Parallel Admissible Graph Rewriting

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. We investigate the rewrite relation over graphs induced by constructor-based weakly orthogonal graph rewriting systems. It is well known that this relation is not confluent in general whereas it is confluent in the case of weakly orthogonal term rewriting systems. We show, however, that the considered relation is always confluent, as well as confluent modulo bisimilarity, for a large class of graphs called admissible graphs. Afterwards, we define a parallel graph rewriting relation and propose an efficient parallel graph rewriting strategy.

1 Introduction

Graph rewriting is being considered in many different areas ; see for instance [9, 15]. The contributions of this paper are theoretical results which concern graph rewriting as operational semantics of functional or algebraic programming languages [13, 5]. There are many straightforward reasons which motivate the use of graphs in the setting of declarative languages. For instance, graphs allow to represent expressions in a compact way thanks to the sharing of sub-expressions; they also permit to handle efficiently cyclic graphs which represent complex data structures as in imperative languages.

In practice, many functional 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 investigate new graph rewriting systems which could be seen as a natural extension to graphs of the well known weakly orthogonal constructor-based term rewriting systems. Below, we give a sample of the considered graph rewriting systems.

```
ones   -> n:cons(1,n)            B(T,F,x) -> ones
f(x,1) -> cons(x,n:cons(1,n))    B(F,x,T) -> ones
f(1,x) -> n:cons(1,n)            B(x,T,F) -> ones
```

It is well known that weakly orthogonal term rewriting systems are confluent. However, this property does not hold for graphs even if the considered graph rewriting system is orthogonal [10]. Indeed, the orthogonal system which consists of the rules `(R1) A(x)` $\rightarrow$ `x` and `(R2) B(x)` $\rightarrow$ `x` induces a confluent rewrite relation on terms and a non confluent rewrite relation on graphs as it is shown by the following counter-example [10] : the graph `n:A(B(n))` may be rewritten into two different normal forms, namely, `u:A(u)` and `v:B(v)`. The source of the non confluence of the graph rewriting system above comes from the use of the so-called collapsing rules. A rule is collapsing if its right-hand side is a variable. However, collapsing rules cannot be prohibited in any programming discipline since most of access functions (e.g., `car`, `cdr`) are defined by means of collapsing rules.

In this paper, we investigate first the confluence of the graph rewrite relation in the framework of weakly orthogonal constructor-based graph rewriting systems. We show that the rewrite relation is confluent, even in the presence of collapsing rules, for a wide class of graphs called admissible graphs. A graph is admissible if its cycles do not include defined operators (see Definition 1).

Efficient implementation of functional languages encodes terms as dags. The soundness of such an encoding can be easily obtained when the considered graph rewriting system is confluent modulo bisimilarity. Two graphs are bisimilar if they represent the same rational term. We show that the considered graph rewriting systems are also confluent modulo bisimilarity.

The property of confluence allows to evaluate admissible graphs in a deterministic way by using rewriting strategies. For constructor-based weakly orthogonal term rewriting systems, efficient strategies have been proposed in the literature. For example, O'Donnell has shown that parallel-outermost strategy is normalizing [12], Sekar and Ramakrishnan [16] as well as Antoy [1] improved O'Donnell's strategy by proposing strategies which rewrite in parallel a necessary set of redexes. Their extension to graphs is not such an easy task. Indeed, the notion of outermost redexes in a cyclic graph is not always meaningful and parallel reductions of graphs that share some subgraphs need some care. We propose a graph rewriting strategy which reduces admissible graphs in parallel at some necessary set of outermost redexes. This strategy is a conservative extension of the one presented for terms in [1].

The rest of the paper is organized as follows : The next section lists some definitions and notations used later in the paper. Section 3 defines

the graph rewriting systems we consider and establishes some confluence results. In Section 4, a parallel graph rewriting relation is proposed as well as an efficient parallel graph rewriting strategy. Section 5 concludes the paper. Due to lack of space, all the proofs have been omitted. They can be consulted in [7, 6].

2 Definitions and notations

Many different notations are used in the literature to investigate graph rewriting [9, 17, 14]. 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]. Some precise definitions which are omitted can be found in [6].

