Complete Strategies for Admissible-Graph Collapsing Narrowing

Rachid Echahed and Jean-Christophe Janodet

Laboratoire Leibniz – Institut IMAG, CNRS
46, Av. Felix Viallet, F-38031 Grenoble – France
Tel: (+33) 4 76 57 48 91; Fax: (+33) 4 76 57 46 02
Rachid.Echahed@imag.fr Jean-Christophe.Janodet@imag.fr

Abstract Narrowing constitutes the basis of the operational semantics of modern declarative languages which integrate functional and logic programming paradigms. Efficient implementations of these languages consider first-order terms as graphs. In this paper, we investigate narrowing in the setting of graph rewriting systems. We take the full advantage of graph structures by allowing maximal sharing of subexpressions and extend the completeness results of the best known narrowing strategies such as needed narrowing or parallel narrowing to the case of constructor-based weakly admissible graph rewrite systems. The resulting graph narrowing strategies share the same optimality results as the corresponding ones for first-order terms and in addition develop shorter derivations.

1 Introduction

The operational semantics of modern declarative programming languages which integrate functional and logic programming paradigms (e.g., Curry [14]) are mainly based on narrowing. Classical declarative languages encode data by means of first-order terms. In this setting, optimal narrowing strategies have been proposed [2, 3]. However, in practice, for efficiency reasons as well as expressivity, the implementation of declarative programming languages encodes data as graphs (e.g., [19]).

There are many advantages in using graphs in declarative languages. They represent real-world data types as in classical imperative programming languages. Graph structures allow also to optimize the storage of information by sharing common subexpressions. A graph g_1 collapses (or folds) into a graph g_2 if g_1 and g_2 represent the same knowledge and the size of g_2 is smaller than the size of g_1. A graph is called maximally collapsed if it cannot be compressed any further.

Narrowing a graph, g, consists in rewriting an instance of it, $\sigma(g)$ [7]. In this paper, we investigate the integration of graph collapsing and graph

narrowing. In other words, we would like to perform narrowing steps over (possibly maximally) collapsed graphs. The motivation of this work comes from the fact that non-tractable (exponential) computations over terms may turn to tractable (polynomial) ones over graphs. Consider, for instance, the case of the Fibonacci function [20] (see Fig. 1). It is clear that the processing of collapsed graphs improves drastically the operational semantics of functional logic programming languages.

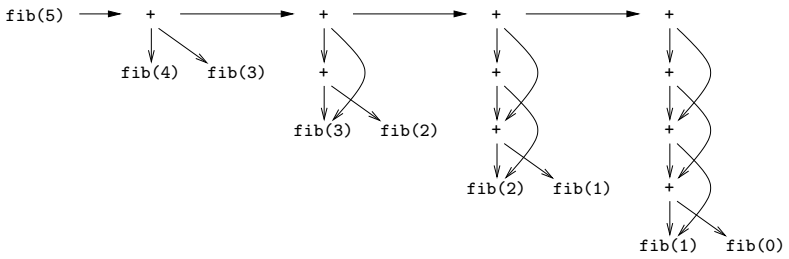

Figure1.

In Fig. 2, we show three different narrowing derivations starting with the term `C(A(X),A(O),A(O))` and using the rule `A(O)` $\rightarrow$ `O`. The first one corresponds to a term narrowing derivation; its length is 3. The second one corresponds to a simple graph narrowing derivation. Its length is 2. The third one corresponds to a maximally collapsing graph narrowing derivation. Its length is 1. The reader may notice that the more collapsed are the graphs, the shorter are the narrowing derivations.

A first step towards the investigation of graph narrowing has been done in [21, 17, 12] for acyclic graphs and in [7] for cyclic admissible graphs. Integration of graph collapsing and graph narrowing has been considered recently [17, 13] in the particular case of acyclic term graphs. As for graph narrowing strategies, only basic narrowing was considered in the literature [17, 13]. However, basic narrowing was the first term narrowing strategy to be studied [15, 18]. It has not the optimality properties of modern narrowing strategies such as needed narrowing [2] or parallel narrowing [3].

In this paper, we take the full advantage of the graph structure and investigate the first cyclic admissible graph collapsing narrowing relations and strategies. For efficiency reasons, we establish our results in the setting of weakly admissible graph rewriting systems, a restricted class of constructor-based graph rewriting systems which is commonly used in

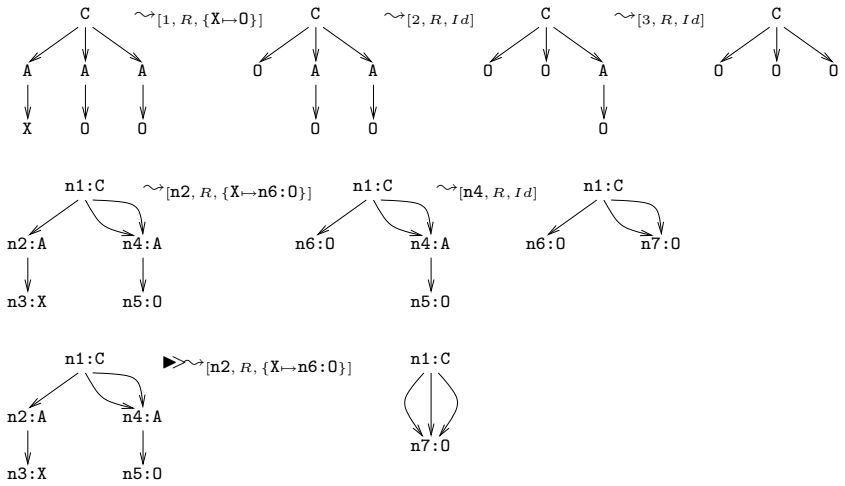

Figure2.

functional logic languages. We show that collapsing narrowing is complete. We also succeeded to generalize parallel term narrowing, and thus needed narrowing, to collapsing graph narrowing.

The rest of the paper is organized as follows. We briefly introduce the framework of weakly admissible graph rewriting systems in the next section. Section 3 establishes the soundness and completeness of collapsing narrowing in the setting of weakly admissible graph rewriting systems. In Section 4, we generalize optimal term narrowing strategies to the framework of graphs and list their main properties. Section 5 concludes the paper. Due to the lack of space, several technical definitions and all the proofs are omitted. They can be found in [8, 9].

2 Weakly Admissible Term Graph Rewriting Systems

The aim of this section is to precise the graphs and the graph rewriting systems we consider. Our notations are similar to those of [6, 16]. We are consistent with [7, 10].

Every node of a graph is labeled with an operation symbol or a variable. For practical reasons, we assume in this paper that operation symbols belong to many-sorted constructor-based signatures, i.e., triples $\Sigma = \langle S, \mathcal{C}, \mathcal{D} \rangle$ where S is a set of sorts, $\mathcal{C}$ is a set of sorted *constructor symbols* whose rôle consists in building data structures and $\mathcal{D}$ is a set of sorted *defined operations* such that $\mathcal{C} \cap \mathcal{D} = \emptyset$. Let $\mathcal{X}$ be a set of *variables*

and $\mathcal{N}$ a set of *nodes*. We assume, in the sequel, that Σ, $\mathcal{X}$ and $\mathcal{N}$ are given.

