Parallel Graph Narrowing

Rachid Echahed and Jean-Christophe Janodet
46, avenue Felix Viallet
38031 Grenoble - France
Firstname.Lastname@imag.fr

Abstract

We investigate graph narrowing as the operational semantics of functional logic programming languages. We mainly show and discuss how weakly needed narrowing as well as parallel narrowing may be extended to graph structures.

1 Introduction

Functional logic programming languages integrate, in a uniform way, functional languages and logic languages. The resulting languages (e.g., [12, 13]) have the advantages of both paradigms. Their operational semantics is often based on *first-order term* narrowing (see [11]).

However, in practice, data structures are not always represented as first-order terms but rather as cyclic graphs. Hence, several declarative languages such as Haskell, Clean or Life allow to work with graphs explicitely.

There are many reasons which motivate the use of graphs. They allow to go beyond the processing of first-order terms by handling efficiently real-wold data types represented as complex cyclic graphs. They also permit the sharing of subexpressions which leads to efficient computations. Consider, for instance, the rule $R =$ `0+Z` $\rightarrow$ `Z`. In Fig. 1, the length of the first (term) narrowing derivation is $2^p - 1$ whereas the length of the second (graph) narrowing derivation is p.

In practice, many programming languages are constructor-based, i.e., operators called constructors which are used to build the data structures are distinguished from operators called defined functions which are defined by means of rewrite rules. In this paper, we follow this discipline and study the narrowing relation induced by the so-called weakly admissible graph rewriting systems (WAGRSs) [8].

Actually, using graph rewriting systems instead of term rewriting systems is not an easy task. The classical properties of term rewriting systems such as confluence or the completeness of the narrowing relation cannot be lifted without caution to graph rewriting systems (see [6, 10]). Therefore, WAGRSs have been tailored so that they preserve the wanted properties.

Moreover, WAGRSs extend the constructor-based weakly orthogonal term rewriting systems [3]. In this setting, efficient narrowing strategies have been proposed such as (weakly) needed narrowing [2, 3] and parallel narrowing [3]. In this paper, we show that (weakly) needed narrowing and parallel narrowing can be extended to the framework of WAGRSs.

In the following section, we give briefly some preliminaries. Section 3 defines the WAGRSs and the sequential and parallel rewrite relations they induce. Most general narrowing and weakly needed narrowing are defined in Section 4 where we

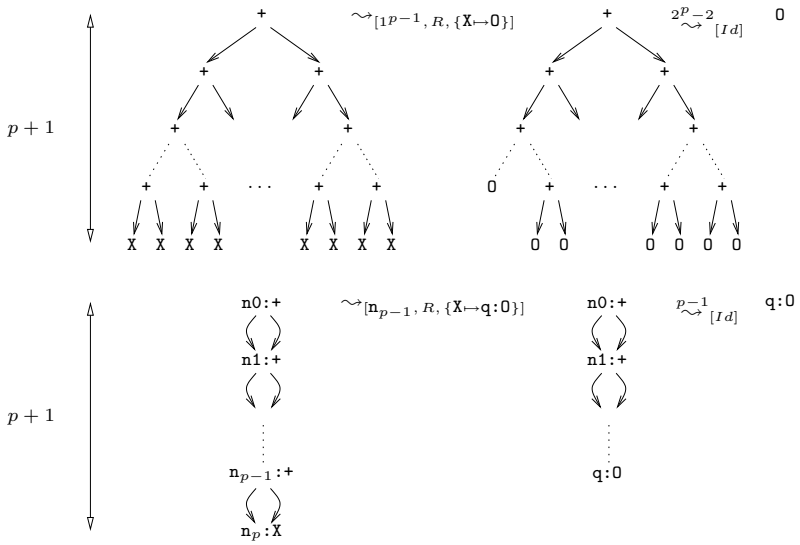

Figure 1:

establish their completeness. Section 5 is devoted to parallel graph narrowing. We conclude the paper in Section 6.

2 Definitions and Notations

Many different notations are used in the literature to investigate graph rewriting [9, 15, 16]. The aim of this section is to give briefly some key definitions in order to make easier the understanding of the paper. We are mostly consistent with [5].

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 $\Sigma = \langle S, \Omega \rangle$ be a *many-sorted signature*, $\mathcal{X}$ a set of *variables* and $\mathcal{N}$ a set of *nodes*. 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. (1) 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 $\mathcal{L}_g(n)$. (2) 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. (3) 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 [4]. We write $g_1 \sim g_2$ when 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. $\mathcal{N}$) :

GRAPH ::= NODE | NODE + GRAPH
NODE ::= n:A(NODE,...,NODE) | n

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 In Fig. 2, we give two examples of graphs denoted G and T. The term graph G is given by (1) $\mathcal{N}_G = \{\mathtt{n1}, \ldots, \mathtt{n5}\}$, (2) $\mathcal{R}\mathrm{oot}_G = \mathtt{n1}$, (3) $\mathcal{L}_G$ is defined by $\mathcal{L}_G(\mathtt{n1}) = \mathcal{L}_G(\mathtt{n5}) = \mathtt{c}$, $\mathcal{L}_G(\mathtt{n2}) = \mathtt{g}$, $\mathcal{L}_G(\mathtt{n3}) = \mathtt{s}$ and $\mathcal{L}_G(\mathtt{n4}) = \mathtt{a}$ and (4) $\mathcal{S}_G$ is defined by $\mathcal{S}_G(\mathtt{n1}) = \mathtt{n2.n5}$, $\mathcal{S}_G(\mathtt{n2}) = \mathtt{n3.n3}$, $\mathcal{S}_G(\mathtt{n3}) = \mathtt{n4}$, $\mathcal{S}_G(\mathtt{n4}) = \varepsilon$ and $\mathcal{S}_G(\mathtt{n5}) = \mathtt{n3.n1}$, thus $G = \mathtt{n1:c(n2:g(n3:s(n4:a),n3),n5:c(n3,n1))}$. On the other hand, T is a graph with two roots $\{\mathtt{l1,r1}\}$ representing a rewrite rule (see Def. 3.1) : $T = \mathtt{l1:g(l2:s(l3:u),l4:s(l5:v))} + \mathtt{r1:s(r2:g(l3,l4))}$.

