On Constructor-based Graph Rewriting Systems

R. Echahed J.C. Janodet
IMAG-LSR, CNRS
B.P.72 Cedex
F-38402 Grenoble
France
{echahed, janodet}@imag.fr

Research Report IMAG 985-I

Draft of October 8, 1997

Abstract

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

On Constructor-based Graph Rewriting Systems

R. Echahed J.C. Janodet
IMAG-LSR, CNRS
B.P.72 Cedex
F-38402 Grenoble
France
{echahed, janodet}@imag.fr

Abstract

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

1 Introduction

Graph rewriting is being investigated in various areas nowadays (see for instance [Cou93, ET96, SPvE93]). In this paper, we consider graph rewriting as the underlying operational semantics of rule-based (functional or logic) programming languages (e.g. [PvE93, AEH$^+$96]). There are many reasons that motivate the use of graphs. They actually allow sharing of subexpressions which leads to efficient computations. They also permit to go beyond the processing of first-order terms by handling efficiently real-world data types represented by cyclic graphs.

Using a graph rewriting system (GRS) is not an easy task. Indeed, the classical properties of term rewriting systems (TRS) cannot be lifted without caution to GRSs. One of these properties is confluence. Let us consider the rule $F(a, a, x) \to x$ where a is a constant and x is a variable. This rule, which constitutes an orthogonal TRS, generates a confluent rewrite relation over (finite or infinite) terms whereas it generates a non confluent rewrite relation over graphs as it is witnessed by the following counter-example [KKSV94]. The graph K (cf. figure 1) rewrites into two syntactically different graphs K_1 and K_2. But, K_1 and K_2 cannot rewrite to a same graph. It is well-known

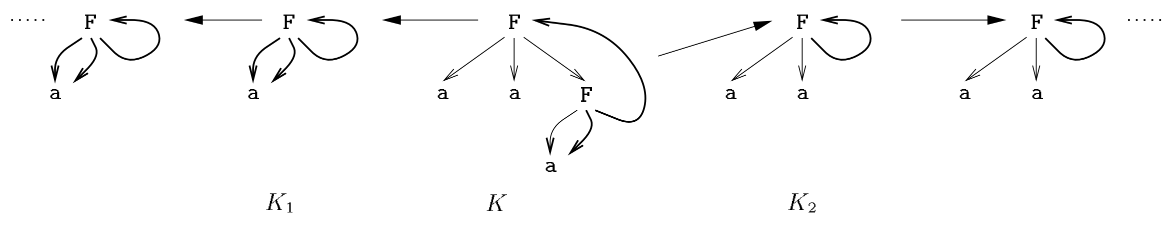

Figure 1:

that this source of nonconfluency of GRSs comes from the so-called "collapsing rules" in orthogonal GRSs. A rewrite rule is collapsing if its right-hand side is a variable. However, collapsing rules are very often used in programming and thus cannot be prohibited in any programming discipline. Most of access functions are defined by means of collapsing rules, e.g.,

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

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

It is notorious that finite graphs represent rational terms. Sometimes, one would like to equate two graphs if they represent the same rational term. In this case we say that the two graphs are *bisimilar* (if we refer to the theory of concurrency). We show the confluence modulo bisimilarity of the considered GRSs. This result is mandatory to prove the completeness of narrowing [EJ97].

The confluence of a rewrite relation allows to evaluate expressions in a deterministic and efficient way by using rewriting strategies. Such strategies have been well investigated in the setting of finite and infinite orthogonal TRSs (e.g., [O'D77, HL91, KKSV95]). In [Ant92], a strategy that computes outermost needed redexes based on definitional trees has been designed in the framework of orthogonal constructor-based TRSs. In this article, we show that Antoy's strategy can be extended to orthogonal constructor-based GRSs with the same nice properties. We particularly prove that the resulting strategy is c-hyper-normalizing on the class of admissible graphs and develops shortest derivations.

The rest of the paper is organized as follows. The next section contains some preliminaries on graphs. In section 3, we introduce the framework of constructor-based GRS. Section 4 exhibits the results concerning confluence and section 5 deals with confluence modulo bisimilarity. We define our rewriting strategy in section 6 and list its properties.

2 Preliminaries on graphs

The aim of this section is to precise the definitions we use in our paper. Many different notations are used in the literature to investigate graph rewriting : systems of equations [AK96], hypergraphs [HP95], graphs [KKSV94] among others. We are mostly consistent with [BvEG$^+$87].

2.1 Definition of graphs

Definition 1 (Signature)
A *many-sorted signature* $\Sigma = \langle S, \Omega \rangle$ consists of a set S of sorts and an S-indexed family $\Omega = \uplus_{s \in S} \Omega_s$ with $\Omega_s = \uplus_{(w,s) \in S^* \times S} \Omega_{w,s}$ of sets of operation symbols. 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, *rank* $s_1 \dots s_n$ and *profile* $s_1 \dots s_n, s$. □

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.* Both $\mathcal{X}$ and $\mathcal{N}$ are fixed throughout the rest of the paper.

Definition 2 (Graph)
A *(rooted) graph* g over $\langle \Sigma, \mathcal{N}, \mathcal{X} \rangle$ is a tuple $g = \langle \mathcal{N}_g, \mathcal{L}_g, \mathcal{S}_g, \mathcal{R}\mathrm{oots}_g \rangle$ such that :

1. $\mathcal{N}_g$ is the set of nodes of g, i.e., $\mathcal{N}_g = \uplus_{s \in S}(\mathcal{N}_g)_s$ with $(\mathcal{N}_g)_s \subseteq \mathcal{N}_s$.

2. $\mathcal{L}_g$, the *labeling function of* g, is an S-indexed family of functions associating an operation symbol or a variable to each node of g, i.e., $\mathcal{L}_g = \uplus_{s \in S}(\mathcal{L}_g)_s$ with $(\mathcal{L}_g)_s : (\mathcal{N}_g)_s \to \Omega_s \cup \mathcal{X}_s$.

3. $\mathcal{S}_g$, the *successor function of* g, is an S-indexed family of functions associating a (possibly empty) string of nodes to each node of g, i.e., $\mathcal{S}_g = \uplus_{s \in S}(\mathcal{S}_g)_s$ with $(\mathcal{S}_g)_s : (\mathcal{N}_g)_s \to \mathcal{N}_g^*$ such that for every node $n \in (\mathcal{N}_g)_s$:

   - if $(\mathcal{L}_g)_s(n) = f$ with $f : s_1 \ldots s_k \to s$, then there exist $n_1, \ldots, n_k \in \mathcal{N}_g$ such that $(\mathcal{S}_g)_s(n) = n_1 \ldots n_k$ and $n_i \in (\mathcal{N}_g)_{s_i}$ for all $i \in 1..k$.
   - if $(\mathcal{L}_g)_s(n) = c$ with $c \in \Omega_{\varepsilon,s}$ (c is a constant), then $(\mathcal{S}_g)_s(n) = \varepsilon$ (i.e., n has no successor).
   - if $(\mathcal{L}_g)_s(n) = x$ with $x \in \mathcal{X}_s$ (x is a variable), then $(\mathcal{S}_g)_s(n) = \varepsilon$.

   We write $n \in \mathcal{S}_g(m)$ if n is a successor of m.

4. $\mathcal{R}\mathrm{oots}_g$ is a nonempty set of distinguished nodes of g, called its *roots* : $\mathcal{R}\mathrm{oots}_g \subseteq \mathcal{N}_g$.