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 included in $\mathcal{N}$, $\mathcal{L}_g : \mathcal{N}_g \rightarrow \mathcal{C} \cup \mathcal{D} \cup \mathcal{X}$ 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 string (possibly empty, for nodes labeled by either variables or constants) of nodes and $\mathcal{R}\text{oots}_g$ is a nonempty set of distinguished nodes of g, called its *roots*. We also assume two conditions of well-definedness. (1) All nodes are accessible from at least one root, i.e., 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. (2) Every variable $x \in \mathcal{V}_g$, where $\mathcal{V}_g$ stands for the variables occurring in graph g, labels one and only one node of g. We call *functional node* a node labeled with a defined operation. A graph g is said to be a *constructor graph* if none of its nodes is a functional node. 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.

In this paper, we investigate graph narrowing for the class of what we call *admissible term graphs (atgs)* [7]. A *term graph* g is a (possibly cyclic) graph with one root denoted $\mathcal{R}\text{oot}_g$. We say that a term graph g is an *atg* iff there exists no path from a node labeled with a defined operation to itself in g. Hence, an atg corresponds, according to the imperative point of view, to nested procedure (function) calls whose parameters are complex constructor cyclic graphs (i.e., classical data structures).

Example 1. Figure 3 (at the end of the paper) shows several examples of graphs. For instance, the term graph H is given by (1) $\mathcal{N}_H = \{\mathtt{n1}, \ldots, \mathtt{n5}\}$, (2) $\mathcal{R}\text{oot}_H = \mathtt{n1}$, (3) $\mathcal{L}_H$ is defined by $\mathcal{L}_H(\mathtt{n1}) = \mathcal{L}_H(\mathtt{n4}) = \mathtt{c}$, $\mathcal{L}_H(\mathtt{n2}) = \mathcal{L}_H(\mathtt{n5}) = \mathtt{g}$ and $\mathcal{L}_H(\mathtt{n3}) = \mathtt{z}$ and (4) $\mathcal{S}_H$ is defined by $\mathcal{S}_H(\mathtt{n1}) = \mathtt{n2.n4}$, $\mathcal{S}_H(\mathtt{n2}) = \mathcal{S}_H(\mathtt{n5}) = \mathtt{n3.n3}$, $\mathcal{S}_H(\mathtt{n3}) = \varepsilon$ and $\mathcal{S}_H(\mathtt{n4}) = \mathtt{n5.n1}$. An equivalent description of H is `n1:c(n2:g(n3:z,n3),n4:c(n5:g(n3,n3),n1))`, following the syntax defined in [6]. In the following, we assume that `c` and `s` are constructor symbols, `g` is a defined operation and `x`, `y`, `z` and `u` are variables. As a consequence, all the term graphs of Fig. 3 are atgs. `p:g(p,p)` and `p:g(n:s(p),n)` are not atgs (since `g` is a defined operation which belongs to a cycle).

Graph homomorphisms are essential for atgs rewriting and narrowing since they allow to define matching and unification [7]. A *homomorphism* $h : g_1 \rightarrow g_2$ is a mapping from $\mathcal{N}_{g_1}$ to $\mathcal{N}_{g_2}$ such that $\mathcal{R}\text{oots}_{g_2} = h(\mathcal{R}\text{oots}_{g_1})$ 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 $\mathcal{S}_{g_2}(h(n)) = h(\mathcal{S}_{g_1}(n))$. We say that two atgs g_1 and g_2 are *unifiable* iff there exist two graphs G and H and a homomorphism $h : G \rightarrow H$ such that (1) g_1 and g_2 are both subgraphs of G and (2) $h(g_1) = h(g_2)$. 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* in the following sense : there exists a unifier $h : G \rightarrow H$ such that (1) $G = g_1 \oplus g_2$, (2) $h(g_1) = h(g_2) = H$ and (3) for all unifiers $h' : G' \rightarrow H'$, there exists a homomorphism $\phi : H \rightarrow H'$.

Example 2. Consider the subgraph $H_{|\mathtt{n2}}$ and the atg L of Fig. 3. Let υ be the mapping from $\mathcal{N}_{(H_{|\mathtt{n2}})} \cup \mathcal{N}_L$ to $\mathcal{N}_{(I_{|\mathtt{n2}})}$ such that $\upsilon(\mathtt{n2}) = \upsilon(\mathtt{l1}) = \mathtt{n2}$, $\upsilon(\mathtt{n3}) = \upsilon(\mathtt{l2}) = \upsilon(\mathtt{l4}) = \mathtt{m1}$ and $\upsilon(\mathtt{l3}) = \upsilon(\mathtt{l5}) = \mathtt{m2}$. υ is a homomorphism from the bi-rooted graph $H_{|\mathtt{n2}} \oplus L$ to the subgraph $I_{|\mathtt{n2}}$ such that $\upsilon(H_{|\mathtt{n2}}) = \upsilon(L)$. Therefore, $H_{|\mathtt{n2}}$ and L are unifiable. Moreover, all other unifiers of $H_{|\mathtt{n2}}$ and L must instantiate $I_{|\mathtt{n2}}$. Therefore, υ is a most general unifier of $H_{|\mathtt{n2}}$ and L.

The next definition introduces the notion of admissible rewrite rule [7]. Such rules are tailored so that the set of atgs is closed under rewriting and narrowing. An *admissible rewrite rule* is a bi-rooted graph of roots l and r, denoted $l \rightarrow r$, such that (1) the left-hand side, l, is a *pattern*, that is to say, an atg which has a tree structure (i.e., linear first-order term) with only one defined operation situated at its root, (2) the right-hand side, r, is an atg, (3) l is not a subgraph of r and (4) $\mathcal{V}_r \subseteq \mathcal{V}_l$. The graphs $L \rightarrow R$ and $L' \rightarrow R'$ of Fig. 3 are examples of admissible rewrite rules. We say that two admissible rules $l_1 \rightarrow r_1$ and $l_2 \rightarrow r_2$ *overlap* iff their left-hand sides are unifiable.

A *constructor-based graph rewriting system (cGRS)* is a pair $SP = \langle \Sigma, \mathcal{R} \rangle$ where Σ is a constructor-based signature and $\mathcal{R}$ is a set of admissible rules. We say that SP is a *weakly admissible graph rewriting system (WAGRS)* iff $\mathcal{R}$ is a set of admissible rules such that if two rules $l_1 \rightarrow r_1$ and $l_2 \rightarrow r_2$ overlap, then their instantiated right-hand sides are equal up to renaming of nodes.