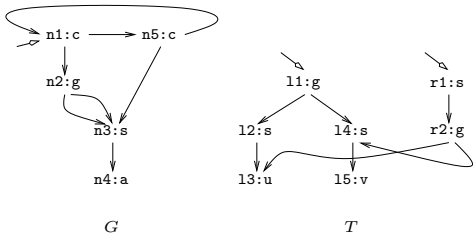

Figure 2:

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 (e.g., $G_{|\mathtt{n2}} = \mathtt{n2:g(n3:s(n4:a),n3)}$ in Fig. 2). 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.

A *multiple pointer redirection* ρ from the nodes $p_1, \ldots, p_n$ to the nodes $q_1, \ldots, q_n$ is a function $\rho : \mathcal{N} \to \mathcal{N}$ such that $\rho(p_i) = q_i$ for all $i \in 1..n$ and $\rho(p) = p$ for all nodes p such that $p \neq p_1$, $p \neq p_2$, ...and $p \neq p_n$. Given a graph g and a pointer redirection $\rho = \{p_1 \mapsto q_1, \ldots, p_n \mapsto q_n\}$, we define $\rho(g)$ as the graph whose nodes and labeling function are those of g, whose successor function satisfies $\mathcal{S}_{\rho(g)}(n) = \rho(n_1) \ldots \rho(n_k)$ if $\mathcal{S}_g(n) = n_1 \ldots n_k$ for some $k \geq 0$ and whose roots are $\mathcal{R}\mathrm{oots}_{\rho(g)} = \{\rho(n_1), \ldots, \rho(n_k)\} \cup \{n_1, \ldots, n_k\}$ if $\mathcal{R}\mathrm{oots}_g = \{n_1, \ldots, n_k\}$.

Given two term graphs g and u and a node p of the same sort as $\mathcal{R}\mathrm{oot}_u$, we define the *replacement by u of the subgraph rooted by p in g*, denoted $g[p \leftarrow u]$, in three stages : (1) Let $H = g \oplus u$. (2) Let ρ be the pointer redirection from p to $\mathcal{R}\mathrm{oot}_u$, $H' = \rho(H)$ and $r = \rho(\mathcal{R}\mathrm{oot}_g)$. (3) $g[p \leftarrow u] = H'_{|r}$.

Example 2.2 Let G be the term graph of Example 2.1 and $D =$ $\mathtt{r1:s(r2:g(n4:a,n3:s(n4)))}$. The sum $G \oplus D$ is given by $\mathtt{n1:c(n2:g(n3:s(n4:a),n3),n5:c(n3,n1))} + \mathtt{r1:s(r2:g(n4,n3))}$ (see Fig. 3). Let ρ be the pointer redirection such that $\rho(\mathtt{n2}) = \mathtt{r1}$ and $\rho(p) = p$ for all $p \neq \mathtt{n2}$. The graph $\rho(G \oplus D)$ is defined by $\mathtt{n1:c(r1:s(r2:g(n4:a,n3:s(n4)),n5:c(n3,n1))} + \mathtt{n2:g(n3,n3)}$. Thus the replacement by D of the subgraph rooted by $\mathtt{n2}$ in G is $G[\mathtt{n2} \leftarrow D] = \mathtt{n1:c(r1:s(r2:g(n4:a,n3:s(n4)),n5:c(n3,n1))}$.

A *(rooted) homomorphism* h from a graph g_1 to a graph g_2, denoted $h : g_1 \to g_2$, 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 $h : g_1 \to g_2$ is a homomorphism and g is a subgraph

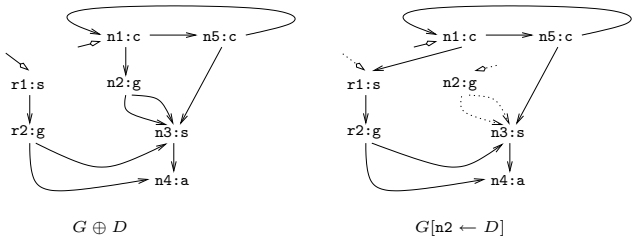

Figure 3:

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.

A term graph l *matches* a graph g at node n if there exists a homomorphism $h : l \rightarrow g_{|n}$. h is called *the matcher* of l on g at node n. Two term graphs 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 exist a graph g and a homomorphism $h : (g_1 \oplus g_2) \rightarrow g$ such that (1) $h(g_1) = h(g_2) = g$ and (2) for all unifiers $h' : G \rightarrow H$, there exists a homomorphism $\phi : g \rightarrow h'(g_1 \oplus g_2)$.

Example 2.3 Consider the subgraph $G_{|\mathtt{n2}}$ of Example 2.1, let $L =$ `l1:g(l2:s(l3:u),l4:s(l5:v))` and μ the mapping from $\mathcal{N}_L$ to $\mathcal{N}_{(G_{|\mathtt{n2}})}$ such that $\mu(\mathtt{l1}) = \mathtt{n2}$, $\mu(\mathtt{l2}) = \mu(\mathtt{l4}) = \mathtt{n3}$ and $\mu(\mathtt{l3}) = \mu(\mathtt{l5}) = \mathtt{n4}$. μ is a homomorphism from L to $G_{|\mathtt{n2}}$, thus L matches G at node `n2`. On the other hand, let $R =$ `r1:s(r2:g(l3:u,l4:s(l5:v)))`. R and L share the subgraphs `l3:u` and `l4:s(l5:v)` whose images by μ are respectively `n4:a` and `n3:s(n4:a)`. Hence $\mu[R] =$ `r1:s(r2:g(n4:a,n3:s(n4)))`, i.e., $\mu[R]$ is the graph D of Example 2.2. Finally, let $L1 =$ `n1:f(n2:a,n3:x)` and $L2 =$ `m1:f(m2:y,m3:s(m4:a))`. $L1$ and $L2$ are unifiable since there exist a term graph $L3 =$ `p1:f(p2:a,p3:s(p4:a))` and a homomorphism $\upsilon : (L1 \oplus L2) \rightarrow L3$ such that $\upsilon(L1) = \upsilon(L2) = L3$.