5. Every variable of g labels one and only one node, i.e., for all nodes n_1, n_2 of g, if $\mathcal{L}_g(n_1) \in \mathcal{X}$ and $\mathcal{L}_g(n_2) \in \mathcal{X}$, then $\mathcal{L}_g(n_1) = \mathcal{L}_g(n_2)$ implies $n_1 = n_2$.

6. *Connectivity condition :* Each node of g is either a root or is accessible from a root (see definition 3).

A graph g is called *term graph* iff it has one root $\mathcal{R}\mathrm{oot}_g$ (i.e., $\mathcal{R}\mathrm{oots}_g = \{\mathcal{R}\mathrm{oot}_g\}$). We write $\mathcal{V}(g)$ for the set of variables appearing in g. □

Example 1 In figure 2, we give two examples of graphs. Their roots are pointed by arrows without any antecedent. The left-hand graph, G, represents a cyclic list alternating the expressions `succ(0) + succ(0)` and `succ(0)`. G is a term graph, since it has just one root $\mathcal{R}\mathrm{oot}_G = \{$`x1`$\}$. Nodes of G are $\mathcal{N}_G = \{$`x1`, ..., `x5`$\}$. Its labeling function is defined by $\mathcal{L}_G($`x1`$) = \mathcal{L}_G($`x5`$) =$ `cons`, $\mathcal{L}_G($`x2`$) =$ `+`, $\mathcal{L}_G($`x3`$) =$ `succ`, et $\mathcal{L}_G($`x4`$) =$ `0`. Its successor function is defined by $\mathcal{S}_G($`x1`$) =$ `x2.x5` , $\mathcal{S}_G($`x2`$) =$ `x3.x3` , $\mathcal{S}_G($`x3`$) =$ `x4`, $\mathcal{S}_G($`x4`$) = \varepsilon$, $\mathcal{S}_G($`x5`$) =$ `x3.x1` . The right-hand graph, T represents the classical rewriting rule `succ(x)+y` $\to$ `succ(x+y)` (cf. definition 20). T has two roots $\mathcal{R}\mathrm{oots}_T = \{$`l1,r1`$\}$.

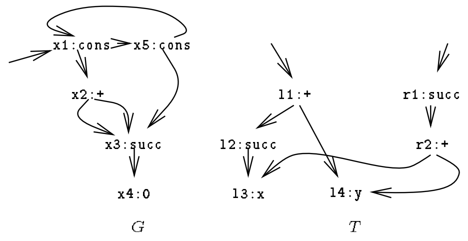

Figure 2: Two examples of graphs.

Definition 3 (Paths)
Let g be a graph and n_1 and n_2 two nodes of g. We say that n_2 is a *successor* of n_1 if $\mathcal{S}_g(n_1) = v_1 \ldots v_k$ and there exists a $j \in 1..k$ such that $v_j = n_2$. A *path* C from a node n_1 to a node n_2 in

a graph g is a sequence of nodes of g, $C = [u_0, u_1, \ldots, u_k]$, such that : $u_0 = n_1$, $u_k = n_2$, $k \leq 1$, $u_i \in \mathcal{N}_g$ for all $i \in 0..k$, and u_i is a successor of u_{i-1} for all $i \in 1..k$.
We denote by $\mathcal{P}_g(n_1, n_2)$ the (possibly empty) set of paths from n_1 to n_2 in the graph g. We say that a node n_2 is *accessible* from a node n_1 in a graph g if there exists a path from n_1 to n_2 in g, i.e., if $\mathcal{P}_g(n_1, n_2) \neq \emptyset$.
By definition of a graph g, the connectivity condition says that each node is either a root or is accessible from a root : $\forall n \in \mathcal{N}_g, n \notin \mathcal{R}\text{oots}_g \Rightarrow \exists r \in \mathcal{R}\text{oots}_g$ such that $\mathcal{P}_g(r, n) \neq \emptyset$. □

Drawings and the formal definition of graphs are not always useful to give examples. Therefore, we define a linear notation.

Definition 4 (Linear notation for graphs)
In the following grammar, the variable A (resp. n) ranges over the set $\Omega \cup \mathcal{X}$ (resp. $\mathcal{N}$). The linear notation of a graph is defined by :
Graph ::= Node | Node + Graph
Node ::= n:A(Node,...,Node) | n
If a graph is defined with a linear expression, its roots are always composed of the first node appearing in the expression and the nodes appearing just after a +. □

Example 2 With this syntax, the graphs G and T of figure 2 are written :
G = `x1:cons(x2:+(x3:succ(x4:0),x3),x5:cons(x3,x1))`, T = `l1:+(l2:succ(l3:x),l4:y) + r1:succ(r2:+(l3,l4))`.

2.2 Subgraph, sum of graphs and graph replacement

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 inaccessible nodes from p in g.

Definition 5 (Subgraph)
Let g be a graph and P a set of nodes of g. We define the *subgraph of g rooted by P*, denoted $g_{|P}$, as the following graph :

- $\mathcal{N}_{g_{|P}}$ is the set of nodes of g accessible from nodes of P, i.e., $\mathcal{N}_{g_{|P}} = P \cup \{n \in \mathcal{N}_g \mid \exists r \in P, \mathcal{P}_g(r, n) \neq \emptyset\}$.
- For all nodes n of $g_{|P}$, $\mathcal{L}_{g_{|P}}(n) = \mathcal{L}_g(n)$, i.e., $\mathcal{L}_{g_{|P}}$ is the restriction of $\mathcal{L}_g$ to $\mathcal{N}_{g_{|P}}$.
- For all nodes n of $g_{|P}$, $\mathcal{S}_{g_{|P}}(n) = \mathcal{S}_g(n)$, i.e., $\mathcal{S}_{g_{|P}}$ is the restriction of $\mathcal{S}_g$ to $\mathcal{N}_{g_{|P}}$.
- $\mathcal{R}\text{oots}_{g_{|P}} = P$.

Given two graphs g_1 and g_2, we say that g_2 is a *subgraph* of g_1 if there exists a set P of nodes of g_1 such that $g_2 = g_{1|P}$. Finally, we abbreviate $g_{1|\{n\}}$ by $g_{1|n}$. □

Example 3 In figure 2, the subgraph of G rooted by `x2` is $G_{|\texttt{x2}}$ = `x2:+(x3:succ(x4:0),x3)` and the subgraph of T rooted by {`l3`,`l4`} is $T_{|\{\texttt{l3},\texttt{l4}\}}$ = `l3:x + l4:y`.

The notion of replacement of a one-rooted subgraph by another term graph in a graph is essential for graph rewriting. Figure 3 shows the different stages of the replacement by D, whose root is `r1`, of the subgraph of G rooted by `x2`. First, as G and D share nodes (`x3` and `x4`), we must not consider G and D as two different graphs but as one graph with two roots (H_1 in fig. 3). This operation is called the *sum of graphs G and D*. Second, we redirect all arrows pointing on `x2` to point on `r1` (H_2). This operation is called *pointer redirection*. Notice that `x2` became a root.