Example 3. Consider the following cGRS :

```
(R1) l1:f(l2:a,l3:x)        -> r1:c(l3:x,r1)
(R2) l1:f(l2:x,l3:s(l4:y)) -> r1:c(l3:s(l4:y),r1)
(R3) l1:g(l2:a,l3:a,l4:x)  -> l4:x
```

```
(R4) l1:h(l2:a)             -> l2:a
(R5) l1:i(l2:a,l3:x,l4:y)   -> l2:a
(R6) l1:i(l2:x,l3:a,l4:y)   -> l2:x
(R7) l1:i(l2:x,l3:y,l4:a)   -> l2:x
```

The rules `R1` and `R2` (resp. `R5`, `R6` and `R7`) overlap and their instantiated right-hand sides are equal up to renaming of nodes. Therefore, this cGRS is a WAGRS.

Due to the lack of space, we do not give the definition of the graph rewriting relation $\rightarrow$ induced by the WAGRSs. It is the same as that of [6, 16, 7]. We have proved in [8] that $\rightarrow$ is confluent (and confluent modulo bisimilarity [5, 20]) w.r.t. atgs. We have also introduced in [8] a new *parallel* graph rewriting relation, $\rightarrow\!\!\!\!\!\!+$, which allows to reduce several arbitrary redexes of an atg in one shot and to boost the efficiency of rewriting. We have also defined in [8] the *parallel graph rewriting strategy* $\bar{\Phi}$: a partial function which takes an atg g and returns a set S of pairs (p, R) such that $g \rightarrow\!\!\!\!\!\!+_S g'$ for some graph g'. Our strategy, $\bar{\Phi}$, extends to WAGRSs the strategy designed in [1] for constructor-based weakly orthogonal first-order term rewriting systems. In Section 4, we will lift this rewriting strategy to narrowing.

3 Collapsing Narrowing

Collapsing graph narrowing have been investigated first in the setting of acyclic term graphs [17, 13]. This section introduces collapsing narrowing in the framework of atgs and WAGRSs [11].

Collapsing narrowing requires to fold atgs. We say that g_1 *collapses (or folds) into* g_2, denoted $g_1 \rhd g_2$, iff there exists a variable-preserving homomorphism $h : g_1 \rightarrow g_2$, that is to say, a homomorphism such that if $n \in \mathcal{N}_{g_1}$ is labeled with a variable $x \in \mathcal{V}_{g_1}$, then $h(n)$ is also labeled with x in g_2. An algorithm which collapses atgs is called a collapsing strategy. By *collapsing strategy*, we mean any total function $\blacktriangleright$ over the atgs such that if $\blacktriangleright(g) = g'$, then $g \rhd g'$. We assume that $\blacktriangleright$ is a deterministic function up to renaming of nodes. We shall use two noteworthy collapsing strategies in the following. The first one is the identity. It has no effect when it is applied on atgs. The second one, denoted $\blacktriangleright\!\!\!\!\blacktriangleright$, is the *maximally collapsing strategy* [13]: $\blacktriangleright\!\!\!\!\blacktriangleright(g_1) = g_2$ iff (1) $g_1 \rhd g_2$ and (2) for all atgs g, if $g_1 \rhd g$, then $g \rhd g_2$.

Example 4. Consider the atgs I and H' of Fig. 3. There exists a variable-preserving homomorphism $h : I \rightarrow H'$ such that $h(\texttt{n1}) = h(\texttt{n4}) = \texttt{p1}$,

$h(\texttt{n2}) = h(\texttt{n5}) = \texttt{p2}$, $h(\texttt{m1}) = \texttt{p3}$ and $h(\texttt{m2}) = \texttt{m2}$. So we deduce that $I \rhd H'$. Moreover, as H' cannot be collapsed any further, we conclude that $\blacktriangleright\!\!\gg(I) = H'$.

Seeking for a better readability, we deliberately use, in this paper, substitutions within narrowing steps instead of homomorphisms [21, 7]. A *substitution* σ is a partial function from the set of variables $\mathcal{X}$ to a set of term graphs. All the usual notions concerning term substitutions such as the *domain* $\mathcal{D}\sigma$, the *image* $\mathcal{I}\sigma$, the *identity* Id, the *restriction* $\sigma_{|V}$, the *application* $\sigma(g)$ or the *composition* $\sigma_2 \circ \sigma_1$ can be extended to graph substitutions (see [9] for details). For example, $\sigma(g)$ denotes the graph built from g by replacing all the variables $x \in \mathcal{D}\sigma$ by their images $\sigma(x)$. Hence, applying a substitution to a graph is roughly the same as applying a substitution to a first-order term, except that it preserves the sharing of subgraphs.

When narrowing an atg g using a rule $l \to r$, one needs to compute a unifier of a subgraph of g w.r.t. the left-hand side pattern l. This unifier can be represented by a substitution instead of a homomorphism. Such a substitution is computed as follows : Assume that g and l are unifiable, i.e., there exist two graphs G and H and a homomorphism $h : G \to H$ such that g and l are both subgraphs of G and $h(g) = h(l)$. Then $\mathcal{D}\sigma \subseteq \mathcal{V}_g$. Moreover, a variable x is in $\mathcal{D}\sigma$ iff it is assigned by h to a subgraph of H which is not reduced to a single node labeled with x. This subgraph is exactly $\sigma(x)$. We say that σ is a *most general unifier of g w.r.t. l* iff the unifier $h : G \to H$ is a most general unifier of g and l.

Example 5. In Example 2, we have shown that $\upsilon : (H_{|\texttt{n2}} \oplus L) \to I_{|\texttt{n2}}$ was a most general unifier of the atg $H_{|\texttt{n2}}$ and the pattern L (see Fig. 3). The only variable of $H_{|\texttt{n2}}$ which is assigned by υ is $\texttt{z}$. Therefore, $\sigma = \{\texttt{z} \mapsto \texttt{m1:s(m2:u)}\}$ is a most general unifier of $H_{|\texttt{n2}}$ w.r.t. L.

Now, we are ready to define collapsing narrowing :

Definition 1. *A* collapsing narrowing step from an atg g_1 to an atg g_2 using a functional node p, a rule $l \to r$, a substitution σ and a collapsing strategy $\blacktriangleright$*, is defined as follows :*

$$g_1 \blacktriangleright\!\!\rightsquigarrow_{[p, l\to r, \sigma]} g_2 \iff \begin{cases} \sigma \textit{ is a unifier of } g_{1|p} \textit{ w.r.t. } l \textit{ and} \\ \textit{there exist a graph } g_1' = \blacktriangleright(\sigma(g_1)) \textit{ and} \\ \textit{a homomorphism } h : \sigma(g_1) \to g_1' \textit{ such that} \\ g_1' \to_{[h(p), l\to r]} g_2 \end{cases}$$

We say that $g_1 \blacktriangleright\!\!\rightsquigarrow_{[p, l\to r, \sigma]} g_2$ *is a* most general collapsing narrowing step *iff* σ *is a most general unifier of* $g_{1|p}$ *w.r.t.* l*. The usual graph narrowing relation* $\rightsquigarrow$ *[7] is obtained when the collapsing strategy is the identity.* $\blacktriangleright\!\!\!\rightsquigarrow$ *denotes the* maximally collapsing narrowing relation.