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. By Id, we mean any substitution such that $\mathcal{D}(Id) = \emptyset$. The *restriction* of σ to a set V of variables, $\sigma_{|V}$, is such that $\mathcal{D}(\sigma_{|V}) = V \cap \mathcal{D}\sigma$ 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 $\sigma(x)$. Applying a substitution on a graph is roughly the same as applying a substitution on a first-order term, except that it preserves the sharing of subgraphs. Given two term graphs g_1 and g_2, we write $g_1 \dot{\leq} g_2$ 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 $x \in (\mathcal{D}\sigma_2 - \mathcal{D}\sigma_1)$. An *idempotent* substitution satisfies $\sigma \circ \sigma = \sigma$. Given two substitutions σ_1 and σ_2 and a set V of variables, we say that σ_1 *is more general than* σ_2 *on* V, denoted $\sigma_1 \dot{\leq} \sigma_2$ $[V]$, if

there exists a substitution θ such that $\theta \circ \sigma_1(x) \doteq \sigma_2(x)$ for all x in V. We write $\sigma_1 \doteq \sigma_2$ $[V]$ iff $\sigma_1 \dot{\leq} \sigma_2$ $[V]$ and $\sigma_2 \dot{\leq} \sigma_1$ $[V]$. Finally, let A be a set of substitutions. We denote by $A/\doteq$ the "quotient" set which consists of substitution representatives of A up to renaming and bisimilarity.

Example 2.4 Let $\sigma(\mathtt{u}) = \sigma(\mathtt{v}) = \mathtt{n4:a}$. The reader may check that $\sigma(L) = \mathtt{l1:g(l2:s(n4:a),l4:s(n4))}$. Let $\sigma' = \{\mathtt{x} \mapsto \mathtt{m1:b(m2:u,n4:a)}\}$. Then, $(\sigma \circ \sigma')_{|\{\mathtt{x,u}\}} = \{\mathtt{x} \mapsto \mathtt{m1:b(n4:a,n4)}, \mathtt{u} \mapsto \mathtt{n4:a}\}$.

3 Weakly Admissible Graph Rewriting Systems

This section introduces briefly the graph rewriting systems (GRSs) we consider (see [8] for details). Let $\Sigma = \langle S, \mathcal{C}, \mathcal{D} \rangle$ be a *constructor-based signature* [14]. In Example 2.1, we assume that `c`, `s` and `a` are constructors ($\in \mathcal{C}$) and `g` is a defined operation ($\in \mathcal{D}$). A *functional node* (resp. *constructor node*, *variable node*) is a node labeled with a defined operation (resp. constructor, variable).

In this paper, we investigate graph narrowing for the class of what we call *admissible term graphs* (atg) [8]. Roughly speaking, 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).

Definition 3.1 *A term graph g is an* admissible term graph (atg) *if there exists no path from a functional node of g to itself. An atg is a* pattern *if it has a tree structure (i.e., linear first-order term) which has one and only one defined operation at its root. A* constructor graph *is a graph with no functional node.*
A rewrite rule *is a graph with two roots, denoted $l \to r$, such that (1) l is a* pattern *(thus an atg), (2) r is an atg, (3) l is not a subgraph of r and (4) $\mathcal{V}_r \subseteq \mathcal{V}_l$. We say that two rules $l_1 \to r_1$ and $l_2 \to r_2$* overlap *iff their left-hand sides are unifiable.*

In Fig. 2, G is an atg and T is a rewrite rule. As for `z:g(z,z)` and `z:g(n:s(z),n)`, they are not atgs since `g` is a defined operation which belongs to a cycle. Condition (3) in the definition of rewrite rule is necessary to prove the stability of the set of atgs w.r.t. the rewrite relation [8].

Definition 3.2 *A* weakly admissible graph rewriting system (WAGRS) *is a pair $SP = \langle \Sigma, \mathcal{R} \rangle$ where Σ is a constructor-based signature and $\mathcal{R}$ is a set of rewrite rules such that if two rules $l_1 \to r_1$ and $l_2 \to r_2$ of $\mathcal{R}$ overlap, then their instantiated right-hand sides are equal up to renaming of nodes, i.e., if there exist a graph g and a homomorphism $h : (l_1 \oplus l_2) \to g$ such that $h(l_1) = h(l_2) = g$, then $h[r_1] \sim h[r_2]$.*

Example 3.3 The following set of rules defines a WAGRS :

```
(R1) l1:f(l2:a,l3:x)        -> r1:d(r1,l3:x)
(R2) l1:f(l2:x,l3:s(l4:y)) -> r1:d(r1,r2:s(r3:a))
(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)  -> r1:a
(R6) l1:i(l2:x,l3:a,l4:y)  -> l2:x
(R7) l1:i(l2:x,l3:y,l4:a)  -> l2:x
```

Indeed, the rules `R1` and `R2` (resp. `R5`, `R6` and `R7`) overlap and their instantiated right-hand sides are equal up to renaming of nodes.