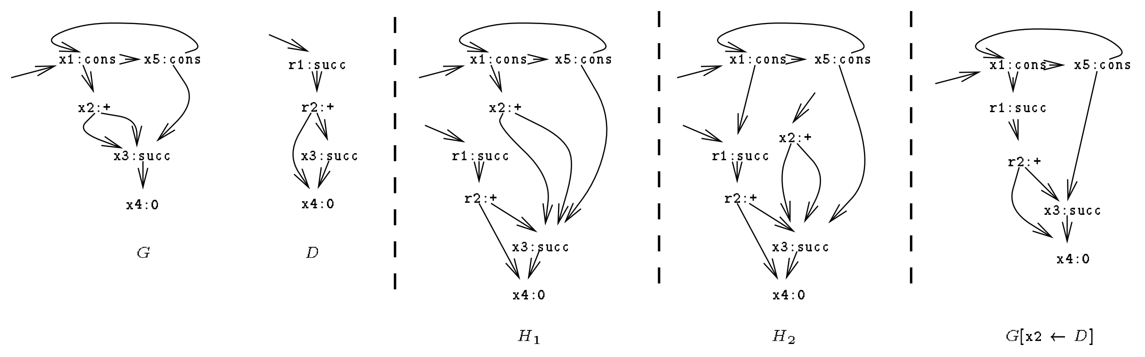

Figure 3: Calculus of a replacement.

Otherwise, the graph obtained after pointer redirection would not be connected. Finally, we ignore all uninteresting nodes such as `x2` (garbage collection). The result of *the replacement by D of the subgraph of G at node* `x2` is denoted $G[\texttt{x2} \leftarrow D]$.

Formally, we first have to define the sum of graphs, which formalizes the fact that given two graphs, we can consider them just as one, doing the union of their nodes and of their roots. Without caution, the result of this calculus is not always a graph. This is the reason of the following technical definition.

Definition 6 (Compatibility between two graphs)
Let g_1 and g_2 be two graphs. We say that g_1 and g_2 are *compatible* if :

1. Every node appearing both in g_1 and g_2 has the same label and the same successors : For all $n \in \mathcal{N}_{g_1} \cap \mathcal{N}_{g_2}$, $\mathcal{L}_{g_1}(n) = \mathcal{L}_{g_2}(n)$ and $\mathcal{S}_{g_1}(n) = \mathcal{S}_{g_2}(n)$.
2. Any variable appearing both in g_1 and g_2 labels a same node : For all $x \in \mathcal{V}(g_1) \cap \mathcal{V}(g_2)$, if $\mathcal{L}_{g_1}(n) = x$ and $\mathcal{L}_{g_2}(m) = x$, then $n = m$.

□

Example 4 Consider the graphs g_1, g_2 and g_3 defined by $g_1 =$ `x1:f(x2:0)`, $g_2 =$ `x2:f(x1:0)` and $g_3 =$ `x3:0`. The node `x1` is labeled with `f` in g_1 whereas it is labeled with `0` in g_2. Hence, g_1 and g_2 are not compatible. The reader may check that g_1 is compatible with g_3, g_3 is compatible with g_2. From this example, we notice that the property of compatibility is not transitive.

Definition 7 (Sum of compatible graphs)
Given two compatible graphs g_1 and g_2, by the *sum*, $g_1 + g_2$, we mean the graph such that :

- $\mathcal{N}_{g_1+g_2} = \mathcal{N}_{g_1} \cup \mathcal{N}_{g_2}$.
- $\mathcal{L}_{g_1+g_2} = \mathcal{L}_{g_1} \cup^1 \mathcal{L}_{g_2}$.
- $\mathcal{S}_{g_1+g_2} = \mathcal{S}_{g_1} \cup^1 \mathcal{S}_{g_2}$.
- $\mathcal{R}\mathrm{oots}_{g_1+g_2} = \mathcal{R}\mathrm{oots}_{g_1} \cup \mathcal{R}\mathrm{oots}_{g_2}$.

□

[1]The union of functions f and g, $f \cup g$ is to be considered as the union of the relations induced by these functions. In our case $f \cup g$ is still a function.

It is clear that a sum of graphs is a graph : the first condition of compatibility insures $\mathcal{L}_{g_1+g_2}$ and $\mathcal{S}_{g_1+g_2}$ are functions and the second condition insures a variable can label at most one node in the sum of graphs.

We define below the notion of pointer redirection, which consists in substituting a node by another one in a graph.

Definition 8 (Pointer redirection)
Let g be a graph and p , q two (not necessarily distinct) nodes of the same sort. The *pointer redirection from p to q in g* is a function $\rho : \mathcal{N}_g \to \mathcal{N}_g$ such that $\rho(p) = q$ and $\rho(n) = n$ for all nodes $n \neq p$.

We define the graph $\rho(g)$ obtained by pointer redirection ρ as the graph $\rho(g)$ such that all nodes having p as successor in g have q as successor in $\rho(g)$:

- $\mathcal{N}_{\rho(g)} = \mathcal{N}_g$.
- $\mathcal{L}_{\rho(g)} = \mathcal{L}_g$.
- For all nodes n, if $\mathcal{S}_g(n) = \varepsilon$, then $\mathcal{S}_{\rho(g)}(n) = \varepsilon$, else, if $\mathcal{S}_g(n) = n_1 \dots n_k$, then $\mathcal{S}_{\rho(g)}(n) = \rho(n_1) \dots \rho(n_k)$.
- $\mathcal{R}\text{oots}_{\rho(g)} = \{p\} \cup \rho(\mathcal{R}\text{oots}_g)$, that is to say if $\mathcal{R}\text{oots}_g = \{n_1, \dots, n_k\}$, then $\mathcal{R}\text{oots}_{\rho(g)} = \{p, \rho(n_1), \dots, \rho(n_k)\}$.

□

It is clear that $\rho(g)$ is well defined : all conditions but connectivity are inherited from those of g ; connectivity is warranted by the fact that $p \in \mathcal{R}\text{oots}_{\rho(g)}$. When $p \notin \mathcal{N}_g$, $\rho(g) = g$.

We are now ready to define the notion of replacement.

Definition 9 (Replacement)
Let g be a graph, u a term graph, such that g and u are compatible, and p a node of the same sort as $\mathcal{R}\text{oot}_u$. If p is a node of g, we define the *replacement by u of the subgraph rooted by p in g*, denoted $g[p \leftarrow u]$, in three stages :

1. Let $H = g + u$.
2. Let ρ be the pointer redirection, defined on H, such that $\rho(p) = \mathcal{R}\text{oot}_u$ and $\rho(n) = n$ for all $n \neq p$. Let $H' = \rho(H)$ and $Q = \rho(\mathcal{R}\text{oots}_g)$.
3. $g[p \leftarrow u] = H'_{|Q}$.

If p is not a node of g, then $g[p \leftarrow u] = g$. □

Notice that if g is a graph and n is a node of g, then $g[n \leftarrow g_{|n}] = g$.

Example 5 We detailed in figure 3 the replacement by
D = `r1:succ(r2:+(x4:0,x3:succ(x4)))` of the subgraph rooted by `x2` in the graph G = `x1:cons(x2:+(x3:succ(x4:0),x3),x5:cons(x3,x1))`. The result is the graph $G[$`x2` $\leftarrow D] =$ `x1:cons(r1:succ(r2:+(x4:0,x3:succ(x4))),x5:cons(x3,x1))`.

The calculus of $g[p \leftarrow d]$ may have surprising results in some cases.

