathcal{S}_G($`n2`$) =$ `n3.n3` , $\mathcal{S}_G($`n3`$) =$ `n4`, $\mathcal{S}_G($`n4`$) = \varepsilon$ and $\mathcal{S}_G($`n5`$) =$ `n3.n1` . An equivalent description of G is
`n1:c(n2:+(n3:s(n4:0),n3),n5:c(n3,n1))`. We define a second graph T with two roots $\{$`l1,r1`$\}$ which represents the classical rewrite rule `s(u)+v` $\rightarrow$ `s(u+v)` : $T =$ `l1:+(l2:s(l3:u),l4:v)` $+$ `r1:s(r2:+(l3,l4))`.

A *subgraph* of a graph g rooted by a node p, denoted $g_{|p}$, is built by considering p as a root and deleting all the nodes which are not accessible from p in g. The *sum* of two graphs g_1 and g_2, denoted $g_1 \oplus g_2$, is the graph whose nodes and roots are those of g_1 and g_2 and whose labeling and successor functions coincide with those of g_1 and g_2. The *replacement by a term graph u of the subgraph rooted by a node p in a term graph g*, denoted $g[p \leftarrow u]$, is built in three steps : (i) compute $g \oplus u$, (ii) redirect all the edges pointing on p to point on $\mathcal{R}\mathrm{oot}_u$ in $g \oplus u$ and (iii) delete all the "uninteresting" nodes (garbage collection).

Example 2.2 Consider the graph G of Example 2.1. The subgraph $G_{|\mathtt{n2}}$ is `n2:+(n3:s(n4:0),n3)`. Let $D =$ `r1:s(r2:+(n4:0,n3:s(n4)))`. $G \oplus D =$ `n1:c(n2:+(n3:s(n4:0),n3),n5:c(n3,n1))` $+$ `r1:s(r2:+(n4,n3))`. We now compute $G[$`n2` $\leftarrow D]$, thus we redirect all edges pointing on `n2` to point on `r1` in the graph $G \oplus D$, and we obtain
`n1:c(r1:s(r2:+(n4:0,n3:s(n4))),n5:c(n3,n1))` $+$ `r1` $+$ `n2:+(n3,n3)`.
After a garbage collection, the resulting graph is
$G[$`n2` $\leftarrow D] =$ `n1:c(r1:s(r2:+(n4:0,n3:s(n4))),n5:c(n3,n1))`.

A *(rooted) homomorphism* h from a graph g_1 to a graph g_2, denoted $h : g_1 \rightarrow g_2$, is a mapping from $\mathcal{N}_{g_1}$ to $\mathcal{N}_{g_2}$ such that $\mathcal{R}\mathrm{oots}_{g_2} = h(\mathcal{R}\mathrm{oots}_{g_1})$ and for all nodes $n \in \mathcal{N}_{g_1}$, if $\mathcal{L}_{g_1}(n) \notin \mathcal{X}$ then $\mathcal{L}_{g_2}(h(n)) = \mathcal{L}_{g_1}(n)$ and $\mathcal{S}_{g_2}(h(n)) = h(\mathcal{S}_{g_1}(n))$ and if $\mathcal{L}_{g_1}(n) \in \mathcal{X}$ then $h(n) \in \mathcal{N}_{g_2}$. If $h : g_1 \rightarrow g_2$ is a homomorphism and g is a subgraph of g_1 rooted by p, then we write $h(g)$ for the subgraph $g_{2|h(p)}$. If $h : g_1 \rightarrow g_2$ is a homomorphism and g is a graph, $h[g]$ is the graph built from g by replacing all the subgraphs shared between g and g_1 by their corresponding subgraphs in g_2. We can prove that there exists a homomorphism $h' : g \rightarrow h[g]$. h' is called the *extension of h to g*.

Example 2.3 Consider the subgraph $G_{|\texttt{n2}}$ of Example 2.2. Let $L =$ `l1:+(l2:s(l3:u),l4:v)` and μ the mapping from $\mathcal{N}_L$ to $\mathcal{N}_{(G_{|\texttt{n2}})}$ such that $\mu(\texttt{l1}) = \texttt{n2}$, $\mu(\texttt{l2}) = \mu(\texttt{l4}) = \texttt{n3}$ and $\mu(\texttt{l3}) = \texttt{n4}$. μ is a homomorphism from L to $G_{|\texttt{n2}}$. On the other hand, let $R =$ `r1:s(r2:+(l3:u,l4:v))`. R and L share the subgraphs `l3:u` and `l4:v` whose images by μ are respectively `n4:0` and `n3:s(n4:0)`, thus $\mu[R] =$ `r1:s(r2:+(n4:0,n3:s(n4)))`. The extension of μ to R is the homomorphism $\mu' : R \to \mu[R]$ such that $\mu'(\texttt{r1}) = \texttt{r1}$, $\mu'(\texttt{r2}) = \texttt{r2}$, $\mu'(\texttt{l3}) = \texttt{n4}$ and $\mu'(\texttt{l4}) = \texttt{n3}$.

A term graph l matches a graph g at node n if there exists a homomorphism $h : l \to g_{|n}$. h is called a *matcher* of l on g at node n. Two term graphs g_1 and g_2 are *unifiable* if there exist two graphs G and H and a homomorphism $h : G \to H$ such that (i) g_1 and g_2 are both subgraphs of G and (ii) $h(g_1) = h(g_2)$. h is called a *unifier* of g_1 and g_2. If g_1 and g_2 are unifiable, we can prove that there exists a *most general unifier* (*mgu*) in the following sense : there exist a graph g and a homomorphism $h : (g_1 \oplus g_2) \to g$ such that $h(g_1) = h(g_2) = g$ and for all unifiers $h' : G \to H$, there exists a homomorphism $\phi : g \to h'(g_1 \oplus g_2)$.

Example 2.4 The homomorphism μ of Example 2.3 is a matcher of L on G at node `n2`. Let $H =$ `p1:c(p2:+(p3:x,p3),p4:c(p3,p1))`. The graphs $H_{|\texttt{p2}}$ and L are unifiable. Indeed, let $H' =$ `q1:+(l2:s(l3:u),l2)` and υ the homomorphism from $(H_{|\texttt{p2}} \oplus L)$ to H' such that $\upsilon(\texttt{p2}) = \upsilon(\texttt{l1}) = \texttt{q1}$, $\upsilon(\texttt{p3}) = \upsilon(\texttt{l2}) = \upsilon(\texttt{l4}) = \texttt{l2}$ and $\upsilon(\texttt{l3}) = \texttt{l3}$. The reader may check that $\upsilon(H_{|\texttt{p2}}) = \upsilon(L) = H'$. Thus υ is a unifier of $H_{|\texttt{p2}}$ and L.