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

Definition 3.4 *Let g_1 be an atg, g_2 a graph, R a rewrite rule and p a node of g_1. A* rewriting step from g_1 to g_2 at node p using the rule R *is defined by $g_1 \rightarrow_{[p, l\rightarrow r]} g_2$ iff there exist a variant $l \rightarrow r$ of R and a homomorphism $h : l \rightarrow g_{1|p}$ (i.e., l matches g_1 at node p) such that $g_2 = g_1[p \leftarrow h[r]]$. In this case, $g_{1|p}$ is called a* redex *of g_1 rooted by p. $\stackrel{*}{\rightarrow}$ denotes the reflexive and transitive closure of $\rightarrow$.*

In [8], we have proved that the rewrite relation is confluent (and confluent modulo the bisimilarity) w.r.t. atgs and WAGRSs.

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

We now introduce parallel graph rewriting over admissible graphs. If parallel rewriting can be easily conceived in the framework of first-order terms, this is unfortunately not the case when one deals with graph structures. The main difficulty comes from the sharing of subgraphs. The reader may find more details in [8].

Definition 3.6 *Let g_1 be an atg, g_2 a graph, $p_1, \ldots, p_n$ n distinct nodes of g_1 and $R_1, \ldots, R_n$ n rewrite rules. A* parallel rewriting step from g_1 to g_2 at nodes $p_1, \ldots, p_n$ using the rules $R_1, \ldots, R_n$, *denoted $g_1 \mathrel{\text{⫲}}\!\!\rightarrow_{[p_1, R_1]\ldots[p_n, R_n]} g_2$, is given by :*

1. *Let $l_i \rightarrow r_i$ be a variant of R_i and $h_i : l_i \rightarrow g_{1|p_i}$ for all $i \in 1..n$.*
2. *Let $H = g \oplus h_1[r_1] \oplus \ldots \oplus h_n[r_n]$.*
3. *Let $\rho_1, \ldots, \rho_n$ be the pointer redirections such that for all $i \in 1..k$, $\rho_i(p_i) = \mathcal{R}oot_{h_i[r_i]}$ and $\rho_i(p) = p$ for all nodes p such that $p \neq p_i$.*
4. *Let $\rho = \rho_{\mu(1)} \circ \ldots \circ \rho_{\mu(n)}$ where $\mu : 1..n \rightarrow 1..n$ is a permutation such that if $i < j$, then there exists no path from $p_{\mu(i)}$ to $p_{\mu(j)}$.*
5. *Let $H' = \rho(H)$ and $r = \rho(\mathcal{R}oot_g)$.*
6. *$g_2 = H'_{|r}$.*

Condition (4) in the previous definition can always be fulfilled. Its rôle is to take into account the relative positions of the different redexes to be transformed so that the parallel rewrite relation $\mathrel{\text{⫲}}\!\!\rightarrow$ can be simulated by the rewrite relation $\rightarrow$ (i.e., if $g_1 \mathrel{\text{⫲}}\!\!\rightarrow g_2$ then $g_1 \stackrel{*}{\rightarrow} g_2$).

Example 3.7 Let g = `p1:d(p2:u(p3:a),p4:v(p2))` be an atg and $l_1 \rightarrow r_1$ and $l_2 \rightarrow r_2$ two rules such that l_1 = `l1:u(l2:x)`, r_1 = `r1:s(l2:x)`, l_2 = `l3:v(l4:y)` and r_2 = `l4:y`. l_1 matches g at node `p2` using the homomorphism $h_1 : l_1 \rightarrow g_{|\texttt{p2}}$ and $h_1[r_1]$ = `r1:s(p3:a)`. On the other hand, l_2 matches g at node `p4` using the homomorphism $h_2 : l_2 \rightarrow g_{|\texttt{p4}}$ and $h_2[r_2]$ = `p2:u(p3:a)`. Let $H = g \oplus h_1[r_1] \oplus h_2[r_2]$ = `p1:d(p2:u(p3:a),p4:v(p2)) + r1:s(p3) + p2`. (see Fig. 4). Let $\rho_1 = \{\texttt{p2} \mapsto \texttt{r1}\}$ and $\rho_2 = \{\texttt{p4} \mapsto \texttt{p2}\}$. There exists a path from `p4` to `p2` in g but none from `p2` to `p4`. So we define $\rho = \rho_1 \circ \rho_2 = \{\texttt{p2} \mapsto \texttt{r1}, \texttt{p4} \mapsto \texttt{r1}\}$. The reader may check that $\rho(H) = H'$ = `p1:d(r1:s(p3:a),r1) + r1 + p2:u(p3) + p4:v(r1)` and $\rho(\mathcal{R}\text{oot}_g)$ = `p1`. Hence, by Def. 3.6, $g \mathrel{\text{⫲}}\!\!\rightarrow_{[\texttt{p2}, l_1\rightarrow r_1][\texttt{p4}, l_2\rightarrow r_2]} g_2$ where g_2 = `p1:d(r1:s(p3:a),r1)`.

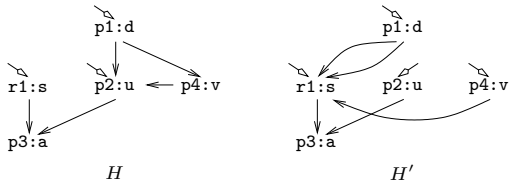

Figure 4:

4 Weakly Needed Graph Narrowing

In this section, we define graph narrowing and extend weakly needed term narrowing [3] to graphs. Roughly speaking, a graph g_2 is obtained from a graph g_1 by means of graph narrowing iff there exists a substitution σ such that $\sigma(g_1)$ rewrites into g_2 :

Definition 4.1 *Let g_1 be an atg, R a rewrite rule, p a non variable node of g_1, σ a substitution and g_2 a graph. A* narrowing step from g_1 to g_2 at node p using the rule R and the substitution σ *is defined by* $g_1 \leadsto_{[p,\,R,\,\sigma]} g_2 \Longleftrightarrow \sigma(g_1) \rightarrow_{[p,\,R]} g_2$.