Example 6 Let g = `l1:+(l2:0,l3:0)`, d = `r1:+(l1:+(l2:0,l3:0),r2:0)`, and p = `l1`. Then the reader may check that $g[p \leftarrow d]$ = `r1:+(r1,r2:0)`, whereas he could expect to obtain $g[p \leftarrow d] = d$. d is modified in $g[p \leftarrow d]$ because p is a common node to g and d.

In the rest of this section, we give some technical properties concerning subgraphs and replacement that we use later in the proofs. These properties have first been established in the framework of first-order terms. Their use in our setting needs however some caution (cf. example 6). The vocabulary used is mostly borrowed from [Hue80].

We first introduce new notations in order to compare the relative positions of two nodes in a graph. Let n_1, n_2 be two nodes of a graph g. We write $n_1 \prec n_2$ iff n_2 is accessible from n_1 in g, i.e., $\mathcal{P}_g(n_1, n_2) \neq \emptyset$. We write $n_1 \preceq n_2$ iff $n_1 \prec n_2$ or $n_1 = n_2$. We say n_1 and n_2 are *disjoint*, denoted $n_1 \| n_2$, iff there exists no path from n_1 to n_2 nor from n_2 to n_1 in g, i.e., $\neg(n_1 \preceq n_2 \vee n_2 \preceq n_1)$, or equivalently, $\mathcal{P}_g(n_1, n_2) = \mathcal{P}_g(n_2, n_1) = \emptyset$. Notice that $\preceq$ is not a partial order, but just a preorder.

Proposition 1 (Double subgraph)
Let g be a graph and n_1 and n_2 two nodes of g. If $n_1 \preceq n_2$, then $(g_{|n_1})_{|n_2} = g_{|n_2}$.

Proposition 2 (Persistence)
Let g be a graph, u a term graph compatible with g, n_1 and n_2 two nodes of g. If there is no path from n_2 to n_1 in g and there exists a path from a root of g to n_2 which does not go through n_1, then $(g[n_1 \leftarrow u])_{|n_2} = g_{|n_2}$. This proposition holds in particular when $n_1 \| n_2$.

Proposition 3 (Embedding)
Let g be a graph, u a term graph compatible with g, n_1 and n_2 two nodes of g. If $n_1 \notin \mathcal{N}_u$ and $n_2 \in \mathcal{N}_u$, then $(g[n_1 \leftarrow u])_{|n_2} = u_{|n_2}$.

From the example 6, it appears that an expression such as $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2]$ may be not well defined. Indeed, suppose that g is a graph compatible with two term graphs u_1 and u_2. Since g is compatible with u_1, $g[n_1 \leftarrow u_1]$ is computable. However, this graph may be no more compatible with u_2.

Example 7 Let $g =$ `x1:f(x2:g(x3:0))`, $u_1 =$ `x4:1` and $u_2 =$ `x5:h(x2:g(x3:0))` be three compatible term graphs. Then $g[$`x3` $\leftarrow u_1] =$ `x1:f(x2:g(x4:1))`. This graph is no more compatible with u_2, since `x2` has not the same successors in $g[$`x3` $\leftarrow u_1]$ as in u_2. The problem is that `x3` appears both in g and u_2.

The following proposition gives a sufficient condition to avoid the above problem.

Proposition 4 Let g be a graph, n_1 and n_2 two nodes, u_1 and u_2 two term graphs such that g, u_1 and u_2 are compatible. If $n_1 \notin \mathcal{N}_{u_2}$, then $g[n_1 \leftarrow u_1]$ and u_2 are compatible, thus $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2]$ is well defined.

Proof Assume that $g[n_1 \leftarrow u_1]$ and u_2 are not compatible, whereas g, u_1 and u_2 are compatible. By definition of compatibility, there must exist a node p common to $g[n_1 \leftarrow u_1]$ and u_2 such that $\mathcal{S}_{g+u_1}(p) = \mathcal{S}_{u_2}(p)$ but $\mathcal{S}_{g[n_1 \leftarrow u_1]}(p) \neq \mathcal{S}_{u_2}(p)$. Suppose $\mathcal{S}_{g+u_1}(p) = p_1 \ldots p_k$. Let ρ be the pointer redirection such that $\rho(n_1) = \mathcal{R}\mathrm{oot}_{u_1}$ and $\rho(n) = n$ for all $n \neq n_1$. Then, by definition of replacement, $\mathcal{S}_{g[n_1 \leftarrow u_1]}(p) = \rho(p_1) \ldots \rho(p_k)$. Since $\mathcal{S}_{u_2}(p) = p_1 \ldots p_k$ and $\mathcal{S}_{g[n_1 \leftarrow u_1]}(p) \neq \mathcal{S}_{u_2}(p)$, there exists $i \in 1..k$ such that $\rho(p_i) \neq p_i$. Hence, there exists $i \in 1..k$ such that $p_i = n_1$. Since $\mathcal{S}_{u_2}(p) = p_1 \ldots p_k$, then $n_1 \in \mathcal{N}_{u_2}$. Thus, if n_1 is not a node of u_2, then $g[n_1 \leftarrow u_1]$ and u_2 are compatible.

□

Proposition 5 (Associativity)
Let g be a graph, u_1 and u_2 two term graphs such that g, u_1 and u_2 are compatible, n_1 and n_2 two nodes of g such that $n_1 \notin \mathcal{N}_{u_2}$. Then $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = (g[n_2 \leftarrow u_2])[n_1 \leftarrow (u_1[n_2 \leftarrow u_2])]$.

Proof Hint : the proof is done in three stages :

1. Let $\rho_1(n_1) = \mathcal{R}\mathrm{oot}_{u_1}$ and $\rho_2(n_2) = \mathcal{R}\mathrm{oot}_{u_2}$ be two pointer redirections. Since $n_1 \notin \mathcal{N}_{u_2}$, $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2]$ is well defined. Then, we can show that
$(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = (\rho_2(\rho_1(g + u_1 + u_2)))_{|\rho_2(\rho_1(\mathcal{R}\mathrm{oots}_g))}$.

2. Let $\rho_3(n_1) = \rho_2(\mathcal{R}\mathrm{oot}_{u_1})$ be a new pointer redirection. $(g[n_2 \leftarrow u_2])[n_1 \leftarrow (u_1[n_2 \leftarrow u_2])]$ is well defined because if g and u_1 are compatible, then $g[n_2 \leftarrow u_2]$ and $u_1[n_2 \leftarrow u_2]$ remain compatible. Then we can show that
$(g[n_2 \leftarrow u_2])[n_1 \leftarrow u_1[n_2 \leftarrow u_2]] = (\rho_3(\rho_2(g + u_1 + u_2)))_{|\rho_3(\rho_2(\mathcal{R}\mathrm{oots}_g))}$.

3. Finally, we can show that for all $p \in \mathcal{N}_{g+u_1+u_2}$, $\rho_2(\rho_1(p)) = \rho_3(\rho_2(p))$. This proof is done by analysis of the different cases for p.

□

Proposition 6 (Commutativity)
Let g be a graph, u_1 and u_2 two term graphs such that g, u_1 and u_2 are compatible, n_1 and n_2 two nodes of g such that $n_1 \notin \mathcal{N}_{u_2}$ and $n_2 \notin \mathcal{N}_{u_1}$. Then $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = (g[n_2 \leftarrow u_2])[n_1 \leftarrow u_1]$.