Example 6. In Example 2, we have shown that $\sigma = \{\mathtt{z} \mapsto \mathtt{m1{:}s(m2{:}u)}\}$ was a most general unifier of $H_{|\mathtt{n2}}$ w.r.t. L. Moreover, by Example 4, $\blacktriangleright\!\!(\sigma(H)) = H'$ and $h(\mathtt{n2}) = \mathtt{p2}$. We can show that $H' \to_{[\mathtt{p2}, L\to R]} H_1$ (see Fig. 3). So we conclude that $H \blacktriangleright\!\!\!\rightsquigarrow_{[\mathtt{n2}, L\to R, \sigma]} H_1$. A second example of a collapsing narrowing step is presented in Fig. 3, namely $H_1 \blacktriangleright\!\!\!\rightsquigarrow_{[\mathtt{q2}, L'\to R', \sigma']} H_2$ where $\sigma' = \{\mathtt{u} \mapsto \mathtt{m3{:}a}\}$.

Narrowing is used to solve goals. A solution of a goal is often represented by a substitution. We say that a substitution σ *is computed by a narrowing derivation from an atg* g_1 *to an atg* g_2 and write $g_1 \overset{*}{\blacktriangleright\!\rightsquigarrow}_\sigma g_2$ iff there exists a derivation $g_1 \blacktriangleright\!\rightsquigarrow_{[p_1, R_1, \sigma_1]} \cdots \blacktriangleright\!\rightsquigarrow_{[p_k, R_k, \sigma_k]} g_2$ and $\sigma = (\sigma_k \circ \ldots \circ \sigma_1)_{|\mathcal{V}_{g_1}}$.

Example 7. In Fig. 3, the derivation
$H \blacktriangleright\!\!\!\rightsquigarrow_{[\mathtt{n2}, L\to R, \sigma]} H_1 \blacktriangleright\!\!\!\rightsquigarrow_{[\mathtt{q2}, L'\to R', \sigma']} H_2$ allows to compute the substitution $\theta = (\sigma' \circ \sigma)_{|\mathcal{V}_H} = \{\mathtt{z} \mapsto \mathtt{m1{:}s(m3{:}a)}\}$.

Below, we recall the notions of soundness and completeness of narrowing [7]. 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 (atgs). The traditional goals such as equations can be represented as boolean expressions.

These definitions use two preorders $\dot{\leq}$ defined on atgs and substitutions. Two atgs g_1 and g_2 are *bisimilar*, denoted $g_1 \doteq g_2$, iff they represent the same (infinite) tree when one unravels them [4]. We write $g_1 \dot{\leq} g_2$ iff there exists a substitution θ such that $\theta(g_1) \doteq g_2$. We extend $\doteq$ and $\dot{\leq}$ to substitutions : two substitutions σ_1 and σ_2 are *bisimilar* on a set V of variables, denoted $\sigma_1 \doteq \sigma_2$ $[V]$, iff $\sigma_1(x) \doteq \sigma_2(x)$ for all $x \in V$. We write $\sigma_1 \dot{\leq} \sigma_2$ $[V]$ iff there exists a substitution θ such that $\theta \circ \sigma_1 \doteq \sigma_2$ $[V]$.

Definition 2. *We say that the narrowing relation* $\blacktriangleright\!\rightsquigarrow$ *is* sound *iff for all atgs* g*, constructor graphs* c *and substitutions* θ *such that* $g \overset{*}{\blacktriangleright\!\rightsquigarrow}_\theta c$*, there exists a constructor graph* s *such that* $\theta(g) \overset{*}{\to} s$ *and* $s \doteq c$*.*
We say that the narrowing relation $\blacktriangleright\!\rightsquigarrow$ *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* s *and a substitution* σ *such that* $g \overset{*}{\blacktriangleright\!\rightsquigarrow}_\sigma s$*,* $s \dot{\leq} c$ *and* $\sigma \dot{\leq} \theta$ $[\mathcal{V}_g]$.

Theorem 1. *$\blacktriangleright\!\!\rightsquigarrow$, thus $\rightsquigarrow$ and $\blacktriangleright\!\!\gg\!\!\rightsquigarrow$, are sound and complete over the WAGRSs.*

Notice that we restricted Theorem 1 to the WAGRSs. Indeed, the induced rewriting relation over atgs is always confluent [8]. Actually, we proved in [7] that $\rightsquigarrow$ was sound and complete for general cGRSs, but this is not the case for $\blacktriangleright\!\!\gg\!\!\rightsquigarrow$ (and hence for $\blacktriangleright\!\!\rightsquigarrow$), as shown by the following counter-example :

Example 8. The three following rules constitute a non confluent cGRS : `(R1) A(B(X))` $\rightarrow$ `C(D,D)`, `(R2) D` $\rightarrow$ `E` and `(R3) D` $\rightarrow$ `F`. We assume that `A` and `D` are defined operations and `B`, `C`, `E`, `F` and `O` are constructor symbols. Let $s =$ `n1:A(n2:Y)` and $t = \theta(s) =$ `n1:A(n3:B(n4:O))` where $\theta = \{$`Y` $\mapsto$ `n3:B(n4:O)`$\}$. It is easy to check that $t \stackrel{*}{\rightarrow} t'$ with $t' =$ `m1:C(m4:E,m5:F)`. Consider the maximally collapsing narrowing derivations starting with s. There is only one way to begin them : $s \blacktriangleright\!\!\gg\!\!\rightsquigarrow_{[\texttt{n1},\texttt{R1},\sigma]} s'$ with $\sigma = \{$`Y` $\mapsto$ `q1:B(q2:Z)`$\}$ and $s' =$ `p1:C(p2:D,p2':D)`. s' maximally collapses into $s'' =$`p1:C(p2:D,p2)`. Then, s'' narrows into either $s_1 =$ `p1:C(p3:E,p3)` or $s_2 =$ `p1:C(p4:F,p4)` with the identity substitution. s_1 and s_2 are constructor atgs. It is clear that $s_1 \not\dot{\leq} t'$ and $s_2 \not\dot{\leq} t'$. So we conclude that $\blacktriangleright\!\!\gg\!\!\rightsquigarrow$, thus $\blacktriangleright\!\!\rightsquigarrow$, are not complete in the framework of general cGRSs.

On the other hand, the maximally collapsing rewriting relation $\blacktriangleright\!\!\gg\!\!\rightarrow$ was shown to be not confluent in general (see [5]). However, from Theorem 1, it follows that $\blacktriangleright\!\!\gg\!\!\rightarrow$ is confluent on the class of graphs which have a constructor normal form (i.e., $\mathcal{C}$-normalizable atgs).

Corollary 1. *Let SP be a WAGRS. Then, $\blacktriangleright\!\!\gg\!\!\rightarrow$ is confluent over $\mathcal{C}$-normalizable atgs. I.e., constructor normal forms are unique when they exist.*

4 Collapsing Narrowing Strategies

Needed narrowing [2] and Parallel narrowing [3] are the best known narrowing strategies for constructor-based weakly orthogonal term rewriting systems and inductively sequential term rewriting systems respectively. However, basic narrowing is the only known strategy for acyclic term graph collapsing narrowing [17, 13]. In this section, we show that parallel narrowing as well as needed narrowing can be extended to collapsing narrowing in the setting of atgs. The resulting strategies, inherit all the nice

properties of the corresponding ones over terms but in addition develop shorter derivations thanks to sharing of subexpressions.