In this paper, we deliberately use substitutions within narrowing steps instead of homomorphisms [6] for a better readability. Such a substitution may be, for instance, the *most general unifier of $g_{|p}$ w.r.t. the left-hand side of R* :
We say that *an atg g is unifiable w.r.t. a pattern l* iff there exist an idempotent substitution σ and a homomorphism $h : (g \oplus l) \rightarrow \sigma(g)$ such that $h(g) = h(l) = \sigma(g)$. σ is called a *unifier of g w.r.t. l*. We say that σ is a *most general unifier of g w.r.t. l* iff (1) σ is a unifier of g w.r.t. l, (2) the homomorphism $h : (g \oplus l) \rightarrow \sigma(g)$ is a most general unifier of g and l and (3) $\sigma \dot{\leq} \eta$ for all unifiers η of g w.r.t. l.

Example 4.2 Let $H =$ `n1:c(n2:g(n3:x,n3),n4:c(n3,n1))` and $L \rightarrow R$ the rule represented in Fig. 2. We claim that $H_{|\mathtt{n2}}$ is unifiable w.r.t. L. Indeed, let $\sigma = \{\mathtt{x} \mapsto \mathtt{m1:s(m2:z)}\}$. Then, $\sigma(H) =$ `n1:c(n2:g(m1:s(m2:z),m1),n4:c(m1,n1))` and there exists a homomorphism υ from $(H_{|\mathtt{n2}} \oplus L)$ to $\sigma(H_{|\mathtt{n2}})$ such that $\upsilon(\mathtt{n2}) = \upsilon(\mathtt{l1}) = \mathtt{n2}$, $\upsilon(\mathtt{n3}) = \upsilon(\mathtt{l2}) = \upsilon(\mathtt{l4}) = \mathtt{m1}$ and $\upsilon(\mathtt{l3}) = \upsilon(\mathtt{l5}) = \mathtt{m2}$. The reader may check that if $H_1 =$ `n1:c(p1:s(p2:g(m2:z,m1:s(m2))),n4:c(m1,n1))`, then $\sigma(H) \rightarrow_{[\mathtt{n2},\,L \rightarrow R]} H_1$. So we conclude that $H \leadsto_{[\mathtt{n2},\,L \rightarrow R,\,\sigma]} H_1$.

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 $g_1 \overset{*}{\leadsto}_{\sigma} g_2$ iff there exists a derivation $g_1 \leadsto_{[p_1,\,l_1 \rightarrow r_1,\,\sigma_1]} \cdots \leadsto_{[p_k,\,l_k \rightarrow r_k,\,\sigma_k]} g_2$ and $\sigma = (\sigma_k \circ \ldots \circ \sigma_1)_{|\mathcal{V}_{g_1}}$.

Example 4.3 Let $H \leadsto_{[\mathtt{n2},\,L \rightarrow R,\,\sigma]} H_1$ be the narrowing step of Example 4.2. Consider a second narrowing step starting with H_1 and the rule $L' \rightarrow R'$ where $L' =$ `l1:g(l2:a,l3:w)` and $R' =$ `l3:w`. $H_{1|\mathtt{p2}}$ and L' unify and an m.g.u. of $H_{1|\mathtt{p2}}$ w.r.t. L' is $\sigma' = \{\mathtt{z} \mapsto \mathtt{m3:a}\}$. Therefore, $H_1 \leadsto_{[\mathtt{p2},\,L' \rightarrow R',\,\sigma']} H_2$ where $H_2 =$ `n1:c(p1:s(m1:s(m2:z)),n4:c(m1,n1))`. Hence, $H \overset{*}{\leadsto}_{\theta} H2$ with $\theta = (\sigma' \circ \sigma)_{|\mathcal{V}_H} = \{\mathtt{x} \mapsto \mathtt{m1:s(m3:a)}\}$.

Below, we recall 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 4.4 *We say that the narrowing relation $\leadsto$ is* sound *iff for all atgs g, constructor graphs c and substitutions θ such that $g \overset{*}{\leadsto}_\theta c$, there exists a constructor graph s such that $\theta(g) \overset{*}{\rightarrow} s$ and $s \doteq c$. We say that the narrowing relation $\leadsto$ is* complete *iff for all atgs g, constructor graphs c and constructor substitutions θ such that $\theta(g) \overset{*}{\rightarrow} c$, there exist a constructor graph s and a substitution σ such that $g \overset{*}{\leadsto}_\sigma s$, $s \dot{\leq} c$ and $\sigma \dot{\leq} \theta\ [\mathcal{V}_g]$.*

In [6], we have established that most general narrowing $\leadsto$ is sound and complete.

We now define a sequential graph narrowing strategy, denoted $\bar{\Lambda}$, which extends weakly needed term narrowing [3] to atgs. A *sequential graph narrowing strategy*, e.g. $\bar{\Lambda}$, is a partial function which takes an atg g and returns a set of tuples of the form (p, R, σ) such that $g \leadsto_{[p,\,R,\,\sigma]} g'$ for some atg g'. We write $g \leadsto_{\bar{\Lambda}} g'$ iff $g \leadsto_{[p,\,R,\,\sigma]} g'$ and $(p, R, \sigma) \in \bar{\Lambda}(g)$.

The sequential graph narrowing strategy $\bar{\Lambda}$ 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 4.5 *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 $\mathcal{T}$.
A definitional tree *$\mathcal{T}$ of an 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 4.6 Consider the WAGRS of Example 3.3. A definitional tree $\mathcal{T}_{\mathtt{g}}$ of the operation `g` is represented in Fig. 5 and formally defined by :

$$\begin{aligned}\mathcal{T}_{\mathtt{g}} = branch(&\mathtt{k1:g(}\underline{\mathtt{k2:X1}}\mathtt{,k3:X2,k4:X3)},\ \mathtt{k2},\\ &branch(\mathtt{k1:g(k5:a,}\underline{\mathtt{k3:X2}}\mathtt{,k4:X3)},\ \mathtt{k3},\\ &\quad rule(\mathtt{k1:g(k5:a,k6:a,k4:X3)} \rightarrow \mathtt{k4:X3})))\end{aligned}$$

Notice that the rules `R1` and `R2` of Example 3.3 cannot be represented in only one definitional tree. This is why we introduced the notion of fdts. In Fig. 5, 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`.