Proof Follows from Proposition 5 by taking into account the fact that n_2 is not a node of u_1. □

Proposition 7 (Weak dominance)
Let g be a graph, u_1 and u_2 two term graphs such that g, u_1, u_2 are compatible and n_1 and n_2 two nodes of g such that $n_1 \notin \mathcal{N}_{u_2}$ and every path from a root of g to n_1 go through n_2, i.e., $\forall r \in \mathcal{R}\mathrm{oots}_g, \forall C \in \mathcal{P}_g(r, n_1), n_2 \in C$. Then, $(g[n_1 \leftarrow u_1])[n_2 \leftarrow u_2] = g[n_2 \leftarrow u_2]$.

Proof Hint : Prove that with these hypothesis, n_1 is not a node of $g[n_2 \leftarrow u_2]$. Then simplify the expression of associativity. □

2.3 Homomorphisms

We now recall the notions about homomorphisms. They play almost the same rôle as do substitutions for first-order terms.

Definition 10 (Homomorphism)
Given two graphs g_1 and g_2, by a *(rooted) homomorphism* $h : g_1 \to g_2$ from g_1 to g_2, we mean any mapping $h = \uplus_{s \in S} h_s$, $h_s : (\mathcal{N}_{g_1})_s \to (\mathcal{N}_{g_2})_s$ such that :

1. For every node of g_1 which is not labeled with a variable, h preserves the labeling and the successor function : For all nodes $n \in (\mathcal{N}_{g_1})_s$, $(\mathcal{L}_g)_s(n) \notin \mathcal{X}$,
   - $(\mathcal{L}_{g_2})_s(h_s(n)) = (\mathcal{L}_{g_1})_s(n)$.
   - $(\mathcal{S}_{g_2})_s(h_s(n)) = h_{s_1 \ldots s_k}((\mathcal{S}_{g_1})_s(n))$, that is to say for all non variable nodes n, if $(\mathcal{S}_{g_1})_s(n) = \varepsilon$, then $(\mathcal{S}_{g_2})_s(h_s(n)) = \varepsilon$, else if $(\mathcal{S}_{g_1})_s(n) = n_1 \ldots n_k$, where $n_i \in (\mathcal{N}_g)_{s_i}$ for all $i \in 1..k$, then $(\mathcal{S}_{g_2})_s(h_s(n)) = h_{s_1}(n_1) \ldots h_{s_k}(n_k)$.

2. For all nodes $n \in (\mathcal{N}_{g_1})_s$, $(\mathcal{L}_g)_s(n) \in \mathcal{X}$, $h_s(n) \in (\mathcal{N}_{g_2})_s$.

3. The images of roots of g_1 are the roots of g_2 : $\mathcal{R}\mathrm{oots}_{g_2} = h(\mathcal{R}\mathrm{oots}_{g_1})$, i.e., if $\mathcal{R}\mathrm{oots}_{g_1} = \{n_1, \ldots, n_k\}$, where $n_i \in (\mathcal{N}_g)_{s_i}$ for all $i \in 1..k$, then $\mathcal{R}\mathrm{oots}_{g_2} = \{h_{s_i}(n_i) \mid i \in 1..k\}$.

We say that h is a *variable-preserving homomorphism* if h is a homomorphism between two graphs with variables treated as constants, i.e., $\mathcal{L}_{g_2}(h(n)) = \mathcal{L}_{g_1}(n)$ for every node n of g_1, *in particular for every variable node n of g_1.* □

Example 8 In figure 4, we give an example of homomorphism h from a graph
L = `l1:+(l2:succ(l3:x),l4:y)` to the subgraph of G rooted by `x2`. Formally, h is a mapping from $\mathcal{N}_L$ to $\mathcal{N}_{(G_{|\mathtt{x2}})}$ defined by $h(\mathtt{l1}) = \mathtt{x2}$, $h(\mathtt{l2}) = h(\mathtt{l4}) = \mathtt{x3}$ and $h(\mathtt{l3}) = \mathtt{x4}$. h is not a variable-preserving homomorphism since $\mathcal{L}_L(\mathtt{l3}) = \mathtt{x}$ whereas $\mathcal{L}_G(h(\mathtt{l3})) = \mathcal{L}_G(\mathtt{x4}) \neq \mathtt{x}$.

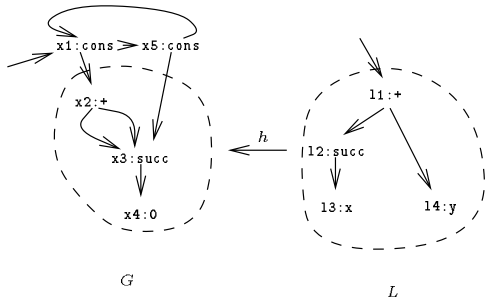

Figure 4: Example of homomorphism.

Given three graphs g_1, g_2 and g_3 and two homomorphisms $h_1 : g_1 \to g_2$ and $h_2 : g_2 \to g_3$, by the *composition* of h_1 and h_2, denoted $h_2 \circ h_1$, we mean the mapping $h : \mathcal{N}_{g_1} \to \mathcal{N}_{g_3}$, such that $h(n) = h_2(h_1(n))$ for all $n \in \mathcal{N}_{g_1}$. It is clear that $h_2 \circ h_1$ is a homomorphism from g_1 to g_3. Finally, given a graph g, the identity function on $\mathcal{N}_g$ is a homomorphism from g to itself. An *isomorphism* is a bijective homomorphism.

Definition 11 (Restriction of a homomorphism)
Let g_1 and g_2 be two graphs, $h : g_1 \to g_2$ a homomorphism, and P a set of nodes of g_1. Let $h_{|P}$ be the restriction of h to $\mathcal{N}_{g_1|P}$ (h is seen as a function between nodes). $h_{|P}$, which remains a homomorphism from $g_{1|P}$ to $g_{2|h(P)}$, is called the *restriction of h to $g_{1|P}$*. □

Matching is essential for graph rewriting. An algorithm is given in appendix A.

Definition 12 (Matching)
Given a graph g, a term graph l and a node n of g, we say that l *matches g at node n*, denoted $l \leq g_{|n}$, if there exists a homomorphism $h : l \to g_{|n}$. h is called the *matcher* of l on g at node n.
□

Example 9 The homomorphism h of figure 4 is a matcher of L on G at node `x2`.

Finally, a precise definition of a rewriting step requires the notion of "application" of a homomorphism $h : g_1 \to g_2$ on a graph g, denoted $h(g)$. This process corresponds to the application of the extension of a substitution on a first-order term. Informally, computing $h(g)$ consists in replacing all shared subgraphs between g and g_1 by their corresponding subgraphs in g_2. This is done by pointer redirection.

Example 10 Consider the homomorphism $h : L \to G_{|\mathtt{x2}}$ of example 8. Let
R = `r1:succ(r2:+(l3:x,l4:y))`. Computing $h(R)$ consists in replacing all subgraphs shared

between R and L by their corresponding subgraphs in $G_{|\texttt{x2}}$. Here, R and L share subgraphs `13:x` and `14:y`. Their corresponding graphs by h in G are respectively `x4:0` and `x3:succ(x4:0)`. Hence, we obtain a graph $h(R)$ = `r1:succ(r2:+(x4:0,x3:succ(x4)))`, i.e., the graph D of figure 3.

We now give a formal definition of $h(g)$. First, we define the border between two graphs, i.e., the set of nodes where we shall execute the replacements.