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. (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. A *path* from a node n_0 to a node n_k is a sequence $[n_0, i_0, n_1, \dots, i_{k-1}, n_k]$ of alternating nodes and integers such that $k \geq 1$ and n_{p+1} is the ${i_p}^{th}$ successor of n_p for all $p \in 0..k-1$. (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 [3]. 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 1. In Fig. 1, 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 $\mathcal{S}_G(\mathtt{n5}) = \mathtt{n3.n1}$. An equivalent description of G is `n1:c(n2:g(n3:s(n4:a),n3),n5:c(n3,n1))`. On the other hand, T is a graph with two roots {`l1,r1`} representing a rewrite rule (see Def. 2) : T = `l1:g(l2:s(l3:x),l4:s(l5:y)) + r1:s(r2:g(l3,l4))`.

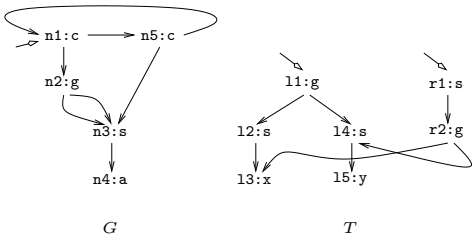

Fig. 1.

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. 1). 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 *pointer redirection* from a node p to a node q is a function $\rho : \mathcal{N} \to \mathcal{N}$ such that $\rho(p) = q$ and $\rho(n) = n$ for all nodes $n \neq p$. More generally, if $p_1, \ldots, p_n, q_1, \ldots, q_n$ are some nodes, we define the *(multiple) pointer redirection* ρ from p_1 to q_1, ..., p_n to q_n as the 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 $\mathcal{R}\text{oots}_{\rho(g)} = \{\rho(n_1), \ldots, \rho(n_k), p_1, p_2, \ldots, p_n\}$ if $\mathcal{R}\text{oots}_g = \{n_1, \ldots, n_k\}$.

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

Example 2. Let G be the term graph of Example 1 and $D =$ `r1:s(r2:g(n4:a,n3:s(n4)))`. The sum $G \oplus D$ is given by `n1:c(n2:g(n3:s(n4:a),n3),n5:c(n3,n1)) + r1:s(r2:g(n4,n3))` (see Fig. 2). Let ρ be the pointer redirection such that $\rho($`n2`$) =$ `r1` and $\rho(p) = p$ for all nodes $p \neq$ `n2`. The graph $\rho(G \oplus D)$ is defined by `n1:c(r1:s(r2:g(n4:a,n3:s(n4)),n5:c(n3,n1)) + n2:g(n3,n3)`. Thus the replacement by D of the subgraph rooted by `n2` in G is defined by $G[$`n2` $\leftarrow D] =$ `n1:c(r1:s(r2:g(n4:a,n3:s(n4)),n5:c(n3,n1))`. An

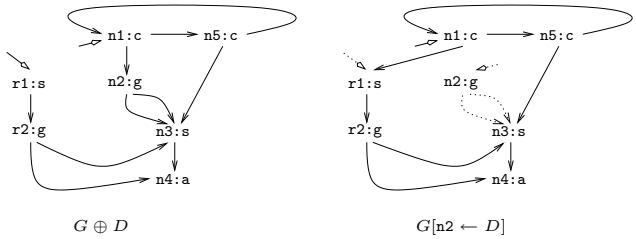

Fig. 2.

example of multiple pointer redirection is shown in Fig. 3 where the graph H' is obtained from H by applying $\rho = \{$`p2` $\mapsto$ `r1`, `p4` $\mapsto$ `r1`$\}$.

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 $\mathcal{R}\text{oots}_{g_2} = h(\mathcal{R}\text{oots}_{g_1})$ 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 $\mathcal{S}_{g_2}(h(n)) = h(\mathcal{S}_{g_1}(n))$ 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 obtained 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 $h : l \to g_{|n}$. h is called *the matcher* of l on g at node n. Two term graphs g_1 and g_2 are *unifiable* if there exist a term graph g and a homomorphism $h : (g_1 \oplus g_2) \to g$ such that $h(g_1) = h(g_2) = g$. Such an h is called a *unifier* of g_1 and g_2.