Independently of homomorphisms, we need substitutions in order to define solutions computed by narrowing. A *substitution* σ is a partial function from the set of variables $\mathcal{X}$ to a set of term graphs. $\mathcal{D}\sigma$ denotes the *domain* of σ, i.e., the set of variables x such that $\sigma(x)$ is not a graph reduced to a single node labeled with the variable x. The *restriction* of σ to a set V of variables is denoted $\sigma_{|V}$. $\mathcal{D}(\sigma_{|V}) = V \cap \mathcal{D}\sigma$ and $\sigma_{|V}(x) = \sigma(x)$ for all $x \in \mathcal{D}(\sigma_{|V})$. $\sigma(g)$ denotes the graph built from g by replacing all the variables $x \in \mathcal{D}\sigma$ by their images $\sigma(x)$. Given two term graphs g_1 and g_2, we write $g_1 \stackrel{.}{\leq} g_2$ iff there exists a substitution θ such that $\theta(g_1) \doteq g_2$. The *composition* of two substitutions σ_1 and σ_2 is the substitution $\sigma_2 \circ \sigma_1$ such that $\mathcal{D}(\sigma_2 \circ \sigma_1) = \mathcal{D}\sigma_1 \cup \mathcal{D}\sigma_2$ and $\sigma_2 \circ \sigma_1(x) = \sigma_2(\sigma_1(x))$ for all $x \in \mathcal{D}\sigma_1$ and $\sigma_2 \circ \sigma_1(x) = \sigma_2(x)$ for all $x \in (\mathcal{D}\sigma_2 - \mathcal{D}\sigma_1)$. Given two substitutions σ_1 and σ_2 and a set V of variables, we write $\sigma_1 \stackrel{.}{\leq} \sigma_2\ [V]$ iff there exists a substitution θ such that $\theta \circ \sigma_1(x) \doteq \sigma_2(x)$ for all x in V. Given a homomorphism $h : g_1 \to g_2$, we define σ_h as the substitution such that for every x in $\mathcal{V}(g_1)$ which labels some node $n \in \mathcal{N}_{g_1}$, $\sigma_h(x) = g_{2|h(n)}$.

Example 2.5 Consider the homomorphism μ of Example 2.3. σ_μ is defined by $\sigma_\mu(\texttt{u}) =$ `n4:0` and $\sigma_\mu(\texttt{v}) =$ `n3:s(n4:0)`. The reader may check that

$\sigma_\mu(L) =$ `l1:+(l2:s(n4:0),n3:s(n4))` and that $\mu[L] = G_{|\texttt{n2}} =$ `n2:+(n3:s(n4:0),n3)`. We remark that $\mu[L]$ and $\sigma_\mu(L)$ are not equal up to renaming of nodes but bisimilar. Consider now the homomorphism υ of Example 2.4. σ_υ is defined by $\sigma_\upsilon(\texttt{u}) =$ `l3:u` and $\sigma_\upsilon(\texttt{x}) = \sigma_\upsilon(\texttt{v}) =$ `l2:s(l3:u)`. We can notice that $\sigma_\upsilon \stackrel{\cdot}{\leq} \sigma_\mu\ [\{\texttt{u}, \texttt{v}\}]$.

3 Admissible graph rewriting systems

This section introduces the class of GRSs we consider. For practical reasons, many declarative languages use constructor-based signatures. A *constructor-based signature* Σ is a triple $\Sigma = \langle S, \mathcal{C}, \mathcal{D}\rangle$ where S is a set of sorts, $\mathcal{C}$ is an S-indexed family of sets of *constructor symbols*, $\mathcal{D}$ is an S-indexed family of sets of *defined operations* such that $\mathcal{C} \cap \mathcal{D} = \emptyset$ and $\langle S, \mathcal{C} \cup \mathcal{D}\rangle$ is a signature. In Example 2.1, `0`, `s` and `c` are constructor symbols and `+` is a defined operation.

In the rest of the paper, we investigate graph rewriting and narrowing for the class of what we call *admissible* term graphs (atgs). Roughly speaking, an atg corresponds to nested procedure (function) calls whose parameters are complex constructor graphs (classical data structures).

Definition 3.1 *Let g be a term graph over a constructor-based signature. A node $n \in \mathcal{N}_g$ is called a* defined node *(resp.* variable node*) if $\mathcal{L}_g(n)$ is a defined operation ($\in \mathcal{D}$) (resp. variable ($\in \mathcal{X}$)). g is an admissible term graph (atg) if there exists no path from a defined node of g to itself. An atg g is a* pattern *if g has a tree structure (i.e., linear first-order term) which contains one and only one defined operation at its root. A* constructor graph *is a graph which has no defined node. An atg is* operation-rooted *if its root is a defined node.*

The graph G of Example 2.1 is admissible, but `z:cdr(c(0,z))` and `z:+(z,z)` are not. `l1:+(l2:0,l3:0)` is a pattern, whereas `l1:+(l2:0,l2)` is not.

The next definition introduces the notion of admissible rewrite rule. Such rules are tailored so that the set of atgs is stable with respect to the rewrite relation induced by admissible rules (see (counter)Example 3.7).

Definition 3.2 *A* rewrite rule *is a graph with two roots, denoted $l \to r$. l (resp. r) is a term graph called the* left-hand side *(resp.* right-hand side*) of the rule. A rule $l \to r$ is* admissible *iff (i) l is a* pattern *(thus an atg), (ii) r is an atg, (iii) l is not a subgraph of r and (iv) $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. A rule e' is a* variant *of another rule e if e and e' are equal up to renaming of nodes and variables and all the nodes and the variables of e' are new. Two admissible rules are* non-overlapping *if their left-hand sides are not unifiable.*

By the definition of graphs, the variables occurring in a rewrite rule are always shared between the left-hand side and the right-hand side. The graph T of Example 2.1 shows an example of an admissible rule.