Definition 13 (Border)
Let g and g' be two compatible graphs. We call *border of g with respect to g'* the set $\mathcal{B}(g,g') = \{p \in \mathcal{N}_g \cap \mathcal{N}_{g'} \mid p \in \mathcal{R}\text{oots}_g \text{ or } \exists q \in (\mathcal{N}_g - \mathcal{N}_{g'}), p \in \mathcal{S}_g(q)\}$. □

Example 11 Consider the compatible graphs g = `x0:f(x1:+(x2:x,x3:y),x3)` and g_1 = `x1:+(x2:x,x3:y)`. Then the border of g w.r.t. g_1 is the set $\mathcal{B}(g,g_1) = \{$`x1,x3`$\}$.

Definition 14 (Application of a homomorphism on a graph)
Let g, g_1 and g_2 be three *compatible* graphs, $h : g_1 \to g_2$ a homomorphism and $B = \{p_1, \ldots, p_k\}$ the border of g w.r.t. g_1. We suppose that $B \cap \mathcal{N}_{g_2} = \emptyset$. The *application of h on g*, denoted $h(g)$, is defined by :

$$h(g) = g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_k \leftarrow g_{2|h(p_k)}]$$

Thanks to proposition 6, the condition $B \cap \mathcal{N}_{g_2} = \emptyset$ is sufficient to insure the well definedness of the expression $g[p_1 \leftarrow g_{2|h(p_1)}] \ldots [p_k \leftarrow g_{2|h(p_k)}]$. □

Example 12 Consider the three compatible graphs g = `x0:f(x1:+(x2:x,x3:y),x3)`, g_1 = `x1:+(x2:x,x3:y)` and g_2 = `y1:+(y2:0,y2)`, and the homomorphism h from the graph g_1 to the graph g_2. The reader may check that $\mathcal{B}(g,g_1) \cap \mathcal{N}_{g_2} = \emptyset$ and $h(g)$ = `x0:f(y1:+(y2:0,y2),y2)`.

2.4 Graph equalities

Homomorphisms are also used to define *equality relations* on graphs. We already used the *syntactic equality*, denoted $=$, between graphs. Hereafter, we define two other equalities.

Definition 15 (Equality up to renaming of nodes)
Given two graphs g_1 and g_2, we say that g_1 *is equal to g_2 up to renaming of nodes* if there exists a variable-preserving isomorphism $\phi : g_1 \to g_2$ between g_1 and g_2. We write $g_1 \sim_\phi g_2$ or simply $g_1 \sim g_2$. □

With this notion of renaming, one can change nodes and variable nodes of a graph, but not variables names : the isomorphism is variable-preserving. Equality up to renaming of nodes is similar to α-conversion in λ-calculus.

We now define bisimilarity. Informally, two term graphs are bisimilar if they represent the same infinite tree when one unravels them. Bisimilarity is called *tree-equivalence* in [BvEG⁺87]. Bisimilarity between term graphs without variable is studied in [AK96].

Definition 16 (Bisimilarity)
Let g_1 and g_2 be two term graphs. We say that g_1 and g_2 are *bisimilar*, denoted $g_1 \doteq g_2$, if there exists a graph g', and two *variable-preserving* homomorphisms $h_1 : g_1 \to g'$ and $h_2 : g_2 \to g'$. □

Example 13 In figure 5, the left-hand and right-hand graphs are bisimilar, since there exists a variable-preserving homomorphism from each of them to the middle graph.

Remark : In the definition of graph bisimilarity, h_1 and h_2 are variable-preserving homomorphisms. Therefore, two graphs with different sets of variables cannot be bisimilar. Bisimilarity can be defined by another way : one can prove that two bisimilar term graphs represent the same (infinite) tree when one unravels them [AK96].

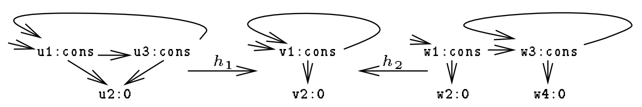

Figure 5: Three bisimilar graphs.

3 Constructor-based graph rewriting systems (cGRS)

This section introduces the class of GRSs we consider. For practical reasons, many declarative languages use constructor-based signatures.

Definition 17 (Constructor-based signature)
By a *constructor-based signature* Σ, we mean a triple $\Sigma = \langle S, \mathcal{C}, \mathcal{D} \rangle$ such that S is a set of sorts, $\mathcal{C}$ is an S-indexed family of sets of *constructor symbols*, $\mathcal{D}$ is an S-indexed family of sets of *defined operations*, $\mathcal{C} \cap \mathcal{D} = \emptyset$ and $\langle S, \mathcal{C} \uplus \mathcal{D} \rangle$ is a signature. □

Example 14 Figure 6 shows a constructor-based signature defining *(cyclic) lists of naturals.*

```
signature list-of-nat
sorts
   list nat
constructors
   0 :   → nat
   succ :  nat → nat
   nil :  → list
   cons :  nat list → list
operations
   + :  nat nat → nat
   * :  nat nat → nat
   car :  list → nat
   cdr :  list → list
end signature
```

Figure 6: `list-of-nat` signature.

A graph g is called *constructor* if all its nodes are labeled either with variables or with constructor symbols. A term graph is said *operation-rooted* (resp. *constructor-rooted*) if its root is labeled with a defined operation (resp. constructor symbol).

In the rest of the paper, we investigate graph rewriting for the class of what we call *admissible* graphs. Roughly speaking, an admissible graph corresponds to nested procedures (functions) calls whose parameters are complex constructor graphs.

Definition 18 (Admissible graph)
A graph g is *admissible* iff for all $n \in \mathcal{N}_g$, if $\mathcal{L}_g(n) \in \mathcal{D}$, then $\mathcal{P}_g(n, n) = \emptyset$. □

Example 15 The graphs G and T of figure 2 are admissible, whereas the graphs `z:+(z,z)` and `z:cdr(cons(0,z))` are not.

Definition 19 (Rewrite rule)
A *rewrite rule* is a graph e with two roots, denoted $l \to r$. l (resp. r) is a term graph called the *left-hand side* (resp. the *right-hand side*) of the rule. l and r are compatible and $e = l + r$. We shall always assume that $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. We say that a rule e' is a variant of the rule e if e and e' are isomorphic and all nodes and variables of e' are new. □

From the definition of graphs, variables occurring in a rewrite rule are always shared between the left-hand side and the right-hand side. The graph T of figure 2 shows an example of rules.

In this paper, we consider particular rewrite rules called *admissible rewrite rules*. Such rules are tailored so that the set of admissible graphs is stable by rewriting (see Example 19 and Proposition 9). In order to define these rewrite rules, we introduce below the notion of pattern.

Definition 20 (Pattern)
A term graph g is a *pattern* if :

1. It is operation-rooted : $\mathcal{L}_g(\mathcal{R}\mathrm{oot}_g) \in \mathcal{D}$.
2. All its other nodes are labeled either with a constructor symbol or with a variable : For all $n \in \mathcal{N}_g$, if $n \neq \mathcal{R}\mathrm{oot}_g$, then $\mathcal{L}_g(n) \in \mathcal{X}$ or $\mathcal{L}_g(n) \in \mathcal{C}$.
3. There is no cycle including the root : $\mathcal{P}_g(\mathcal{R}\mathrm{oot}_g, \mathcal{R}\mathrm{oot}_g) = \emptyset$.
4. For all nodes n of g different from the root, there exists one and only one node m in g such that n is a successor of m : for all $n \in \mathcal{N}_g$, if $n \neq \mathcal{R}\mathrm{oot}_g$, then $cardinal(\{m \in \mathcal{N}_g \mid n \in \mathcal{S}_g(m)\}) = 1$.