Example 3. Consider the subgraph $G_{|\mathtt{n2}}$ of Example 1, let $L =$ `l1:g(l2:s(l3:x),l4:s(l5:y))` 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:x,l4:s(l5:y)))`. R and L share the subgraphs `l3:x` and `l4:s(l5:y)` 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. Finally, consider the term graphs $L1 =$ `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 $\upsilon : (L1 \oplus L2) \to L3$ such that $\upsilon(L1) = \upsilon(L2) = L3$.

3 Admissible graph rewriting

This section introduces the different classes of graph rewriting systems we consider and establishes new confluence results. 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* whose rôle consists in building data structures and $\mathcal{D}$ 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. For instance, in Example 1, we suppose that `c`, `s` and `a` are constructors and `g` is a defined operation.

In the rest of the paper, we investigate graph rewriting for the class of *admissible* term graphs (atg). 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 1. *Let g be a term graph over a constructor-based signature $\Sigma = \langle S, \mathcal{C}, \mathcal{D} \rangle$. A node $n \in \mathcal{N}_g$ is called a* defined node *if $\mathcal{L}_g(n)$ is a defined operation ($\in \mathcal{D}$). 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 has one and only*

one defined operation at its root. A constructor graph *is a graph with no defined node. An atg is* operation-rooted *if its root is a defined node.*

Example 4. The term graph G of Example 1 is admissible but `z:g(z,z)` and `z:g(n:s(z),n)` are not (since `g` is a defined operation which belongs to a cycle). `l1:g(l2:a,l3:a)` is a pattern whereas `l1:g(l2:a,l2)` is not.

The next definition introduces the notion of admissible rewrite rule. Such rules are tailored so that the set of atgs is closed by the rewrite relation induced by admissible rules (see Remark 1).

Definition 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 an* admissible rule *iff (1) l is a* pattern *(thus an atg), (2) r is an atg, (3) l is not a subgraph of r and (4) $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. 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. We say that two admissible rules $l_1 \to r_1$ and $l_2 \to r_2$* overlap *iff their left-hand sides are unifiable.*

Example 5. The graph T of Example 1 is an admissible rule. An example of a non admissible rule is given in Remark 1.

Definition 3. *A* constructor-based graph rewriting system (cGRS) *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.*
We say that SP is an admissible graph rewriting system (AGRS) *iff every two distinct rules in $\mathcal{R}$ do not overlap.*
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 $l_1 \to r_1$ and $l_2 \to r_2$ overlap, then their instantiated right-hand sides are equal up to renaming of nodes, i.e., if there exist a term 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 6. Consider the following cGRS :

```
(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(l4:y))
(R3) l1:g(l2:a,l3:x) -> r1:s(r2:a)
(R4) l1:g(l2:x,l3:a) -> r1:s(l2:x)
(R5) l1:g(l2:s(l3:x),l4:s(l5:y)) -> r1:s(r2:g(l3:x,l4:s(l5:y)))
(R6) l1:h(l2:x,l3:y) -> r1:h(l2:x,l3:y)
```

`R1` and `R2` (resp. `R3` and `R4`) overlap and their instantiated right-hand sides are equal up to renaming of nodes. Thus this cGRS is a WAGRS.

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

Definition 4. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, 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. $\overset{*}{\to}$ denotes the reflexive and transitive closure of $\to$.*

Example 7. According to Example 3, L matches G at node `n2` with homomorphism μ and $\mu[R]$ = `r1:s(r2:g(n4:a,n3:s(n4)))`. As $\mu[R]$ is equal to the graph D of Example 2, we infer that $G[\texttt{n2} \leftarrow \mu[R]]$ = `n1:c(r1:s(r2:g(n4:a,n3:s(n4))),n5:c(n3,n1))`. Let G' be this last graph. By Definition 4, $G \to_{[\texttt{n2}, L \to R]} G'$.