Definition 3.3 *An* admissible graph rewriting system (AGRS) *is a pair $SP = \langle \Sigma, \mathcal{R} \rangle$ where $\Sigma = \langle S, \mathcal{C}, \mathcal{D} \rangle$ is a constructor-based signature and $\mathcal{R}$ is a set of admissible rules such that every two distinct rules in $\mathcal{R}$ are non-overlapping.*

Below, we recall the definition of a graph rewriting step [5].

Definition 3.4 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an AGRS, g_1 an atg, g_2 a graph, $l \to r$ a variant of a rewrite rule of $\mathcal{R}$ and p a node of g_1. We say that g_1* rewrites to g_2 at node p using the rule $l \to r$ *and write $g_1 \to_{[p,l\to r]} g_2$ if there exists a homomorphism $h : l \to g_{1|p}$ (i.e., l matches g_1 at node p) and $g_2 = g_1[p \leftarrow h[r]]$. In this case, we say that $g_{1|p}$ is a* redex *of g_1 rooted by p. An atg is in* normal form *if it has no redex. $\xrightarrow{*}$ denotes the reflexive and transitive closure of $\to$.*

Example 3.5 According to the Example 2.3, L matches G at node `n2` with homomorphism μ and $\mu[R]$ = `r1:s(r2:+(n4:0,n3:s(n4)))`. As $\mu[R]$ is equal to the graph D of Example 2.2, we infer that $G[\texttt{n2} \leftarrow \mu[R]] =$ `n1:c(r1:s(r2:+(n4:0,n3:s(n4))),n5:c(n3,n1))`. Let G' be this last graph. By Definition 3.4, $G \to_{[\texttt{n2},L\to R]} G'$.

The following proposition states the stability of the set of admissible term graphs w.r.t. rewriting with admissible rules.

Proposition 3.6 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an AGRS, g_1 an atg and $g_1 \to_{[p,l\to r]} g_2$ a rewriting step. Then g_2 is an atg.*

Example 3.7 The rule `l1:+(l2:0,l3:x)` $\to$ `r1:+(l1,r2:0)` is not admissible since the root of its left-hand side is a node of its right-hand side, thus its left-hand side is a subgraph of its right-hand side, thus the Condition (iii) of Definition 3.2 is not fulfilled. By using this rule, the reader may check that the atg `x1:+(x2:0,x3:0)` rewrites to `r1:+(r1,r2:0)`. This last graph is not an atg, since there is a cycle on the defined operation `+`.

One of the main properties of GRSs is confluence. We have seen in Section 1 that the confluence of AGRSs is not a straightforward extension of the confluence of orthogonal TRSs. We establish below the confluence (modulo $\sim$ and $\doteq$) of the graph rewriting relation with respect to atgs.

Definition 3.8 *We say that the rewriting relation $\to$ is* confluent modulo renaming of nodes (resp. bisimilarity) w.r.t. atgs *iff for all atgs g_1, g_2, g'_1 and g'_2 such that $g_1 \sim g_2$ (resp. $g_1 \doteq g_2$), $g_1 \xrightarrow{*} g'_1$ and $g_2 \xrightarrow{*} g'_2$, there exist two atgs g''_1 and g''_2 such that $g'_1 \xrightarrow{*} g''_1$, $g'_2 \xrightarrow{*} g''_2$ and $g''_1 \sim g''_2$ (resp. $g''_1 \doteq g''_2$).*

Theorem 3.9 *$\to$ is confluent modulo $\sim$ and $\doteq$ w.r.t. atgs.*

4 An optimal rewriting strategy

In this section, we define an optimal rewriting strategy for atgs. A *(sequential) graph rewriting strategy* is a partial function $\mathcal{S}$ which takes an atg g and returns a pair (p, R) such that p is a node of g, R is a rewrite rule of $\mathcal{R}$ and g can be rewritten at node p with rule R. We denote by $g \rightarrow_{\mathcal{S}} g'$ the $\mathcal{S}$*-step* from g to g' such that $\mathcal{S}(g) = (p, R)$ and $g \rightarrow_{[p,R]} g'$. A derivation $g \overset{*}{\rightarrow}_{\mathcal{S}} g'$ is called an $\mathcal{S}$*-derivation.* A strategy $\mathcal{S}$ is *c-normalizing* iff for all atgs g and constructor graphs c such that $g \overset{*}{\rightarrow} c$, there exists a constructor graph c' such that $g \overset{*}{\rightarrow}_{\mathcal{S}} c'$ and $c' \sim c$. A strategy $\mathcal{S}$ is *c-hyper-normalizing* iff for all atgs g and constructor graphs c such that $g \overset{*}{\rightarrow} c$, every derivation starting with g which alternates $\mathcal{S}$-steps with some arbitrary reductions ends with a constructor normal form c' such that $c' \sim c$.

Definition 4.1 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an AGRS, g and g' two atgs and $B = g \overset{*}{\rightarrow} g'$ a rewriting derivation. A defined node q in g* is a residual node by B *if q remains a node of g'. We call* descendant of $g_{|q}$ *the subgraph $(g')_{|q}$. A redex u rooted by q in g is a* needed redex *iff in every rewriting derivation from g to a constructor normal form, a descendant of $g_{|q}$ is rewritten at its root q. A redex u rooted by q in g is an* outermost redex *iff $q = \mathcal{R}oot_g$ when g is a redex or else $\mathcal{S}_g(\mathcal{R}oot_g) = r_1 \ldots r_k$ and u is an outermost redex of $g_{|r_i}$ for some $i \in 1..k$. A node q is* the leftmost-outermost defined node *of g iff $q = \mathcal{R}oot_g$ when $\mathcal{R}oot_g$ is a defined node or else $\mathcal{S}_g(\mathcal{R}oot_g) = r_1 \ldots r_k$ and there exists $i \in 1..k$ such that q is the leftmost-outermost defined node of $g_{|r_i}$ and the subgraphs $g_{|r_j}$ are constructor graphs for all $j < i$.*

Example 4.2 Let $g =$ `x1:c(x2:*(x3:0,x4:+(x3,x3)),x5:c(x4,x1))`. The leftmost-outermost node of g is `x2`. $g_{|\texttt{x2}}$ and $g_{|\texttt{x4}}$ are outermost needed redexes (though there exists a path from `x2` to `x4`).