In other words, a pattern is a term graph corresponding to a linear first-order term (tree) which contains only one occurrence of defined operation situated at its root. □

Example 16 `l1:+(l2:0,l3:0)` is a pattern, whereas `l1:+(l2:0,l2)` is not.

Definition 21 (Admissible rewrite rule)
An *admissible rewrite rule* is a rewrite rule $e = l \to r$ such that :

1. l is a *pattern* (thus an admissible term graph).
2. r is an *admissible* term graph.
3. l is not a subgraph of r.
4. $\mathcal{V}(r) \subseteq \mathcal{V}(l)$.

□

We say that two admissible rewrite rules are *non-overlapping* if their left-hand sides are not unifiable. This notion of unification is straightforward : it is the same as that for first-order terms, since left-hand sides are patterns.

Definition 22 (cGRS)
A *constructor-based graph rewriting system (cGRS)* is a pair $SP = \langle \Sigma, \mathcal{R} \rangle$ such that $\Sigma = \langle S, \mathcal{C}, \mathcal{D} \rangle$ is a constructor-based signature and $\mathcal{R}$ is a set of admissible rewrite rules such that every two distinct rules in $\mathcal{R}$ are non-overlapping. □

Example 17 Figure 7 shows the set of admissible rewrite rules associated to the `list-of-nat` signature of figure 6.

Rewriting a graph is done in two stages. First, one computes the matcher of the left-hand side of a rule on a subgraph of the graph to be rewritten and second, the subgraph is replaced by the instantiated right-hand side of the rule. This definition is consistent with [BvEG+87].

```
rules list-of-nat
   0 + l1:y → l1
   succ(l1:x) + l2:y → succ(l1 + l2)
   l1:0 * x → l1
   succ(l1:x) * l2:y → (l1 * l2) + l1
   car( cons(l1:x,l2:l) ) → l1
   cdr( cons(l1:x,l2:l) ) → l2
end rules
```

Figure 7: `Rules for list-of-nat`.

Definition 23 (Rewriting step)
Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. Let g_1 be an admissible graph, g_2 a graph, $e = l \rightarrow r$ 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* $l \rightarrow r$, denoted $g_1 \rightarrow_{[p, l \rightarrow r]} g_2$, if :

1. l matches g_1 at node p (i.e., $l \leq g_{1|p}$) : there exists a homomorphism $h : l \rightarrow g_{1|p}$.

2. $g_2 = g_1[p \leftarrow h(r)]$.

In this case, we say $g_{1|p}$ is a *redex* of g_1 rooted by p. A graph is in *normal form* if it has no redex. A constructor graph is necessarily in normal form. $\overset{\varepsilon}{\rightarrow}$ denotes the reflexive closure of $\rightarrow$ and we say that g_1 is rewritten *in at most one step*[2] into g_2 whenever $g_1 \overset{\varepsilon}{\rightarrow} g_2$. $\overset{+}{\rightarrow}$ denotes the transitive closure of $\rightarrow$ and $\overset{*}{\rightarrow}$, its transitive and reflexive closure. We speak of a *derivation* from g_1 to g_2 whenever $g_1 \overset{*}{\rightarrow} g_2$. $\square$

Example 18 Consider the graph G and the rule $T = L \rightarrow R$ of figure 2, where L = `l1:+(l2:succ(l3:x,l4:y))` and R = `r1:succ(r2:+(l3:x,l4:y))`. According to figure 4, L matches G at node `x2` with homomorphism h. We saw in example 10 that $h(R)$ = `r1:succ(r2:+(x4:0,x3:succ(x4)))`. Finally, thanks to the calculus presented in figure 3, we have $G[$`x2` $\leftarrow h(R)]$ = `x1:cons(r1:succ(r2:+(x4:0,x3:succ(x4))),x5:cons(x3,x1))`. Let G' be this last graph. Then, by definition, $G \rightarrow_{[\mathtt{x2}, L \rightarrow R]} G'$.

We now prove the well definedness of a rewriting step.

Proposition 8 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g_1 an admissible graph, $e = l \rightarrow r$ a variant of a rewrite rule of $\mathcal{R}$ and p be a node of g_1. If l matches g_1 at node p, then there exists a graph g_2 such that $g_1 \rightarrow_{[p, l \rightarrow r]} g_2$.

Proof Let $h : l \rightarrow g_{1|p}$ be the matcher. We must prove that the expression $g_1[p \leftarrow h(r)]$ is well defined. By hypothesis, l, r and $g_{1|p}$ are compatible because l and r are compatible, $\mathcal{N}_{l \rightarrow r} \cap \mathcal{N}_{g_1} = \emptyset$ and $\mathcal{V}(l \rightarrow r) \cap \mathcal{V}(g_1) = \emptyset$. Moreover, $\mathcal{B}(l, r) \subseteq \mathcal{N}_{l \rightarrow r}$ so $\mathcal{B}(l, r) \cap \mathcal{N}_{(g_{1|p})} = \emptyset$. Hence $h(r)$ is well defined. Let $\mathcal{B}(l, r) = \{p_1, \ldots, p_k\}$. By the definition of the application of a homomorphism to a graph, $h(r) = r[p_1 \leftarrow (g_{1|p})_{|h(p_1)}] \ldots [p_k \leftarrow (g_{1|p})_{|h(p_k)}]$. This last graph is composed of nodes of r and of subgraphs of g_1, which are not modified by the pointer redirections (since $p_i \notin \mathcal{N}_{g_1}$ for all $i \in 1..k$). So $h(r)$ is compatible with g_1, i.e., $g_1[p \leftarrow h(r)]$ is well defined. $\square$

Now, we show the stability of the set of admissible graphs w.r.t. rewriting with admissible rules. The following lemma gives a sufficient condition which ensures the stability of admissible graphs by replacement.

[2]This notation is introduced in [Hue80]. In [Klo92], it is written $\rightarrow^{\equiv}$.

Lemma 1 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g an admissible graph, u an admissible term graph compatible with g and $p \in \mathcal{N}_g$. If $p \notin \mathcal{N}_u$, then $g[p \leftarrow u]$ is an admissible graph.

Proof Since $p \notin \mathcal{N}_u$, we have $\mathcal{R}\text{oot}_u \not\preceq p$ in $g + u$. Thus, the pointer redirection from p to $\mathcal{R}\text{oot}_u$ does not modify u to create a cycle on a node labeled with a defined operation. □

Proposition 9 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g_1 an admissible graph, $e = l \to r$ a variant of a rewrite rule of $\mathcal{R}$, p a node of g_1 and g_2 a graph such that $g_1 \to_{[p, l \to r]} g_2$. Then, g_2 is an admissible graph.