We can prove that the set of atgs is stable w.r.t. rewriting with admissible rules [6], i.e., if g_1 is an atg and $g_1 \to_{[p, l \to r]} g_2$ is a rewriting step, then g_2 is an atg. In the following example, we show that the set of atgs is not stable w.r.t. non admissible rewrite rules :

Remark 1. The rule `l1:g(l2:a,l3:x)` $\to$ `r1:g(l1,r2:a)` 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, i.e., Condition 3 of Def. 2 is not fulfilled. By using this rule, the reader may check that the atg `n1:g(n2:a,n3:a)` is rewritten into `r1:g(r1,r2:a)`. This last graph is not an atg, since the defined operation `g` belongs to a cycle.

The property of confluence is one of the main properties of rewrite relations. We consider here the classical notion of confluence as well as confluence modulo bisimilarity. The reader may find in [4] a survey of confluence properties over acyclic term graphs.

Definition 5. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. We say that the* rewriting relation $\to$ is confluent modulo renaming of nodes $\sim$ (resp. bisimilarity $\doteq$) 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. $g_1 \doteq g_2$), $g_1 \overset{*}{\to} g_1'$ and $g_2 \overset{*}{\to} g_2'$, there exist two atgs g_1'' and g_2'' such that $g_1' \overset{*}{\to} g_1''$, $g_2' \overset{*}{\to} g_2''$ and $g_1'' \sim g_2''$ (resp. $g_1'' \doteq g_2''$).*

We have seen in Section 1 that confluence (modulo bisimilarity) of AGRSs is not a straightforward extension of that of orthogonal term

rewriting systems. In [11], it is proved that orthogonal graph rewriting systems (and thus AGRSs) are confluent modulo the equivalence of the so-called hypercollapsing graphs. A graph g is hypercollapsing if $g \to g$. We are not interested in confluence modulo the equivalence of hypercollapsing graphs in the present paper.

Theorem 1. *Let SP be a WAGRS. $\to$ is confluent modulo $\sim$ and $\doteq$ w.r.t. atgs.*

Since bisimilar graphs are naturally handled in the setting of graphs, one may wonder whether the class of weakly orthogonal graph rewriting systems, such that the instantiated right-hand sides of overlapping rules are bisimilar (and not necessarily equal up to renaming of nodes), still defines a rewrite relation which is confluent modulo bisimilarity or not.

Definition 6. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. We say that SP is a* bisimilar weakly admissible graph rewriting system (bWAGRS) *iff $\mathcal{R}$ is a set of admissible rules such that if two rules $l_1 \to r_1$ and $l_2 \to r_2$ overlap, then their instantiated right-hand sides are bisimilar, i.e., if there exist a term graph g and a homomorphism $h : (l_1 \oplus l_2) \to g$ such that $h(l_1) = h(l_2) = g$, then $h[r_1] \doteq h[r_2]$.*

Example 8. Consider the following rules :

```
(S1) l1:f(l2:a,l3:x) -> r1:d(r2:d(r1,l3:x),l3)
(S2) l1:f(l2:x,l3:s(l4:a)) -> r1:d(r1,r2:s(r3:a))
```

As the rules `S1` and `S2` overlap and their instantiated right-hand sides are bisimilar, this cGRS is a bWAGRS.

It is clear that a bWAGRS cannot be confluent modulo renaming of nodes, in general. Nevertheless, a bWAGRS may be confluent modulo bisimilarity. We can prove this property in the case where a bWAGRS is Noetherian w.r.t. atgs (i.e., for all atgs g, there is no infinite derivation $g \to g_1 \to g_2 \to \ldots$).

Theorem 2. *Let SP be a bWAGRS which is Noetherian w.r.t. atgs. Then $\to$ is confluent modulo $\doteq$ w.r.t. atgs.*

When a bWAGRS is not Noetherian w.r.t. atgs, we conjecture that the rewrite relation $\to$ is still confluent modulo $\doteq$ w.r.t. atgs.

4 Parallel admissible graph rewriting

In this section we propose a definition of parallel graph rewriting relation and define an efficient strategy for it. If parallel rewriting can be easily conceived in the framework of first-order terms, this is unfortunately not the case when one has to deal with graph structures. The main difficulty comes essentially from the sharing of subgraphs. In [13, Chap. 14], parallel graph rewriting has been investigated w.r.t. implementation point of view. That is to say, some annotations are proposed to be added to right-hand sides in order to indicate which redexes can be evaluated in parallel. In the following definition, we are rather interested in the general notion of *parallel* graph rewriting which allows to reduce several arbitrary redexes of an atg in one shot.