The notions of leftmost-outermost defined node and outermost redex are well-defined in the framework of atgs. Indeed, if p and q are two defined nodes of an atg such that there exists a path from p to q (i.e., p is *outer* than q), then there is no path from q to p.

Our graph rewriting strategy is based on definitional trees [1]. A definitional tree is a hierarchical structure whose leaves are the rules of an AGRS used to define a given operation. In the following definition, *branch* and *rule* are uninterpreted symbols, used to construct the nodes of a definitional tree.

Definition 4.3 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an AGRS. A tree $\mathcal{T}$ is a* partial definitional tree, *or* pdt, *with pattern π iff one of the following cases holds :*

- *$\mathcal{T} = rule(\pi \rightarrow r)$, where $\pi \rightarrow r$ is a variant of a rule of $\mathcal{R}$.*
- *$\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, where o is a variable node of π, o is of sort s, $c_1, \ldots, c_k$ ($k > 0$) are different constructors of the sort s and for all $j \in 1..k$, $\mathcal{T}_j$ is a pdt with pattern $\pi[o \leftarrow p : c_j(o_1 : X_1, \ldots, o_n : X_n)]$,*

such that n is the number of arguments of c_j, $X_1, \ldots, X_n$ are new variables and $p, o_1, \ldots, o_n$ are new nodes.

We write pattern($\mathcal{T}$) to denote the pattern argument of a pdt. A definitional tree *$\mathcal{T}$ of a defined operation f is a* finite *pdt with a pattern of the form $p : f(o_1 : X_1, \ldots, o_n : X_n)$ where n is the number of arguments of f, $X_1, \ldots, X_n$ are new variables, $p, o_1, \ldots, o_n$ are new nodes, and for every rule $l \to r$ of $\mathcal{R}$, with l of the form $f(g_1, \ldots, g_n)$, there exists a leaf rule($l' \to r'$) of $\mathcal{T}$ such that $l' \to r'$ is a variant of $l \to r$. An AGRS is* inductively sequential *(IGRS) iff for every defined operation f, there exists a definitional tree of f.*

Example 4.4 Consider the following AGRS where `0` and `s` are constructors and `p`, `q`, `r`, `u`, `v`, `w` are variables.

```
(R0)  n1:f(n2:s(n3:p),n4:q) -> n3
(R1)  m1:f(m2:0,m3:r) -> m3
(R2)  l1:g(l2:w,l3:s(l4:u),l5:s(l6:v)) -> l2
```

The definitional trees $\mathcal{T}_{\texttt{f}}$ and $\mathcal{T}_{\texttt{g}}$ of the operations `f` and `g` are

$$\begin{aligned}\mathcal{T}_{\texttt{f}} = \mathit{branch}(\ & \texttt{k1:f(}\underline{\texttt{k2:X1}}\texttt{,k3:X2),k2,}\\ & \mathit{rule}(\texttt{m1:f(m2:0,m3:r)} \to \texttt{m3}),\\ & \mathit{rule}(\texttt{n1:f(n2:s(n3:p),n4:q)} \to \texttt{n3}))\end{aligned}$$

$$\begin{aligned}\mathcal{T}_{\texttt{g}} = \mathit{branch}(\ & \texttt{k4:g(k5:X3,}\underline{\texttt{k6:X4}}\texttt{,k7:X5),k6,}\\ & \mathit{branch}(\ \texttt{k8:g(k9:X6,k10:s(k11:X7),}\underline{\texttt{k12:X8}}\texttt{),k12,}\\ & \qquad \mathit{rule}(\texttt{l1:g(l2:w,l3:s(l4:u),l5:s(l6:v))} \to \texttt{l2})))\end{aligned}$$

We are ready to define our graph rewriting strategy Φ. The strategy Φ is a partial function that operates on atgs in the presence of IGRSs. $\Phi(g)$ returns, when it is possible, a pair (p, R) where p is a node of g and R is a rewrite rule such that g can be rewritten at node p with rule R. Φ uses an auxiliary function φ which takes two arguments : an operation-rooted atg and a pdt of this operation.

Definition 4.5 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an IGRS and g an atg. Φ is the partial function defined by $\Phi(g) = \varphi(g_{|p}, \mathcal{T})$, where p is the leftmost-outermost defined node of g and $\mathcal{T}$ is a definitional tree of the label of p.*
Let g be an operation-rooted atg and $\mathcal{T}$ a pdt such that pattern($\mathcal{T}$) matches g at the root. We define the partial function φ by :

$$\varphi(g, \mathcal{T}) = \begin{cases} (p, R) & \text{if } \mathcal{T} = rule(\pi \to r),\ p = \mathcal{R}oot_g \text{ and} \\ & \quad R \text{ is a variant of } \pi \to r\ ; \\ \varphi(g, \mathcal{T}_i) & \text{if } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ and} \\ & \quad pattern(\mathcal{T}_i) \text{ matches } g \text{ for some } i \in 1..k\ ; \\ (p, R) & \text{if } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad \pi \text{ matches } g \text{ by a homomorphism } h : \pi \to g, \\ & \quad h(o) \text{ is labeled with a defined operation } f \text{ in } g, \\ & \quad \mathcal{T}' \text{ is a definitional tree of } f \text{ and} \\ & \quad \varphi(g_{|h(o)}, \mathcal{T}') = (p, R). \end{cases}$$

Example 4.6 Consider the AGRS defined in Example 4.4 and the atg $t_1 =$ `x1:g(x2:s(x2),x2,x3:f(x2,x2))`.

$$\begin{array}{rcl}\Phi(t_1) & = & \varphi(t_1, \mathcal{T}_{\texttt{g}}) \\ & = & \varphi(t_1, branch(\texttt{k4:g(k5:X3,}\underline{\texttt{k6:X4}}\texttt{,k7:X5)}, \texttt{k6}, \ldots)) \\ & = & \varphi(t_1, branch(\texttt{k8:g(k9:X6,k10:s(k11:X7),}\underline{\texttt{k12:X8}}\texttt{)}, \texttt{k12}, \ldots))\end{array}$$

The pattern of $rule$(`l1:g(l2:w,l3:s(l4:u),l5:s(l6:v))` $\rightarrow$ `l2`) does not match t_1 because `l5` is labeled with `s` whereas `x3` is labeled with `f`. So $\Phi(t_1)$ recursively calls $\varphi(t'_1, \mathcal{T}_{\texttt{f}})$ where $t'_1 = t_{1|\texttt{x3}} =$ `x3:f(x2:s(x2),x2)`. The reader may check that $\Phi(t_1) = \varphi(t'_1, \mathcal{T}_{\texttt{f}}) = (\texttt{x3}, \texttt{R0})$.