Our first strategy, $\bar{\Lambda}$, extends weakly needed narrowing [3] to atgs. $\bar{\Lambda}$ is a *sequential strategy*, i.e., $\bar{\Lambda}$ is a partial function which takes an atg g and returns a set of tuples of the form (p, R, σ) such that $g \blacktriangleright\!\!\rightsquigarrow_{[p,R,\sigma]} g'$ for some atg g'. We write $g \blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}} g'$ the $\bar{\Lambda}$-*step* from g to g'. $\bar{\Lambda}$ is based on the organization of WAGRSs as forests of definitional trees. A definitional tree [1] is a hierarchical structure whose leaves are the rules of a WAGRS used to define some operation. In the following definition, *branch* and *rule* are uninterpreted symbols, used to construct the nodes of a definitional tree.

Definition 3. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. A tree $\mathcal{T}$ is a* partial definitional tree*, or* pdt*, with pattern π iff one of the following cases holds :*

- $\mathcal{T} = rule(\pi \to r)$*, where $\pi \to r$ 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 ζ, $c_1, \ldots, c_k$ ($k > 0$) are different constructors of the sort ζ 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 $\mathcal{T}$. 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 and $p, o_1, \ldots, o_n$ are new nodes. A* forest of definitional trees (fdt) *$\mathcal{F}$ of an operation f is a set of definitional trees such that every rule defining f appears in one and only one tree in $\mathcal{F}$.*

Example 9. Consider the WAGRS of Example 3. A definitional tree $\mathcal{T}_{\mathtt{g}}$ of the operation `g` is given by :

$$\begin{aligned}\mathcal{T}_{\mathtt{g}} = branch(&\mathtt{k12{:}g(\underline{k13{:}x6},k14{:}x7,k15{:}x8)},\ \mathtt{k13},\\ &branch(\mathtt{k12{:}g(k16{:}a,\underline{k14{:}x7},k15{:}x8)},\ \mathtt{k14},\\ &\quad rule(\mathtt{k12{:}g(k16{:}a,k17{:}a,k15{:}x8)} \to \mathtt{k15{:}x8})))\end{aligned}$$

$\mathcal{T}_{\mathtt{g}}$ is represented in Fig. 4. Notice that the rules `R1` and `R2` (or `R5`, `R6` and `R7`) of Example 3 cannot be represented in a single definitional tree because they overlap. This is the reason why we introduced the notion of fdts. In Fig. 4, we represent possible fdts $\mathcal{F}_{\mathtt{f}} = \{\mathcal{T}^1_{\mathtt{f}}, \mathcal{T}^2_{\mathtt{f}}\}$, $\mathcal{F}_{\mathtt{g}} = \{\mathcal{T}_{\mathtt{g}}\}$, $\mathcal{F}_{\mathtt{h}} = \{\mathcal{T}_{\mathtt{h}}\}$ and $\mathcal{F}_{\mathtt{i}} = \{\mathcal{T}^1_{\mathtt{i}}, \mathcal{T}^2_{\mathtt{i}}, \mathcal{T}^3_{\mathtt{i}}\}$ corresponding to the operations `f`, `g`, `h` and `i` defined in Example 3. The reader familiar with the formalism used in [3] will notice that the use of forests departs from the definitional

trees with "OR-nodes". The advantage of using forest structures stems from the fact that every rewrite rule is represented only once. This is unfortunately not the case when using "OR-nodes".

The sequential graph narrowing strategy $\bar{\Lambda}$ is a partial function that operates on atgs in the presence of WAGRSs. $\bar{\Lambda}(g)$ returns, when it is possible, a set of tuples $(n, l \to r, \sigma)$ such that g is narrowable at node n using the rule $l \to r$ and the substitution σ. σ is a particular unifier of $g_{|n}$ w.r.t. l, which is generally *not* a most general unifier of $g_{|n}$ w.r.t. l. Actually, σ may assign some variables of g which are not variables of $g_{|n}$. $\bar{\Lambda}$ uses an auxiliary function $\bar{\lambda}$ which takes two arguments : an operation-rooted atg and a pdt of this operation.

Definition 4. *$\bar{\Lambda}$ is the partial function such that $\bar{\Lambda}(g) = \bar{\lambda}(g_{|p}, \mathcal{T}_1) \cup \ldots \cup \bar{\lambda}(g_{|p}, \mathcal{T}_n)$ where p is the leftmost-outermost functional node of g and $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ is a forest of definitional trees of the label of p in g.*
Let g be an operation-rooted atg and $\mathcal{T}$ a pdt such that $pattern(\mathcal{T})$ unifies with g at the root. $\bar{\lambda}(g, \mathcal{T})$ is a set of triples of the form (p, R, σ), where p is a non variable node of g, R is a rewrite rule and σ is a unifier of $g_{|p}$ w.r.t the left-hand side of R. $\bar{\lambda}(g, \mathcal{T})$ is defined as the smallest set such that :

$$\bar{\lambda}(g, \mathcal{T}) \supseteq \begin{cases} \{(p, R, \sigma)\} & \text{if } \mathcal{T} = rule(\pi \to r),\ p = \mathcal{R}oot_g,\ R = \pi \to r \text{ and} \\ & \quad \sigma \text{ is an mgu of } g \text{ w.r.t. } \pi \ ; \\ \bar{\lambda}(g, \mathcal{T}_i) & \text{if } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ and} \\ & \quad g \text{ and } pattern(\mathcal{T}_i) \text{ unify for some } i \in 1..k \ ; \\ \{(p, R, \sigma)\} & \text{if } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad \tau \text{ is a mgu of } g \text{ w.r.t. } \pi \text{ and} \\ & \quad \text{there exists a homomorphism } h : \pi \to \tau(g), \\ & \quad h(o) \text{ is labeled with a defined operation } f, \\ & \quad \mathcal{F} = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_k\} \text{ is an fdt of } f, \\ & \quad S = \bar{\lambda}(\tau(g_{|h(o)}), \mathcal{T}'_1) \cup \ldots \cup \bar{\lambda}(\tau(g_{|h(o)}), \mathcal{T}'_k), \\ & \quad (p, R, \sigma') \in S \text{ and } \sigma = \sigma' \circ \tau. \end{cases}$$