Definition 7. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g_1 an atg, g_2 a graph, $l_1 \to r_1, \ldots, l_n \to r_n$ n variants of rewrite rules of $\mathcal{R}$ and $p_1, \ldots, p_n$ n distinct nodes of g_1. We say that g_1* rewrites in parallel to g_2 at nodes $p_1, \ldots, p_n$ using the rules $l_1 \to r_1, \ldots, l_n \to r_n$ *and write* $g_1 \nrightarrow\!\!\!\!\!\!\nrightarrow_{[p_1, l_1 \to r_1] \ldots [p_n, l_n \to r_n]} g_2$ *iff :*

1. *There exists n homomorphisms $h_i : l_i \to g_{1|p_i}$ 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 $\mu : 1..n \to 1..n$ 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 $\nrightarrow\!\!\!\!\!\!\nrightarrow$ can be simulated by sequential rewriting $\to$. Indeed, consider for example the pointer redirection ρ' such that $\rho'(p_i) = \mathcal{R}\mathrm{oot}_{h_i[r_i]}$ for all $i \in 1..n$ and $\rho'(p) = p$ for all nodes p such that $p \neq p_1, \ldots, p \neq p_n$. ρ', which seems to be the natural candidate as a pointer redirection to define a parallel rewrite relation, does not always satisfy Condition (4). The reader may verify (see also [7]) that the parallel rewrite relation induced by ρ', say $\nrightarrow\!\!\!\!\!\!\nrightarrow$, cannot be simulated by sequential rewriting $\to$, i.e., $\nrightarrow\!\!\!\!\!\!\nrightarrow$ is not included in $\xrightarrow{*}$.

Example 9. Let $g =$ `p1:d(p2:u(p3:a),p4:v(p2))` be an atg and $l_1 \to r_1$ and $l_2 \to r_2$ two admissible 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 $h_1 : l_1 \to g_{|\mathtt{p2}}$ and $h_1[r_1]$ = r1:s(p3:a). On the other hand, l_2 matches g at node p4 using the homomorphism $h_2 : l_2 \to g_{|\mathtt{p4}}$ 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. 3). Let ρ_1 =

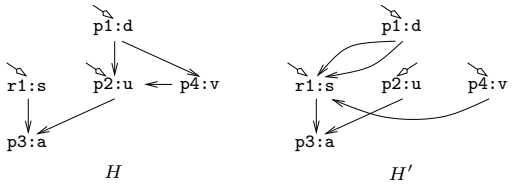

Fig. 3.

$\{\mathtt{p2} \mapsto \mathtt{r1}\}$ and $\rho_2 = \{\mathtt{p4} \mapsto \mathtt{p2}\}$. There exists a path from p4 to p2 in g but none from p2 to p4. So we define $\rho = \rho_1 \circ \rho_2 = \{\mathtt{p2} \mapsto \mathtt{r1}, \mathtt{p4} \mapsto \mathtt{r1}\}$. The reader may check that $\rho(H) = H'$ = p1:d(r1:s(p3:a),r1) + r1 + p2:u(p3) + p4:v(r1) and $\rho(\mathcal{R}\mathrm{oot}_g)$ = p1. Hence, by Definition 7, $g \Rightarrow\!\!\!\!\!\!+\!\!\!\!+\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow_{[\mathtt{p2},l_1 \to r_1][\mathtt{p4},l_2 \to r_2]} g_2$ where g_2 = p1:d(r1:s(p3:a),r1).

Proposition 1. *Let SP be a cGRS. If $g_1 \Rightarrow\!\!\!\!\!\!+\!\!\!\!+\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow g_2$, then $g_1 \xrightarrow{*} g_2$. If $g_1 \to g_2$, then $g_1 \Rightarrow\!\!\!\!\!\!+\!\!\!\!+\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow g_2$.*

Theorem 3. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an WAGRS. $\Rightarrow\!\!\!\!\!\!+\!\!\!\!+\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow$ is confluent modulo $\sim$ and $\doteq$ w.r.t. atgs.*

In the rest of this section we describe a parallel graph rewriting strategy for WAGRSs which is c-hyper-normalyzing and computes necessary sets of redexes. We need first some preliminary definitions.