Below, we list some properties of Φ.

Proposition 4.7 *If $\Phi(g) = (p, R)$, then $g_{|p}$ is an outermost needed redex of g and g can be rewritten at node p with rule R. If $\Phi(g)$ is not defined, then g has no constructor normal form.*

Theorem 4.8 *Φ is c-hyper-normalizing (thus c-normalizing).*

Graph rewriting does not duplicate data. Thus the number of rewriting steps which are necessary to compute a constructor normal form may be optimized.

Theorem 4.9 *Let g be an atg and c a constructor graph such that there exists a rewriting derivation $g \overset{*}{\rightarrow} c$. Then there exists a constructor graph c' with $c' \sim c$ such that the length of the Φ-derivation $g \overset{*}{\rightarrow}_{\Phi} c'$ is less than (or equal to) the length of the derivation $g \overset{*}{\rightarrow} c$.*

5 Admissible graph narrowing

Besides rewriting, narrowing is the basis of many functional logic programming languages [11]. In this section we generalize narrowing to admissible term graphs.

Definition 5.1 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an AGRS, g_1 an atg, $l \rightarrow r$ a variant of a rewrite rule of $\mathcal{R}$, p a non variable node of g_1, $h : G \rightarrow H$ a homomorphism and g_2 a graph. We say that g_1* narrows to g_2 at node p using the rule $l \rightarrow r$ and the homomorphism h, *denoted $g_1 \leadsto_{[p, l \rightarrow r, h]} g_2$, if l and $g_{1|p}$ are unifiable, h is a unifier of l and $g_{1|p}$ and $g_2 = h[g_1][h(p) \leftarrow h[r]]$. A narrowing step is* most general *when the unifier, h, is an mgu.*

Example 5.2 In Example 2.4, $H_{|\texttt{p2}}$ and L are unifiable with mgu υ. The reader may check that $\upsilon(\texttt{p2}) = \texttt{q1}$, $\upsilon[R] =$ `r1:s(r2:+(l3:u,l2:s(l3))` and $\upsilon[H] =$ `p1:c(q1:+(l2:s(l3:u),l2),p4:c(l2,p1))`. Let $H_1 = \upsilon[H][\upsilon(\texttt{p2}) \leftarrow \upsilon[R]] =$ `p1:c(r1:s(r2:+(l3:u,l2:s(l3))),p4:c(l2,p1))`. By Definition 5.1, $H \leadsto_{[\texttt{p2}, L \rightarrow R, \upsilon]} H_1$.

In general, the set of atgs is not stable w.r.t. narrowing.

Example 5.3 Let g = `n1:f(n2:x)` and $R = l \to r$ a rule such that l = `l1:f(l2:0)` and r = `r1:g(r2:f(r3:0))`. g and l are unifiable, say with $h : (g \oplus l) \to r_{|\texttt{r2}}$. Let $g' = h[g][h(\texttt{n1}) \leftarrow h[r]]$. We can show that $h[g] = r_{|\texttt{r2}}$, $h(\texttt{n1}) = \texttt{r2}$ and $h[r] = r$, thus g' = `r1:g(r1)`. By Definition 5.1, $g \rightsquigarrow_{[\texttt{n1}, R, h]} g'$. However, g is admissible whereas g' is not.

Under some technical assumptions on the unifiers detailed in [6], we can prove the following proposition.

Proposition 5.4 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an AGRS, g_1 an atg and $g_1 \rightsquigarrow_{[p, l \to r, h]} g_2$ a most general narrowing step. Then g_2 is an atg.*

Narrowing is used to solve goals. A solution of a goal is often represented by a substitution. A substitution computed by a narrowing derivation is defined as follows.

Definition 5.5 *Let g_1 and g_2 be two atgs and σ a substitution. We say that* σ is computed by a narrowing derivation from g_1 to g_2 *and write $g_1 \overset{*}{\rightsquigarrow}_\sigma g_2$, if there exists a derivation $g_1 \rightsquigarrow_{[p_1, l_1 \to r_1, h_1]} \cdots \rightsquigarrow_{[p_k, l_k \to r_k, h_k]} g_2$ and $\sigma = (\sigma_{h_k} \circ \ldots \circ \sigma_{h_1})_{|\mathcal{V}(g_1)}$.*

Example 5.6 Let $H \rightsquigarrow_{[\texttt{p2}, L \to R, \upsilon]} H_1$ be the narrowing step of Example 5.2. We saw in Example 2.5 that $\sigma_\upsilon(\texttt{x})$ = `l2:s(l3:u)`. Consider a second narrowing step starting with H_1 and the rule $L' \to R'$ where L' = `l5:+(l6:0,l7:w)` and R' = `l7:w`. $H_{1|\texttt{r2}}$ and L' unify and an mgu is $\upsilon' : (H_{1|\texttt{r2}} \oplus L') \to H'_1$ with H'_1 = `q2:+(l6:0,q3:s(l6))`. Therefore, $H_1 \rightsquigarrow_{[\texttt{r2}, L' \to R', \upsilon']} H_2$ where H_2 = `p1:c(r1:s(q3:s(l6:0)),p4:c(q3,p1))`. From υ', we deduce that $\sigma_{\upsilon'}(\texttt{u})$ = `l6:0`. The reader may check that $\sigma = (\sigma_{\upsilon'} \circ \sigma_\upsilon)_{|\{\texttt{x}\}}$ is defined by $\sigma(\texttt{x})$ = `l2:s(l6:0)`. By Definition 5.5, σ is computed by the narrowing derivation from H to H_2.

We define below the notions of soundness and completeness of narrowing. These definitions do not consider narrowing as an inference rule for solving some particular goals but rather as a general computational model for arbitrary expressions (graphs). The traditional goals such as equations can be represented as boolean expressions.