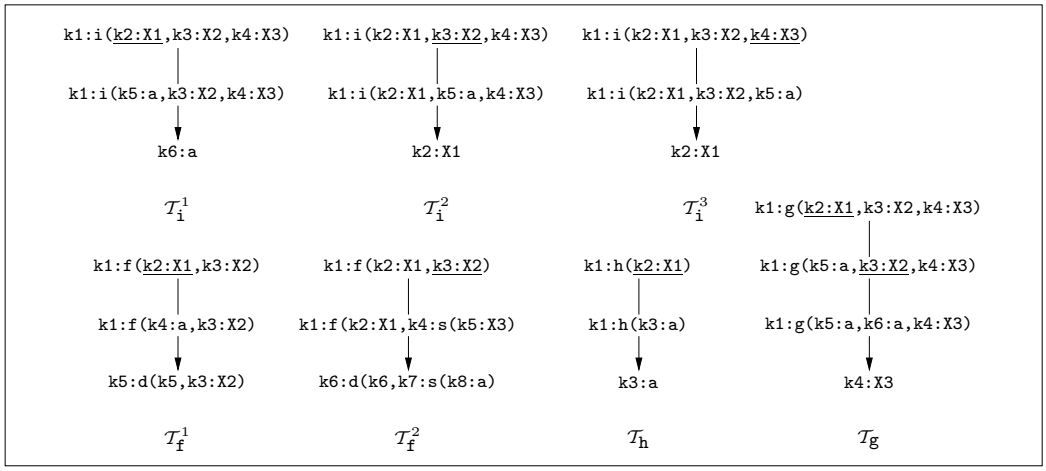

Figure 5:

Our 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 4.7 *Let g be an atg. $\bar{\Lambda}$ 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 an fdt 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 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 :

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

Example 4.8 Consider the WAGRS of Example 3.3. Let $g =$ `n1:f(n2:g(n3:x,n4:y,n5:h(n4)),n6:i(n5,n7:h(n4),n8:a)`.

$$\begin{aligned} \bar{\Lambda}(g) &= \bar{\lambda}(g_{|\mathtt{n1}}, \mathcal{T}^1_{\mathtt{f}}) \cup \bar{\lambda}(g_{|\mathtt{n1}}, \mathcal{T}^2_{\mathtt{f}}) \\ &= \bar{\lambda}(g_{|\mathtt{n2}}, \mathcal{T}_{\mathtt{g}}) \cup \bar{\lambda}(g_{|\mathtt{n6}}, \mathcal{T}^1_{\mathtt{i}}) \cup \bar{\lambda}(g_{|\mathtt{n6}}, \mathcal{T}^2_{\mathtt{i}}) \cup \bar{\lambda}(g_{|\mathtt{n6}}, \mathcal{T}^3_{\mathtt{i}}) \end{aligned}$$

$= \{(\mathtt{n2}, \mathtt{R3}, \sigma_1)\} \cup \bar{\lambda}(g_{|\mathtt{n5}}, \mathcal{T}_\mathtt{h}) \cup \bar{\lambda}(g_{|\mathtt{n7}}, \mathcal{T}_\mathtt{h}) \cup \{(\mathtt{n6}, \mathtt{R7}, Id)\}$
$= \{(\mathtt{n2}, \mathtt{R3}, \sigma_1), (\mathtt{n5}, \mathtt{R4}, \sigma_2), (\mathtt{n7}, \mathtt{R4}, \sigma_2), (\mathtt{n6}, \mathtt{R7}, Id)\}$

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 4.9 *Weakly needed graph narrowing $\rightsquigarrow_{\bar{\Lambda}}$ is sound and complete.*

5 Parallel Graph Narrowing

A graph narrowing step uses a single rewriting step. Therefore, we can improve weakly needed graph narrowing by using a *parallel* rewriting step instead :
A *parallel narrowing step from an atg g_1 to an atg g_2 at nodes $p_1, \ldots, p_n$ using the rules $R_1, \ldots, R_n$ and the substitution σ* is defined by $g_1 \Lsh_{[p_1, R_1] \ldots [p_n, R_n], \sigma} g_2 \iff \sigma(g_1) \Vdash_{[p_1, R_1] \ldots [p_n, R_n]} g_2$.

The definition of a parallel narrowing step needs the computation of substitutions as well as the computation of the different positions used to rewrite in parallel. The computation of substitutions is performed by the *parallel graph narrowing strategy* $\bar{\bar{\Lambda}}$ which is defined below :

Definition 5.1 *Let g be an atg. $\bar{\bar{\Lambda}}$ 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.

Example 5.2 Following Example 4.8, we consider the case of the different triples of $\bar{\Lambda}(g)$ in order to compute $\bar{\bar{\Lambda}}(g)$. $(\mathtt{n2}, \mathtt{R3}, \sigma_1)$ must be eliminated because $\sigma_2 \dot{\leq} \sigma_1 \ [\mathcal{V}_g]$. $(\mathtt{n7}, \mathtt{R4}, \sigma_2)$ must be deleted because the only path from $\mathtt{n1}$ to $\mathtt{n7}$ (i.e., $[\mathtt{n1}, 2, \mathtt{n6}, 2, \mathtt{n7}]$) contains the node $\mathtt{n6}$ and $(\mathtt{n6}, \mathtt{R7}, Id) \in \bar{\Lambda}(g)$ and $Id \dot{\leq} \sigma_2 \ [\mathcal{V}_g]$. $(\mathtt{n5}, \mathtt{R4}, \sigma_2)$ 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., $\mathtt{n2}$), then $\sigma_2 \dot{\leq} \theta \ [\mathcal{V}_g]$. The triple $(\mathtt{n6}, \mathtt{R7}, Id)$ is kept for the same reason. Hence, we conclude that $\bar{\bar{\Lambda}}(g) = \{Id, \sigma_2\}$.