Example 10. Consider the atg g of Fig. 4. By Definition 4,

$$\begin{aligned} \bar{\Lambda}(g) &= \bar{\lambda}(g_{|\mathtt{n1}}, \mathcal{T}^1_{\mathtt{f}}) \cup \bar{\lambda}(g_{|\mathtt{n1}}, \mathcal{T}^2_{\mathtt{f}}) \\ &= \bar{\lambda}(g_{|\mathtt{n2}}, \mathcal{T}_{\mathtt{g}}) \cup \bar{\lambda}(g_{|\mathtt{n6}}, \mathcal{T}^1_{\mathtt{i}}) \cup \bar{\lambda}(g_{|\mathtt{n6}}, \mathcal{T}^2_{\mathtt{i}}) \cup \bar{\lambda}(g_{|\mathtt{n6}}, \mathcal{T}^3_{\mathtt{i}}) \\ &= \{(\mathtt{n2}, \mathtt{R3}, \sigma_1)\} \cup \bar{\lambda}(g_{|\mathtt{n5}}, \mathcal{T}_{\mathtt{h}}) \cup \bar{\lambda}(g_{|\mathtt{n7}}, \mathcal{T}_{\mathtt{h}}) \cup \{(\mathtt{n6}, \mathtt{R7}, Id)\} \\ &= \{(\mathtt{n2}, \mathtt{R3}, \sigma_1), (\mathtt{n5}, \mathtt{R4}, \sigma_2), (\mathtt{n7}, \mathtt{R4}, \sigma_2), (\mathtt{n6}, \mathtt{R7}, Id)\} \end{aligned}$$

where the substitutions σ_1 and σ_2 are defined by
$\sigma_1 = \{\mathtt{x} \mapsto \mathtt{r1} : \mathtt{a}, \mathtt{y} \mapsto \mathtt{r2} : \mathtt{a}\}$ and $\sigma_2 = \{\mathtt{y} \mapsto \mathtt{r3} : \mathtt{a}\}$.

Theorem 2. $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$ *is sound and complete over WAGRSs.*

Each step of $\blacktriangleright\!\!\rightsquigarrow_{\bar{\Lambda}}$ uses a single rewriting step. We can improve sequential narrowing by using a *parallel* rewriting step instead :

Definition 5. *A* parallel collapsing narrowing step from an atg g_1 to an atg g_2 using the functional nodes $p_1, \ldots, p_n$, the rules $R_1, \ldots, R_n$, the substitution σ and the collapsing strategy $\blacktriangleright$, *is defined as follows :*

$$g_1 \blacktriangleright\!\!\Downarrow_{[p_1,R_1]\ldots[p_n,R_n],\sigma} g_2 \iff \begin{cases} \sigma(g_1) \blacktriangleright g_1' \text{ with } h : \sigma(g_1) \to g_1' \text{ and} \\ g_1' \dashrightarrow_{[h(p_1),R_1]\ldots[h(p_n),R_n]} g_2 \end{cases}$$

The definition of a parallel collapsing narrowing step needs the computation of substitutions as well as the computation of the different positions used to rewrite in parallel. In this paper, we consider that parallel rewriting is performed with the rewriting strategy $\bar{\Phi}$ defined in [8]. As for the computation of substitutions, it is performed by the parallel narrowing strategy $\bar{\bar{\Lambda}}$ which is defined below :

Definition 6. $\bar{\bar{\Lambda}}$ *is the partial function such that :*

$$\bar{\bar{\Lambda}}(g) = \left\{ \sigma_{|\mathcal{V}_g} \text{ such that } \left| \begin{array}{l} \exists (p,R,\sigma) \in \bar{\Lambda}(g), \\ (\forall (q,S,\theta) \in \bar{\Lambda}(g), \\ \text{if } \theta \dot{\leq} \sigma \ [\mathcal{V}_g] \text{ and } \theta \neq Id \ [\mathcal{V}_g], \\ \text{then } \sigma \doteq \theta \ [\mathcal{V}_g]) \\ \text{and} \\ (\exists C \in \mathcal{P}aths_g(\mathcal{R}oot_g, p), \\ \forall (q,S,\theta) \in \bar{\Lambda}(g), \\ \text{if } \theta \dot{\leq} \sigma \ [\mathcal{V}_g] \text{ and } q \in C, \\ \text{then } \sigma \doteq \theta \ [\mathcal{V}_g]) \end{array} \right. \right\} / \doteq$$

In the definition of $\bar{\bar{\Lambda}}(g)$, the first condition selects the least instantiated substitutions among those of $\bar{\Lambda}(g)$ which are not the identity, in addition to the identity substitution if there exists some triple (p, R, Id) in $\bar{\Lambda}(g)$. The second condition allows to eliminate substitutions which are below the identity in every path of the graph. Notice that this condition departs from the one given in [3] due to the sharing of subgraphs. Lastly, the set $\bar{\bar{\Lambda}}(g)$ is defined up to bisimilarity : if A denotes a set of substitutions, $A/\doteq$ denotes the "quotient" set of A by $\doteq$ which consists of substitution representatives of A up to renaming and bisimilarity.

The parallel narrowing step $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ induced by $\bar{\bar{\Lambda}}$ and $\bar{\Phi}$ is defined by

$$g_1 \blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi},\sigma} g_2 \iff \sigma \in \bar{\bar{\Lambda}}(g_1), \blacktriangleright(\sigma(g_1)) = g_1' \text{ and } g_1' \dashrightarrow_{\bar{\Phi}(g_1')} g_2$$

Example 11. Following Example 10, we now explain the computation of $\bar{\Lambda}(g)$. In Fig. 4, we have sketched in the graph Π the relative positions of the triples of $\bar{\Lambda}(g)$ in g. This "graph" has no formal semantics but makes easier the understanding of $\bar{\Lambda}(g)$. Substitutions in $\bar{\Lambda}(g)$ are those of $\bar{\Lambda}(g)$ which satify the conditions in Definition 6. σ_1 must be eliminated because $\sigma_2 \dot{\leq} \sigma_1\ [\mathcal{V}_g]$. σ_2 from triple $(\mathtt{n7}, \mathtt{R4}, \sigma_2)$ is discarded because the only path from `n1` to `n7` (i.e., $[\mathtt{n1}, 2, \mathtt{n6}, 2, \mathtt{n7}]$) contains the node `n6` and $(\mathtt{n6}, \mathtt{R7}, Id) \in \bar{\Lambda}(g)$ and $Id \dot{\leq} \sigma_2\ [\mathcal{V}_g]$. σ_2 from triple $(\mathtt{n5}, \mathtt{R4}, \sigma_2)$ is kept since there exists a path $C = [\mathtt{n1}, 1, \mathtt{n2}, 3, \mathtt{n5}]$ such that for all $(q, R, \theta) \in \bar{\Lambda}(g)$, if $p \in C$ (e.g., `n2`), then $\sigma_2 \dot{\leq} \theta\ [\mathcal{V}_g]$. Id from triple $(\mathtt{n6}, \mathtt{R7}, Id)$ is kept for the same reason. Hence, we conclude that $\bar{\bar{\Lambda}}(g) = \{Id, \sigma_2\}$. By the the definition of a parallel rewriting step [8] we get $g \blacktriangleright\!\!\gg\!\!\Downarrow_{\bar{\bar{\Lambda}},\sigma_2} g'$ with $g' = \mathtt{p1{:}f(p2{:}g(n3{:}x,p3{:}a,p3),p3)}$.