Definition 5.7 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an AGRS. We say that a narrowing calculus is* sound *iff for all atgs g, constructor graphs c and substitutions θ such that $g \overset{*}{\rightsquigarrow}_\theta c$, there exists a constructor graph c' such that $\theta(g) \overset{*}{\to} c'$ and $c' \doteq c$. We say that a narrowing calculus is* complete *iff for all atgs g, constructor graphs c and constructor substitutions θ such that $\theta(g) \overset{*}{\to} c$, there exist a constructor graph c' and a substitution σ such that $g \overset{*}{\rightsquigarrow}_\sigma c'$, $c' \dot{\leq} c$ and $\sigma \dot{\leq} \theta$ $[\mathcal{V}(g)]$.*

Theorem 5.8 *The most general narrowing is sound and complete.*

6 An optimal narrowing strategy

In this section, we define a sequential narrowing strategy Λ which generalizes the rewriting strategy Φ to narrowing. A *(sequential) graph narrowing strategy* is a partial function $\mathcal{S}$ which takes an atg g and returns a set of triples of the form (p, R, h) such that g is narrowable at node p using the rule R and homomorphism h. We denote by $g \rightsquigarrow_{\mathcal{S}} g'$ an $\mathcal{S}$*-step* from g to g' such that $\mathcal{S}(g) \ni (p, R, h)$ and $g \rightsquigarrow_{[p,R,h]} g'$. A derivation $g \overset{*}{\rightsquigarrow}_{\mathcal{S}} g'$ is called an $\mathcal{S}$*-derivation.*

Definition 6.1 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an IGRS and g an atg. Λ is the partial function such that $(n, R, h') \in \Lambda(g)$ iff there exists $(n, R, h) \in \lambda(g_{|p}, \mathcal{T})$, p is the leftmost-outermost defined node of g, $\mathcal{T}$ is a definitional tree of the label of p and h' is the extension of h to $g \oplus l$.*
Let g be an operation-rooted atg and $\mathcal{T}$ a pdt such that pattern$(\mathcal{T})$ and g are unifiable. $\lambda(g, \mathcal{T})$ is a set of triples of the form (n, R, h), where n is a non variable node of g, R is a rewrite rule and h is a unifier of $g_{|n}$ and the left-hand side of R. $\lambda(g, \mathcal{T})$ is defined as the smallest set such that :

$$\lambda(g,\mathcal{T}) \supseteq \begin{cases} \{(p,R,h)\} & \textit{if } \ \mathcal{T} = \textit{rule}(\pi \to r),\ p = \mathcal{R}oot_g,\ R = \pi' \to r' \textit{ is} \\ & \quad \textit{a variant of } \pi \to r \textit{ and } h \textit{ is an mgu of } \pi' \textit{ and } g\ ; \\ \lambda(g,\mathcal{T}_i) & \textit{if } \ \mathcal{T} = \textit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \textit{ and} \\ & \quad \textit{pattern}(\mathcal{T}_i) \textit{ and } g \textit{ unify for some } i; \\ \{(p,R,h)\} & \textit{if } \ (1)\ \mathcal{T} = \textit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad (2)\ g \textit{ and } \pi \textit{ are unifiable with} \\ & \quad\quad \textit{most general unifier } \tau : (g \oplus \pi) \to g', \\ & \quad (3)\ \tau(o) \textit{ is labeled with a defined operation } f, \\ & \quad (4)\ \mathcal{T}' \textit{ is a definitional tree of } f, \\ & \quad (5)\ (p', R, h') \in \lambda(g'_{|\tau(o)}, \mathcal{T}'), \\ & \quad (6)\ p \textit{ is the node of } g \textit{ such that } \tau(p) = p', \\ & \quad (7)\ R \textit{ is of the form } l \to r, \\ & \quad (8)\ h = h'' \circ \tau' \textit{ where } h'' \textit{ is the extension} \\ & \quad\quad \textit{of } h' \textit{ to } g' \oplus l \textit{ and } \tau' \textit{ is the extension} \\ & \quad\quad \textit{of } \tau_{|\mathcal{R}oot_g} \textit{ to } g \oplus l. \end{cases}$$

Example 6.2 Consider the AGRS defined in Example 4.4 and the atg $s_1 =$ `y1:g(y2:s(y3:s(y2)),y4:z,y5:f(y6:x,y7:s(y8:y)))`. As the leftmost-outermost defined node of s_1 is its root `y1`, we infer that

$$\begin{aligned} \Lambda(s_1) &= \lambda(s_1, \mathcal{T}_{\mathtt{g}}) \\ &= \lambda(s_1, \textit{branch}(\mathtt{k4:g(k5:X3,}\underline{\mathtt{k6:X4}}\mathtt{,k7:X5),k6,}\ldots)) \\ &= \lambda(s_1, \textit{branch}(\mathtt{k8:g(k9:X6,k10:s(k11:X7),}\underline{\mathtt{k12:X8}}\mathtt{),k12,}\ldots)) \end{aligned}$$

Let $\pi =$ `k8:g(k9:X6,k10:s(k11:X7),k12:X8)`. s_1 and π are unifiable with most general unifier $\tau : (s_1 \oplus \pi) \to s'_1$ where
$s'_1 =$ `p1:g(y2:s(y3:s(y2)),k10:s(k11:X7),y5:f(y6:x,y7:s(y8:y)))`. τ

is precisely defined by $\tau(\texttt{y1}) = \tau(\texttt{k8}) = \texttt{p1}$, $\tau(\texttt{y2}) = \tau(\texttt{k9}) = \texttt{y2}$, $\tau(\texttt{y3}) = \texttt{y3}$, $\tau(\texttt{y4}) = \tau(\texttt{k10}) = \texttt{k10}$, $\tau(\texttt{k11}) = \texttt{k11}$, $\tau(\texttt{y5}) = \tau(\texttt{k12}) = \texttt{y5}$, $\tau(\texttt{y6}) = \texttt{y6}$, $\tau(\texttt{y7}) = \texttt{y7}$ and $\tau(\texttt{y8}) = \texttt{y8}$. Notice that the variable `z` which labels the node `y4` in s_1 is assigned with the constructor graph `k10:s(k11:X7)` by σ_τ. s_1 does not unify with the pattern of
$\mathit{rule}(\texttt{l1:g(l2:w,l3:s(l4:u),l5:s(l6:v))} \rightarrow \texttt{l2})$. Indeed, `l5` is labeled with `s` whereas `y5` is labeled with `f`. So $\Lambda(s_1)$ recursively calls $\lambda(g_1, \mathcal{T}_{\texttt{f}})$ where $g_1 = s'_{1|\tau(\texttt{k12})} = s'_{1|\texttt{y5}} = \texttt{y5:f(y6:x,y7:s(y8:y))}$.