Definition 8. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. A* parallel graph rewriting strategy *is a partial function $\bar{\mathcal{S}}$ which takes an atg g and returns* a set of pairs (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 using the rule R. We write $g \to_{\bar{\mathcal{S}}} g'$ to denote the* parallel $\bar{\mathcal{S}}$-step *from g to g' such that $g \Rightarrow\!\!\!\!\!\!+\!\!\!\!+\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow_{\bar{\mathcal{S}}(g)} g'$. A strategy $\bar{\mathcal{S}}$ is* c-normalizing *iff for all atgs g and constructor graphs c such that $g \xrightarrow{*} c$, there exists a graph c' such that $g \Rightarrow\!\!\!\!\!\!+\!\!\!\!+\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow^*_{\bar{\mathcal{S}}} c'$ and $c' \doteq c$. A strategy $\bar{\mathcal{S}}$ is* c-hyper-normalizing *iff for all atgs g and constructor graphs c such that $g \xrightarrow{*} c$, every derivation D starting with g which alternates $\bar{\mathcal{S}}$-steps with other reduction steps ends with a constructor graph c' such that $c' \doteq c$.*

Definition 9. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g and g' two atgs and $B = g \xrightarrow{*} g'$ 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 graph, a descendant of $g_{|q}$ is rewritten at its root q. A set of redexes $S = \{u_1, \ldots, u_n\}$ of g is a* necessary set of redexes *iff in every rewriting derivation from g to a constructor graph, a descendant of at least one redex $u \in S$ is rewritten at its root. 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 10. Let $g =$ `n1:f(n2:f(n3:a,n3),n4:f(n2,n3))` where `f` is defined with the rules `(T1) l1:f(l2:x,l3:a)` $\rightarrow$ `l2:x` and `(T2) l1:f(l2:a,l3:y)` $\rightarrow$ `l3:y`. The subgraphs $g_{|\texttt{n2}}$ and $g_{|\texttt{n4}}$ are outermost redexes (though there exists a path from `n4` to `n2`). The leftmost-outermost defined node of g is `n2`. The set $S = \{g_{|\texttt{n2}}, g_{|\texttt{n4}}\}$ is a necessary set of redexes of g. None of them is a needed redex of g. Actually, the notion of needed redex is irrelevant in the framework of WAGRSs.

Our strategy is based on an extension of definitional trees [1] to the context of WAGRSs. A definitional tree 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 10. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a WAGRS. 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 node of π, o is labelled with a variable, 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 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 11. Consider the WAGRS defined in Example 6. A definitional tree $\mathcal{T}^1_{\mathtt{g}}$ of the operation `g` is represented in Fig. 4 and formally defined by

$$\begin{aligned}\mathcal{T}^1_{\mathtt{g}} = \mathit{branch}(&\mathtt{k1{:}g(\underline{k2{:}X1},k3{:}X2)}, \mathtt{k2},\\ &\mathit{rule}(\mathtt{k4{:}g(k5{:}a,k6{:}X3)} \rightarrow \mathtt{k7{:}s(k8{:}a)}),\\ &\mathit{branch}(\mathtt{k9{:}g(k10{:}s(k11{:}X4),\underline{k12{:}X5})}, \mathtt{k12},\\ &\quad \mathit{rule}(\mathtt{k13{:}g(k14{:}s(k15{:}X6),k16{:}s(k17{:}X7))} \rightarrow\\ &\quad\quad \mathtt{k18{:}s(k15,k16)}))))\end{aligned}$$

Notice that the rule `R4` of Example 6 is not represented by a leaf of $\mathcal{T}^1_{\mathtt{g}}$. Actually, it is impossible to build only one definitional tree which contains all the rules defining `g`. This is why we introduced the notion of fdts. In Fig. 4, we represent possible fdts $\mathcal{F}_{\mathtt{f}} = \{\mathcal{T}^1_{\mathtt{f}}, \mathcal{T}^2_{\mathtt{f}}\}$, $\mathcal{F}_{\mathtt{g}} = \{\mathcal{T}^1_{\mathtt{g}}, \mathcal{T}^2_{\mathtt{g}}\}$ and $\mathcal{F}_{\mathtt{h}} = \{\mathcal{T}_{\mathtt{h}}\}$ corresponding to the operations `f`, `g` and `h` of Example 6.