Theorem 3. $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ *is sound and complete over WAGRSs.*

5 Conclusion

We have investigated collapsing graph narrowing in the framework of weakly admissible graph rewriting systems. We have first established its completeness over WAGRSs. We have also defined parallel collapsing graph narrowing strategy which is a conservative extension of the best known term narrowing strategies [2, 3].

The parallel collapsing narrowing relation $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ inherits all optimality properties of parallel term narrowing [3]. Indeed, $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ computes only needed graph narrowing derivations [7] in the case of inductively sequential graph rewriting systems, i.e., cGRSs such that the rewrite rules of each defined operation may be stored within one definitional tree. Moreover, $\blacktriangleright\!\!\Downarrow_{\bar{\bar{\Lambda}},\bar{\Phi}}$ normalizes deterministically ground graphs to their constructor normal form when it exists.

In addition to the above properties, graph structures induce new improvements for narrowing. Actually, thanks to the sharing of subexpression in graph structures, needed and parallel graph collapsing narrowing develop shorter derivations than the corresponding term narrowing strategies. Furthermore, the implementation of $\bar{\lambda}$ is more efficient than its corresponding one for terms because $\bar{\lambda}$ can avoid redundant computations which occur when $\bar{\lambda}$ has to revisit several times a same shared subgraph. This kind of improvements are not possible for tree (term) structures. Thus needed and parallel collapsing graph narrowing appear to be appropriate for good implementation of modern declarative languages such as Curry [14].

References

[1] S. Antoy. Definitional trees. In *Proc. of the 3rd International Conference on Algebraic and Logic Programming (ALP'92)*, pages 143–157. LNCS 632, September 1992.

[2] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. In *Proc. of the 21st ACM Symposium on Principles of Programming Languages (POPL'94)*, pages 268–279, Portland, 1994.

[3] S. Antoy, R. Echahed, and M. Hanus. Parallel evaluation strategies for functional logic languages. In *Proc. of the International Conference on Logic Programming (ICLP'97)*, pages 138–152, Portland, 1997. MIT Press.

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

[5] Z. M. Ariola, J. W. Klop, and D. Plump. Confluent rewriting of bisimilar term graphs. *Electronic Notes in Theoretical Computer Science*, 7, 1997.

[6] 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, 1987.

[7] R. Echahed and J. C. Janodet. Admissible graph rewriting and narrowing. In *Proc. of Joint International Conference and Symposium on Logic Programming (JICSLP'98)*, pages 325–340. MIT Press, June 1998.

[8] R. Echahed and J. C. Janodet. On weakly orthogonal constructor-based graph rewriting. Internal report, IMAG, 1998. Long version of [10]. Available at URL : `ftp://ftp.imag.fr/pub/labo-LEIBNIZ/PMP/wa-c-graph-rewriting.ps.gz`.

[9] R. Echahed and J. C. Janodet. Collapsing graph narrowing. Internal report, IMAG, May 1999. Available at URL : `ftp://ftp.imag.fr/pub/LEIBNIZ/PMP/wa-collapsing-narrowing.ps.gz`.

[10] R. Echahed and J. C. Janodet. Parallel admissible graph rewriting. In *Recent Trends in Algebraic Development Techniques*, pages 121–135. LNCS 1589, 1999.

[11] R. Echahed and J. C. Janodet. Completeness of admissible graph collapsing narrowing. In *Proc. of Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (GRATRA 2000)*, page to appear, March 2000.

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

[13] A. Habel and D. Plump. Complete strategies for term graph narrowing. In *Recent Trends in Algebraic Development Techniques*. LNCS 1589, 1999.

[14] M. Hanus, S. Antoy, H. Kuchen, F. J. López-Fraguas, J. J. Moreno-Navarro, and F. Steiner. Curry : An integrated functional logic language. Available at `http://www-i2.informatik.rwth-aachen.de/~hanus/curry/report.html`, January 13 1999. Version 0.5.

[15] J. M. Hullot. Canonical forms and unification. In *Proc. of the 5th Conference on Automated Deduction (CADE'87)*, pages 318–334. LNCS 87, 1980.

[16] 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 Transactions on Programming Languages and Systems*, 16(3):493–523, 1994.

[17] M. R. K. Krishna Rao. Completeness results for basic narrowing in non-copying implementations. In *Proc. of the Joint International Conference and Symposium on Logic Programming (JICSLP'96)*, pages 393–407. MIT press, 1996.

[18] A. Middeldorp and E. Hamoen. Counterexamples to completeness results for basic narrowing. In *Proc. of the 3rd International Conference on Algebraic and Logic Programming (ALP'92)*, pages 244–258. LNCS 632, 1992.

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

[20] D. Plump. Term graph rewriting. In H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2. World Scientific, 1998.

[21] H. Yamanaka. Graph narrowing and its simulation by graph reduction. Research report IIAS-RR-93-10E, Institute for Social Information Science, Fujitsu Laboratories LDT, June 1993.

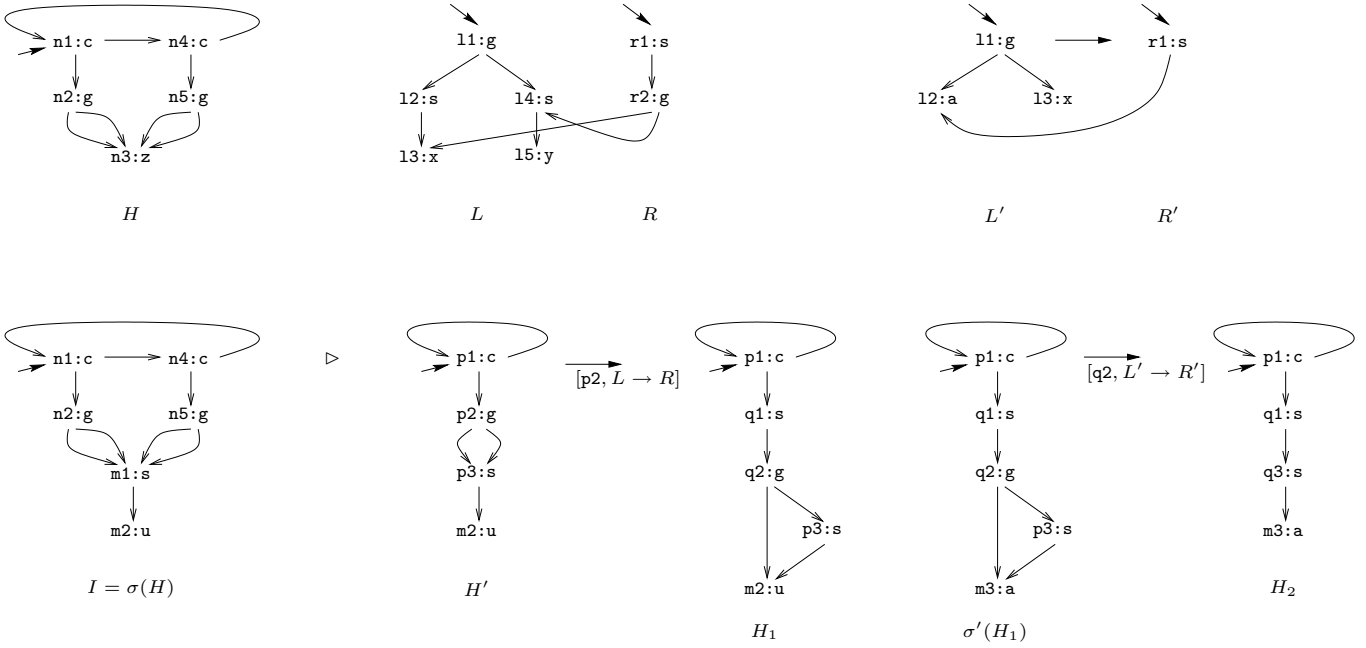
n1:c
n4:c
n2:g
n5:g
n3:z
H
l1:g
r1:s
l2:s
l4:s
r2:g
l3:x
l5:y
L
R
l1:g
r1:s
l2:a
l3:x
L′
R′
n1:c
n4:c
n2:g
n5:g
m1:s
m2:u
I = σ(H)
▷
p1:c
p2:g
p3:s
m2:u
H′
[p2, L → R]
p1:c
q1:s
q2:g
p3:s
m2:u
H1
p1:c
q1:s
q2:g
p3:s
m3:a
σ′(H1)
[q2, L′ → R′]
p1:c
q1:s
q3:s
m3:a
H2