$$\begin{array}{rcl} \lambda(g_1, \mathcal{T}_{\texttt{f}}) & = & \lambda(g_1, \mathit{branch}(\texttt{k1:f(}\underline{\texttt{k2:X1}}\texttt{,k3:X2),k2,}\ldots)) \\ & \supseteq & \lambda(g_1, \mathit{rule}(\texttt{n1:f(n2:s(n3:p),n4:q)} \rightarrow \texttt{n3})) \end{array}$$

Let l be the pattern of $\mathit{rule}(\texttt{n1:f(n2:s(n3:p),n4:q)} \rightarrow \texttt{n3})$, i.e., the left-hand side of the rule `R0` defined in Example 4.4. g_1 and l are unifiable and a most general unifier is the homormophism $h' : (g_1 \oplus l) \rightarrow e$ where $e = \texttt{p2:f(n2:s(n3:p),y7:s(y8:y))}$. The homomorphism h' is defined by $h'(\texttt{y5}) = h'(\texttt{n1}) = \texttt{p2}$, $h'(\texttt{y6}) = h'(\texttt{n2}) = \texttt{n2}$, $h'(\texttt{n3}) = \texttt{n3}$, $h'(\texttt{y7}) = h'(\texttt{n4}) = \texttt{y7}$ and $h'(\texttt{y8}) = \texttt{y8}$. By the definition of λ, we conclude that $\lambda(s'_{1|\tau(\texttt{k12})}, \mathcal{T}_{\texttt{f}}) = \lambda(g_1, \mathcal{T}_{\texttt{f}}) \supseteq \{(\texttt{y5}, \texttt{R0}, h')\}$. h' is a homomorphism from the graph $s'_{1|\tau(\texttt{k12})} \oplus l$ to the graph e. We now compute the extension of h' to the graph $s'_1 \oplus l$. We thus obtain a homomorphism $h'' : (s'_1 \oplus l) \rightarrow e'$ where $e' =$ `p1:g(y2:s(y3:s(y2)),k10:s(k11:X7),p2:f(n2:s(n3:p),y7:s(y8:y)))` $+$ `p2` and h'' is defined by $h''(\texttt{p1}) = \texttt{p1}$, $h''(\texttt{y2}) = \texttt{y2}$, $h''(\texttt{y3}) = \texttt{y3}$, $h''(\texttt{k10}) = \texttt{k10}$, $h''(\texttt{k11}) = \texttt{k11}$, $h''(\texttt{y5}) = h''(\texttt{n1}) = \texttt{p2}$, $h''(\texttt{y6}) = h''(\texttt{n2}) = \texttt{n2}$, $h''(\texttt{n3}) = \texttt{n3}$, $h''(\texttt{y7}) = h''(\texttt{n4}) = \texttt{y7}$ and $h''(\texttt{y8}) = \texttt{y8}$. Let τ' be the extension of $\tau_{|\mathcal{R}\mathrm{oot}_{s_1}}$ to $s_1 \oplus l$, i.e., τ' is the homomorphism from $s_1 \oplus l$ to $s'_1 \oplus l$ such that $\tau'(s_1) = s'_1$ and $\tau'(l) = l$. Let $h = h'' \circ \tau'$. h is a homomorphism from $s_1 \oplus l$ to e' such that $h(\texttt{y1}) = \texttt{p1}$, $h(\texttt{y2}) = \texttt{y2}$, $h(\texttt{y3}) = \texttt{y3}$, $h(\texttt{y4}) = \texttt{k10}$, $h(\texttt{y5}) = h(\texttt{n1}) = \texttt{p2}$, $h(\texttt{y6}) = h(\texttt{n2}) = \texttt{n2}$, $h(\texttt{n3}) = \texttt{n3}$, $h(\texttt{y7}) = h(\texttt{n4}) = \texttt{y7}$ and $h(\texttt{y8}) = \texttt{y8}$. It appears that `y5` is the only node of s_1 such that $\tau(\texttt{y5}) = \texttt{y5}$. By the definition of Λ, we conclude that $\Lambda(s_1) \supseteq \{(\texttt{y5}, \texttt{R0}, h)\}$. The reader may check that s_1 narrows at node `y5` using the rule `R0` and the homomorphism h into the graph $s_2 =$ `p1:g(y2:s(y3:s(y2)),k10:s(k11:X7),n3:p)`.

In the following we list some properties of Λ.

Proposition 6.3 *Let g_1 be an atg such that $g_1 \rightsquigarrow_\Lambda g_2$. Then g_2 is an atg.*

Definition 6.4 *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an AGRS and g and g' two atgs. A narrowing step $g \rightsquigarrow_{[p,R,h]} g'$ is* outermost needed *iff for all substitutions η such that $\sigma_h \stackrel{.}{\leq} \eta\ [\mathcal{V}(g)]$, $\eta(g_{|p})$ is an outermost needed redex of $\eta(g)$.*

Proposition 6.5 *If $\Lambda(g) \ni (p, R, h)$, then $g \rightsquigarrow_{[p,R,h]} g'$ is an outermost needed narrowing step. If $\Lambda(g)$ is not defined, then g cannot be narrowed into a constructor graph.*

Theorem 6.6 *The narrowing relation induced by Λ is sound and complete.*

Λ always computes independent substitutions, that is to say Λ never performs redundant calculus. We say that two substitutions θ and θ' are *independent* on a set V of variables iff there exists $x \in V$ such that $\theta(x)$ and $\theta'(x)$ are not unifiable.

Theorem 6.7 *Let g be an atg, c and c' two constructor graphs and $g \overset{*}{\rightsquigarrow}_{\sigma} c$ and $g \overset{*}{\rightsquigarrow}_{\sigma'} c'$ two distinct Λ-derivations. Then σ and σ' are independent on $\mathcal{V}(g)$.*

The restriction of Λ to first-order terms always develops shortest narrowing derivations [2]. This is not the case in our framework, as it is witnessed by the following (counter-)example.