Since a parallel narrowing step requires the computation of a parallel rewriting step, we recall below the definition of the parallel graph rewriting strategy $\bar{\Phi}$. Further details and examples may be found in [8]. A *parallel graph rewriting strategy*,

e.g. $\bar{\Phi}$, is a partial function which takes an atg g and returns a set of pairs (p, R) such that $g \rightarrow_{[p,\,R]} g'$ for some atg g'. We write $g \rightarrow_{\bar{\Phi}} g'$ iff $g \rightarrow_{[p,\,R]} g'$ and $(p, R) \in \bar{\Phi}(g)$.

The parallel graph rewriting strategy $\bar{\Phi}$ uses two auxiliary functions $\bar{\varphi}$ and $Outer$. $\bar{\varphi}$ takes two arguments : an operation-rooted atg and a pdt of this operation.

Definition 5.3 *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 $\bar{\varphi}$ by :*

$$\bar{\varphi}(g, \mathcal{T}) = \begin{cases} \{(p, R)\} & \text{if } \mathcal{T} = rule(\pi \rightarrow r),\ p = \mathcal{R}oot_g \text{ and } R = \pi \rightarrow r\ ; \\ \bar{\varphi}(g, \mathcal{T}_i) & \text{if } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ and} \\ & \quad pattern(\mathcal{T}_i) \text{ matches } g \text{ for some } i \in 1..k\ ; \\ S & \text{if } \mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad \pi \text{ matches } g \text{ using the homomorphism } h, \\ & \quad h(o) \text{ is labeled with a defined operation } f \text{ in } g, \\ & \quad \mathcal{F} = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_k\} \text{ is an fdt of } f \text{ and} \\ & \quad S = \bar{\varphi}(g_{|h(o)}, \mathcal{T}'_1) \cup \ldots \cup \bar{\varphi}(g_{|h(o)}, \mathcal{T}'_k). \end{cases}$$

In the definition above, $\bar{\varphi}(g, \mathcal{T})$ computes a set S of pairs (p, R) where p is a node of g and R is a rule whose left-hand side matches g at node p. Some pairs (p, R) in S may be useless. Therefore, we define the function $Outer(g, S)$ which chooses a maximal set consisting of outermost functional nodes of S w.r.t. g. If an outermost functional node p occurs several times in S, only one pair (p, R) will appear in $Outer(g, S)$. $Outer(g, S)$ can be defined in a deterministic way by using some ordering on the rewrite rules.

Definition 5.4 *Let g be an atg and $S = \{(p_1, R_1), \ldots, (p_n, R_n)\}$ a set of pairs such that p_i is a node of g and R_i is a rewrite rule. We define $Outer(g, S)$ as a maximal subset $\{(q_1, S_1), \ldots, (q_k, S_k)\}$ of S such that :*

1. *For all $i, j \in 1..k$, $i \neq j \Longrightarrow q_i \neq q_j$.*
2. *For all $i \in 1..k$, there exists a path $[\mathcal{R}oot_g, i_0, u_1, i_1, \ldots, i_{k-1}, q_i]$ such that for all $j \in 0..k-1$, for all rewrite rules R, $(u_j, R) \notin S$.*

Definition 5.5 *Let g be an atg, p the leftmost-outermost functional node of g, f the label of the node p in g and $\mathcal{F} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ an fdt of f. $\bar{\Phi}$ is the partial function defined by $\bar{\Phi}(g) = Outer(g, S)$ where $S = \bar{\varphi}(g_{|p}, \mathcal{T}_1) \cup \ldots \cup \bar{\varphi}(g_{|p}, \mathcal{T}_k)$.*

Example 5.6 We have shown in Example 5.2 that $\sigma_2 \in \bar{\bar{\Lambda}}(g)$ where $\sigma_2 = \{\mathtt{y} \mapsto \mathtt{r3 : a}\}$ and $g = \mathtt{n1:f(n2:g(n3:x,n4:y,n5:h(n4)),n6:i(n5,n7:h(n4),n8:a)}$. Let $g'' = \sigma_2(g) = \mathtt{n1:f(n2:g(n3:x,r3:a,n5:h(r3)),n6:i(n5,n7:h(r3),n8:a)}$.
By Definition 5.5, $\bar{\Phi}(g'') = Outer(g'', S)$ where

$$\begin{aligned} S &= \bar{\varphi}(g''_{|\mathtt{n1}}, \mathcal{T}^1_{\mathtt{f}}) \cup \bar{\varphi}(g''_{|\mathtt{n1}}, \mathcal{T}^2_{\mathtt{f}}) \\ &= \bar{\varphi}(g''_{|\mathtt{n2}}, \mathcal{T}_{\mathtt{g}}) \cup \bar{\varphi}(g''_{|\mathtt{n6}}, \mathcal{T}^1_{\mathtt{i}}) \cup \bar{\varphi}(g''_{|\mathtt{n6}}, \mathcal{T}^2_{\mathtt{i}}) \cup \bar{\varphi}(g''_{|\mathtt{n6}}, \mathcal{T}^3_{\mathtt{i}}) \\ &= \emptyset \cup \bar{\varphi}(g''_{|\mathtt{n5}}, \mathcal{T}_{\mathtt{h}}) \cup \bar{\varphi}(g''_{|\mathtt{n7}}, \mathcal{T}_{\mathtt{h}}) \cup \{(\mathtt{n6}, \mathtt{R7})\} \\ &= \{(\mathtt{n5}, \mathtt{R4}), (\mathtt{n7}, \mathtt{R4}), (\mathtt{n6}, \mathtt{R7})\} \end{aligned}$$

$Outer(g'', S)$ selects the outermost redexes of S in g''. Since every path from $\mathcal{R}\mathrm{oot}_{g''}$ to $\mathtt{n7}$ goes through $\mathtt{n6}$, the pair $(\mathtt{n7}, \mathtt{R4})$ is eliminated. So $\bar{\Phi}(g'') = \{(\mathtt{n5}, \mathtt{R4}), (\mathtt{n6}, \mathtt{R7})\}$. The reader may check that $g'' \stackrel{\bar{\Phi}}{\Longrightarrow} g'$ where $g' = \mathtt{n1:f(n2:g(n3:x,r3:a,r3),r3)}$.