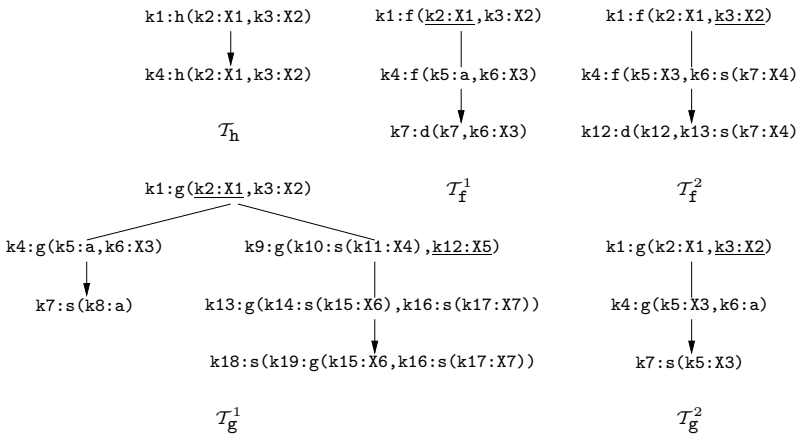

Fig. 4.

We are now ready to define our parallel graph rewriting strategy $\bar{\Phi}$. The strategy $\bar{\Phi}$ is a partial function that operates on atgs in the presence of a WAGRS. $\bar{\Phi}(g)$ returns, when it is possible, a set of pairs (p, R) where p is a node of g and R is a rewrite rule such that g can be rewritten at

node p using the rule R. $\bar{\Phi}$ uses two auxiliary functions $\bar{\varphi}$ and $Outer$. $\bar{\varphi}$ takes two arguments : an operation-rooted atg and a pdt of this operation.

Definition 11. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a WAGRS, g 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 :*

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

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 defined nodes of S w.r.t. g. If an outermost defined 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 12. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a WAGRS, g 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 an admissible 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 13. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a WAGRS, g an atg, p the leftmost-outermost defined node of g, f the label of the node p in g and $\mathcal{F} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ a forest of definitional trees 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 12. Consider the fdts of Example 11. Let $g =$ `n1:c(n2:f(n3:h(n4:g(n5:a,n5),n6:g(n4,n5)),n6),n7:c(n6,n1))`. By Definition 13, $\bar{\Phi}(g) = Outer(g, S)$ where

$$
\begin{aligned}
S &= \bar{\varphi}(g_{|\mathtt{n2}}, \mathcal{T}^1_{\mathtt{f}}) \cup \bar{\varphi}(g_{|\mathtt{n2}}, \mathcal{T}^2_{\mathtt{f}}) \\
&= \bar{\varphi}(g_{|\mathtt{n3}}, \mathcal{T}_{\mathtt{h}}) \cup \bar{\varphi}(g_{|\mathtt{n6}}, \mathcal{T}^1_{\mathtt{g}}) \cup \bar{\varphi}(g_{|\mathtt{n6}}, \mathcal{T}^2_{\mathtt{g}}) \\
&= \{(\mathtt{n3}, \mathtt{k1{:}h(k2{:}X1,k3{:}X2)} \rightarrow \mathtt{k4{:}h(k2,k3)})\} \cup \\
&\quad \bar{\varphi}(g_{|\mathtt{n4}}, \mathcal{T}^1_{\mathtt{g}}) \cup \bar{\varphi}(g_{|\mathtt{n4}}, \mathcal{T}^2_{\mathtt{g}}) \cup \\
&\quad \{(\mathtt{n6}, \mathtt{k4{:}g(k5{:}X3,k6{:}a)} \rightarrow \mathtt{k7{:}s(k5)})\} \\
&= \{(\mathtt{n3}, \mathtt{k1{:}h(k2{:}X1,k3{:}X2)} \rightarrow \mathtt{k4{:}h(k2,k3)}), \\
&\quad (\mathtt{n4}, \mathtt{k4{:}g(k5{:}a,k6{:}X3)} \rightarrow \mathtt{k7{:}s(k8{:}a)}), \\
&\quad (\mathtt{n4}, \mathtt{k4{:}g(k5{:}X3,k6{:}a)} \rightarrow \mathtt{k7{:}s(k5)}), \\
&\quad (\mathtt{n6}, \mathtt{k4{:}g(k5{:}X3,k6{:}a)} \rightarrow \mathtt{k7{:}s(k5)})\}
\end{aligned}
$$