Figure3.

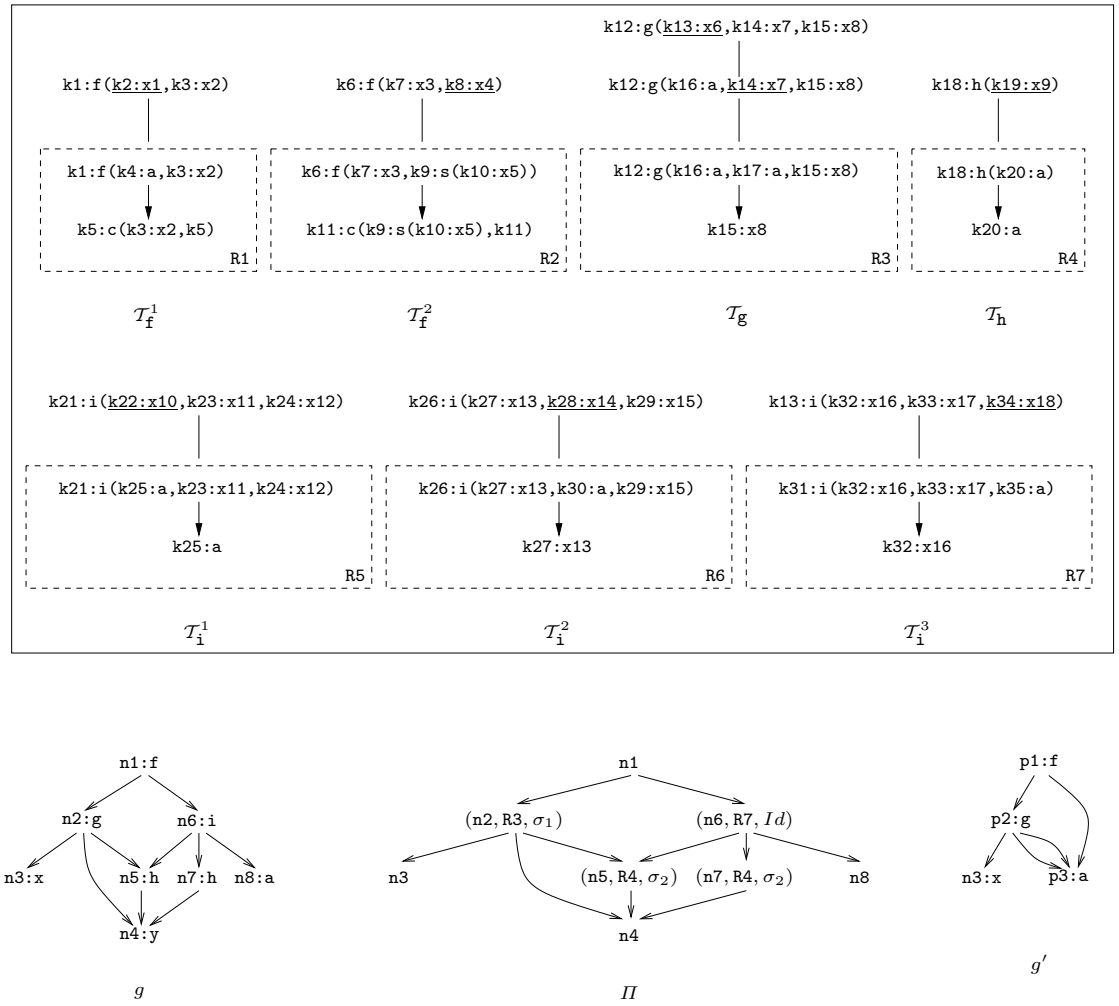
k12:g(k13:x6,k14:x7,k15:x8)
k1:f(k2:x1,k3:x2)
k6:f(k7:x3,k8:x4)
k12:g(k16:a,k14:x7,k15:x8)
k18:h(k19:x9)
k1:f(k4:a,k3:x2)
k5:c(k3:x2,k5)
R1
k6:f(k7:x3,k9:s(k10:x5))
k11:c(k9:s(k10:x5),k11)
R2
k12:g(k16:a,k17:a,k15:x8)
k15:x8
R3
k18:h(k20:a)
k20:a
R4
$\mathcal{T}_{\mathtt{f}}^1$
$\mathcal{T}_{\mathtt{f}}^2$
$\mathcal{T}_{\mathtt{g}}$
$\mathcal{T}_{\mathtt{h}}$
k21:i(k22:x10,k23:x11,k24:x12)
k26:i(k27:x13,k28:x14,k29:x15)
k13:i(k32:x16,k33:x17,k34:x18)
k21:i(k25:a,k23:x11,k24:x12)
k25:a
R5
k26:i(k27:x13,k30:a,k29:x15)
k27:x13
R6
k31:i(k32:x16,k33:x17,k35:a)
k32:x16
R7
$\mathcal{T}_{\mathtt{i}}^1$
$\mathcal{T}_{\mathtt{i}}^2$
$\mathcal{T}_{\mathtt{i}}^3$
n1:f
n2:g
n6:i
n3:x
n5:h
n7:h
n8:a
n4:y
g
n1
$(\mathtt{n2}, \mathtt{R3}, \sigma_1)$
$(\mathtt{n6}, \mathtt{R7}, Id)$
n3
$(\mathtt{n5}, \mathtt{R4}, \sigma_2)$
$(\mathtt{n7}, \mathtt{R4}, \sigma_2)$
n8
n4
Π
p1:f
p2:g
n3:x
p3:a
g'

Figure4.