Example 6.8 Consider the graph $g =$ `n1:f(n2:g(n3:x),n4:g(n3))` and the rules `l4:g(l5:w)` $\rightarrow$ `l5` and `l1:f(l2:u,l3:v)` $\rightarrow$ `r1:c(l2,l3)`. The following Λ-derivation is of length 3 : $g \rightsquigarrow_{\Lambda}$ `r1:c(n2:g(n3:x),n4:g(n3))` $\rightsquigarrow_{\Lambda}$ `r1:c(n3:x,n4:g(n3))` $\rightsquigarrow_{\Lambda}$ `r1:c(n3:x,n3)`. But arbitrary unifiers can collapse nodes labeled with defined operations. The following derivation is of length 2 : $g \rightsquigarrow$ `r1:c(r2:g(n3:x),r2)` $\rightsquigarrow$ `r1:c(n3:x,n3)`.

Nevertheless, the derivations developed by Λ are always shorter than those developed by most general narrowing.

Theorem 6.9 *Let g be an atg, σ and σ' two substitutions and c and c' two constructor graphs such that $A = g \overset{*}{\rightsquigarrow}_{\sigma} c$ is a Λ-derivation, $B = g \overset{*}{\rightsquigarrow}_{\sigma'} c'$ is a most general narrowing derivation and σ and σ' are equal up to renaming of nodes. Then the length of A is less than (or equal to) the length of B (and $c \sim c'$).*

7 Conclusion

We defined the class of admissible graph rewriting systems, AGRSs, and characterized a subset of graphs called admissible graphs for which graph rewriting is confluent (modulo renaming and bisimulation) even in the presence of collapsing rules. We defined a new sequential graph rewriting strategy for the class of inductively sequential AGRSs which is c-normalizing and optimal for the class of admissible graphs. The use of definitional trees allows to combine the elegance of neededness with an efficient implementation by pattern-matching. In [17], a lazy graph rewriting strategy close to ours is described, namely the annotated functional strategy, which combines the discriminating position strategy [20] and rewriting with priority [4]. To our knowledge, no formal result has been proved regarding this strategy. In addition, we considered the extension of narrowing to admissible graphs and established the completeness and the soundness of most general narrowing.

We also used definitional trees to define a sequential graph narrowing strategy which is complete and sound, computes only independent substitutions and develops derivations which are shorter than most general narrowing. As far as we are aware of, the graph narrowing strategies presented in this paper are the first ones that handle graphs with cycles.

Acknowledgements

The first author is very indebted to Sergio Antoy and Michael Hanus for the many fruitful discussions. We are also grateful to the anonymous referees for their helpful comments.

References

[1] S. Antoy. Definitional trees. In *ALP'92*, pages 143–157. LNCS 632.

[2] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. In *POPL'94*, pages 268–279, Portland.

[3] Z.M. Ariola and J.W. Klop. Equational term graph rewriting. *Fundamenta Informaticae*, 26(3-4), 1996.

[4] J.C. Baeten, J.A. Bergstra, and J.W. Klop. Term rewriting systems with priorities. In *RTA'87*, pages 83–94, Bordeaux. LNCS 256.

[5] H. Barendregt, M. van Eekelen, J. Glauert, R. Kenneway, M. J. Plasmeijer, and M. Sleep. Term graph rewriting. In *PARLE'87*, pages 141–158. LNCS 259.

[6] R. Echahed and J.C. Janodet. Introducing graphs in functional logic programming languages. Technical report, IMAG, 1997. Available via URL : ftp://ftp.imag.fr/pub/LEIBNIZ/ATINF/c-graph-narrowing.ps.gz.

[7] R. Echahed and J.C. Janodet. On constructor-based graph rewriting systems. Technical report, IMAG, 1997. Available via URL : ftp://ftp.imag.fr/pub/LEIBNIZ/ATINF/c-graph-rewriting.ps.gz.

[8] H. Ehrig and G. Taentzer. Computing by graph transformation : A survey and annotated bibliography. *Bulletin of the EATCS*, 59:182–226, June 1996.

[9] A. Habel and D. Plump. Term graph narrowing. *Mathematical Structures in Computer Science*, 6:649–676, 1996.

[10] A. Habel and D. Plump. Completeness of narrowing in non-copying implementations. In *not yet published*, 1997.

[11] M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994.

[12] M. Hanus. Curry: An integrated functional logic language. Available at `http://www-i2.informatik.rwth-aachen.de/~hanus/curry`, 1997.

[13] G. Huet and J.-J. Lévy. Computations in orthogonal term rewriting systems. In J.-L. Lassez and G. Plotkin, editors, *Computational logic: essays in honour of Alan Robinson*. MIT Press, Cambridge, MA, 1991.

[14] J.-M. Hullot. Canonical forms and unification. In *CADE'80*, pages 318–334. LNCS 87.

[15] J. R. Kennaway, J. K. Klop, M. R. Sleep, and F. J. De Vries. On the adequacy of graph rewriting for simulating term rewriting. *ACM Trans. on Programming Languages and Systems*, 16(3):493–523, 1994.

[16] J. R. Kennaway, J. K. Klop, M. R. Sleep, and F. J. De Vries. Transfinite reduction in orthogonal term rewriting systems. *Information and Computation*, pages 18–38, 1995.

[17] P.W. Koopman, J.E. Smetsers, M.C. van Eekelen, and M.J. Plasmeijer. Graph rewriting using the annotated functional strategy. In [22], chapter 23, pages 317–332.

[18] J. W. Lloyd. Combining functional and logic languages. In *Proc. of Int. Logic Programming Symp.*, pages 43–57, 1994.

[19] M. O'Donnell. *Computing in Systems Described by Equations.* LNCS 58.

[20] R. Plasmeijer and M. van Eekelen. *Functional Programming and Parallel Graph Rewriting.* Addison-Wesley, 1993.

[21] M.R.K. Krishna Rao. Completeness results for basic narrowing in non-copying implementations. In *Proc. of Joint Int. Conf. and Symp. on Logic Programming*, pages 393–407. MIT press, 1996.

[22] M. R. Sleep, M. J. Plasmeijer, and M. C. J. D. van Eekelen, editors. *Term Graph Rewriting. Theory and Practice.* J. Wiley & Sons, Chichester, UK, 1993.