By Condition (2) in Def. 12, $Outer(g, S)$ selects the outermost redexes of S in g, thus $\bar{\Phi}(g) = \{(\mathtt{n3}, \mathtt{R6}), (\mathtt{n6}, \mathtt{R4})\}$. Notice that S contains two pairs with the node $\mathtt{n4}$, namely $(\mathtt{n4}, \mathtt{R3})$ and $(\mathtt{n4}, \mathtt{R4})$. If $g_{|\mathtt{n4}}$ was also an outermost redex of g, then Condition (1) of Def. 12 requires to choose one pair among $(\mathtt{n4}, \mathtt{R3})$ and $(\mathtt{n4}, \mathtt{R4})$ in order to compute $\bar{\Phi}(g)$. This choice is irrelevant since rewriting g using $\mathtt{R3}$ or $\mathtt{R4}$ leads to the same graph (by the definition of a WAGRS).

Theorem 4. *The redexes computed by $\bar{\Phi}(g)$ constitute a necessary set of redexes.*

As a particular case of the theorem above, it is easy to see that if SP is an AGRS such that for all defined operations f, there exists one definitional tree which contains all the rules defining f, then $\bar{\Phi}(g)$ computes a singleton $\{(p, R)\}$ such that $g_{|p}$ is a needed redex in g [6].

From the previous theorem, we deduce that if an atg g can be rewritten to a constructor graph c, then every derivation from g to c must contain a step which rewrites a node p using a rule R for some $(p, R) \in \bar{\Phi}(g)$.

Theorem 5. *$\bar{\Phi}$ is c-hyper-normalizing (thus c-normalizing).*

5 Conclusion

We gave new confluence results for a wide class of programs described by means of constructor-based graph rewriting systems and conjecture that the bisimilar weakly orthogonal admissible graph rewriting systems are confluent modulo bisimilarity. We also proposed a c-hypernormalizing parallel graph rewriting strategy, $\bar{\Phi}$, which computes necessary sets of redexes in the presence of weakly orthogonal admissible graph rewriting systems. In fact, this set is optimal with respect to strategies that do not consider the right-hand sides of the rules. The proof of this claim needs for example the machinery of so-called arbitrary reductions. The

reader may find in [16, 2] good hints for it. Our strategy, which computes redexes that can be reduced in parallel, departs from the parallel graph rewriting proposed in [13] where emphasis is made on the way parallelism could be implemented on parallel machines. However, $\bar{\Phi}$ can be efficiently implemented by using the annotations proposed in [13]. $\bar{\Phi}$ can also be lifted to a complete narrowing strategy by extending the results in [8].

References

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

[2] 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.

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

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

[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. 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.

[7] R. Echahed and J. C. Janodet. On weakly orthogonal constructor-based graph rewriting. Technical report, 1998. Available via URL : ftp://ftp.imag.fr/pub/LEIBNIZ/ATINF/wa-c-graph-rewriting.ps.gz.

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

[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] 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.

[11] J. R. Kennaway, J. K. Klop, M. R. Sleep, and F. J. De Vries. Transfinite reduction in orthogonal term rewriting systems. *Information and Computation*, 119(1):18–38, 1995.

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

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

[14] 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.

[15] Grzegorz Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 1. World Scientific, 1997.

[16] R. C. Sekar and I. V. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. *Information and Computation*, 104(1):78–109, May 1993.

[17] 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.