In [8], we have proved that the redexes computed by $\bar{\Phi}(g)$ constitute a necessary set of redexes. As a particular case, it is easy to see that in the case of inductively sequential WAGRSs[1] , $\bar{\Phi}(g)$ computes a singleton $\{(p, R)\}$ such that $g_{|p}$ is a needed redex in g [6]. Finally, $\bar{\Phi}$ is a hyper-normalizing strategy (thus a normalizing strategy) w.r.t. the atgs which have a constructor normal form.

Definition 5.7 *The $\bar{\bar{\Lambda}}$-parallel graph narrowing relation $\Downarrow\!\!\rightarrow_{\bar{\bar{\Lambda}}}$ induced by $\bar{\bar{\Lambda}}$ and $\bar{\Phi}$ is defined by $g_1 \Downarrow\!\!\rightarrow_{\bar{\bar{\Lambda}},\sigma} g_2 \Longleftrightarrow \sigma \in \bar{\bar{\Lambda}}(g_1)$ and $\sigma(g_1) \Vdash\!\!\rightarrow_{\bar{\Phi}} g_2$.*

Example 5.8 In Example 5.6, we have seen that $\sigma_2 \in \bar{\bar{\Lambda}}(g)$ and $\sigma_2(g) \Vdash\!\!\rightarrow_{\bar{\Phi}} g'$. So we conclude that $g \Downarrow\!\!\rightarrow_{\bar{\bar{\Lambda}},\sigma_2} g'$.

Theorem 5.9 *$\bar{\bar{\Lambda}}$-parallel graph narrowing $\Downarrow\!\!\rightarrow_{\bar{\bar{\Lambda}}}$ is sound and complete.*

The $\bar{\bar{\Lambda}}$-parallel narrowing relation $\Downarrow\!\!\rightarrow_{\bar{\bar{\Lambda}}}$ inherits all optimality properties of parallel term narrowing [3] :

- $\Downarrow\!\!\rightarrow_{\bar{\bar{\Lambda}}}$ computes only needed graph narrowing derivations [6] in the case of inductively sequential WAGRSs[1].
- $\Downarrow\!\!\rightarrow_{\bar{\bar{\Lambda}}}$ normalizes deterministically ground atgs to constructor atgs.

In addition to the above properties, graph structures induce new improvements for narrowing. Actually, the implementation of $\bar{\lambda}$ is more efficient than its corresponding one for terms. Indeed, thanks to the sharing of subexpression in graph structures, $\bar{\lambda}$ 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.

6 Conclusion

In this paper, we have extended weakly needed term narrowing and parallel term narrowing [3] to graphs in the framework of weakly admissible graph rewriting systems. These new graph narrowing strategies, denoted $\bar{\Lambda}$ and $\bar{\bar{\Lambda}}$ are sound and complete. Moreover, they preserve the same nice properties as that of the term narrowing strategies. As graph narrowing is more efficient than term narrowing, $\bar{\bar{\Lambda}}$ is a good candidate to the implementation of logic functional programming languages.

Nevertheless, parallel graph narrowing can be optimized by using graph collapsing [7]. We say that a graph g_1 collapses into a graph g_2 if both g_1 and g_2 represent the same information but g_2 is more compact than g_1. Therefore, a graph g_2 is obtained from a graph g_1 by means of collapsing graph narrowing if there exist a substitution σ and a graph g_1' such that $\sigma(g_1)$ collapses into g_1' and g_1' rewrites into g_2. In [7], we have extended parallel graph narrowing to parallel collapsing graph narrowing and established that this strategy develops the shortest narrowing derivations that a narrowing based algorithm might ever compute.

[1] An inductively sequential WAGRS is a WAGRS such that the rules of each operation may be stored within one definitional tree.

References

[1] S. Antoy. Definitional trees. In *Proc. of ICALP'92*, pages 143–157. LNCS 632, 1992.

[2] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. In *Proc. of 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 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] 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.

[6] R. Echahed and J.C. Janodet. Admissible graph rewriting and narrowing. In *Proc. of JICSLP'98*, pages 325–340. MIT Press, June 1998.

[7] R. Echahed and J.C. Janodet. On collapsing narrowing. In *Submitted to ICLP'99*. MIT Press, November 1999. Long version available at `ftp://ftp.imag.fr/pub/LEIBNIZ/PMP/collapsing-graph-narrowing.ps.gz`.

[8] R. Echahed and J.C. Janodet. Parallel admissible graph rewriting. In *Recent Dev. in Algebraic Development Techniques*. LNCS 1589, 1999 (to appear). Long version available at `ftp://ftp.imag.fr/pub/LEIBNIZ/PMP/wa-c-graph-rewriting.ps.gz`.

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

[10] A. Habel and D. Plump. Complete strategies for term graph narrowing. In *Recent Dev. in Algebraic Development Techniques*. LNCS 1589, 1999 (to appear).

[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 (ed.). Curry: An integrated functional logic language. Available at `http://www-i2.informatik.rwth-aachen.de/~hanus/curry`, 1999.

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

[14] M. J. O'Donnell. *Computing in Systems Described by Equations*. Springer Verlag LNCS 58, 1977.

[15] 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, to appear.

[16] 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.