Proof Let $h : l \to g_{1|p}$ be the matcher from the left-hand side of the rule on g_1 at node p. Thanks to Proposition 8, we know that $g_2 = g_1[p \leftarrow h(r)]$ is well defined. Let us prove that g_2 is admissible. Thanks to lemma 1, it is sufficient to prove that $p \notin \mathcal{N}_{h(r)}$. In the proof of proposition 8, we established that if $P = \mathcal{B}(l, r) = \{p_1, \ldots, p_k\}$ then $h(r) = r[p_1 \leftarrow (g_{1|p})_{|h(p_1)}] \ldots [p_k \leftarrow (g_{1|p})_{|h(p_k)}]$. This last graph is composed of nodes of r and of subgraphs of g_1, which are not modified by the pointer redirections (since $p_i \notin \mathcal{N}_{g_1}$ for all $i \in 1..k$). p is not a node of r, since $p \in \mathcal{N}_{g_1}$ and $\mathcal{N}_{g_1} \cap \mathcal{N}_r = \emptyset$. Thus if p appears in $h(r)$, then p is a node of $(g_{1|p})_{|h(p_i)}$ for some i. As $p_i \in P$ and $P = \mathcal{B}(l, r)$, $p_i \in \mathcal{N}_l$, thus, $\mathcal{R}\text{oot}_l \preceq p_i$ in l. The matcher h being a homomorphism, we deduce that $h(\mathcal{R}\text{oot}_l) \preceq h(p_i)$ in g_1, and as $h(\mathcal{R}\text{oot}_l) = p$, $p \preceq h(p_i)$ in g_1. $\mathcal{R}\text{oot}_l$ is labeled with a defined operation (since l is a pattern) and $h(\mathcal{R}\text{oot}_l) = p$, so by definition of a homomorphism, p is also labeled with a defined operation. g_1 is an admissible graph, so there exists no path from p to itself in g_1, hence $p \preceq h(p_i)$ and $h(p_i) \not\preceq p$ in g_1. So the only possibility for p to be a node of $(g_{1|p})_{|h(p_i)}$ is that $p = h(p_i)$. $h : l \to g_{1|p}$ is a homomorphism, so $h(\mathcal{R}\text{oot}_l) = p$ and for all node $n \in \mathcal{N}_l$ such that $n \neq \mathcal{R}\text{oot}_l$, $h(n) \neq p$ (otherwise, there would be a cycle on p in g). Thus, if $h(p_i) = p$, then $p_i = \mathcal{R}\text{oot}_l$. However, $p_i \in \mathcal{B}(l, r)$ and $\mathcal{R}\text{oot}_l \notin \mathcal{N}_r$ by definition of an admissible rule, so $p_i \neq \mathcal{R}\text{oot}_l$. Thus p is not a node of $h(r)$ and by lemma 1, $g_2 = g_1[p \leftarrow h(r)]$ is admissible.

□

In the following we give a counter-example for stability when using non admissible rules.

Example 19 Consider the following rewrite rule `l1:+(l2:0,l3:x)` $\to$ `r1:+(l1,r2:0)`. This rule is not admissible since the root of its left-hand side is a node of its right-hand side and thus its left-hand side is a subgraph of its right-hand side. By using this rule, the reader may check that the admissible graph `x1:+(x2:0,x3:0)` rewrites to `r1:+(r1,r2:0)`. This last graph is not admissible, since there is a cycle on the defined operation `+`.

4 Confluence

The confluence of GRSs is actually not a straightforward extension of that of TRSs, as shown in Figure 1. In this section, we establish the confluence of the graph rewriting relation with respect to admissible graphs, in presence of an arbitrary number of collapsing rules.

Definition 24 (Confluence)
Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. We say that *the rewriting relation* $\xrightarrow{*}$ *is confluent w.r.t the class of admissible graphs* iff for all admissible graphs g_1, g_2, g_1' and g_2' such that g_1 and g_2 are equal up to renaming of nodes ($g_1 \sim g_2$), $g_1 \xrightarrow{*} g_1'$ and $g_2 \xrightarrow{*} g_2'$, there exist two admissible graphs g_1'' and g_2'' such that $g_1' \xrightarrow{*} g_1''$, $g_2' \xrightarrow{*} g_2''$ and $g_1'' \sim g_2''$. (see Figure 8). □

Remark : In figures, a continuous line represents a hypothesis, whereas a dashed line represents a conclusion.

$$\begin{tikzcd}
& g_1 \arrow[ld, "*"'] \arrow[r, phantom, "\sim"] & g_2 \arrow[rd, "*"] & \\
g_1' \arrow[rd, dashed, "*"'] & & & g_2' \arrow[ld, dashed, "*"] \\
& g_1'' \arrow[r, phantom, "\sim"] & g_2'' &
\end{tikzcd}$$

Figure 8: Confluence.

Theorem 1 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. Then the rewriting relation $\xrightarrow{*}$ is confluent w.r.t the class of admissible graphs.

In [AK96], it is proved that every GRS with non overlapping rules is confluent. But this surprising result is established for a particular rewriting relation, different from the one considered here.

The rest of this section is dedicated to the proof of Theorem 1. We must first establish a few technical results. The following proposition states that rewriting two graphs equal up to renaming of nodes induces two graphs equal up to renaming of nodes. Its proof is clear.

Proposition 10 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS and g_1 and g_2 two admissible graphs equal up to renaming of nodes ($g_1 \sim_\alpha g_2$). If there exists a graph g_1' such that $g_1 \to g_1'$ (resp. $g_1 \xrightarrow{*} g_1'$), then there exists an admissible graph g_2' such that $g_2 \to g_2'$ (resp. $g_2 \xrightarrow{*} g_2'$) and g_1' and g_2' are equal up to renaming of nodes ($g_1 \sim_\beta g_2'$). (see Figure 9).

$$\begin{tikzcd}
g_1 \arrow[r, phantom, "\sim_\alpha"] \arrow[d] & g_2 \arrow[d, dashed] & & g_1 \arrow[r, phantom, "\sim_\alpha"] \arrow[d, "*"] & g_2 \arrow[d, dashed, "*"] \\
g_1' \arrow[r, phantom, "\sim_\beta"] & g_2' & & g_1' \arrow[r, phantom, "\sim_\beta"] & g_2'
\end{tikzcd}$$

Figure 9:

The following lemma is the key result to establish confluence. It states that if a graph is rewritten in two different ways, then the two resulting graphs can be rewritten in at most one step into a same third graph (up to renaming of nodes). Notice that this lemma does not hold in the general case (see Figure 1).

Lemma 2 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS and g, g_1 and g_2 three admissible graphs. If $g \to g_1$ and $g \to g_2$, then there exist two graphs g_1' and g_2' such that $g_1 \xrightarrow{\varepsilon} g_1'$, $g_2 \xrightarrow{\varepsilon} g_2'$ and g_1' and g_2' are equal up to renaming of nodes ($g_1' \sim g_2'$). (see Figure 10).

Proof This proof is done by case analysis. Let g, g_1 and g_2 be three admissible graphs such that $g \to_{[n_1, l_1 \to r_1]} g_1$ and $g \to_{[n_2, l_2 \to r_2]} g_2$. By definition of a rewriting step, there exist a matcher $h_1 : l_1 \to g_{|n_1}$ such that $g_1 = g[n_1 \leftarrow h_1(r_1)]$ and a matcher $h_2 : l_2 \to g_{|n_2}$ such that $g_2 = g[n_2 \leftarrow h_2(r_2)]$.
If $n_1 = n_2$, then necessarily $l_1 \to r_1$ is a variant of $l_2 \to r_2$ since $\mathcal{R}$ is non-overlapping, so g_1 and g_2 are equal up to renaming of nodes. Thus, the proposition is straightforward (i.e., $g_1' = g_1$ and

$$\begin{tikzcd}
 & g \arrow[dl] \arrow[dr] & \\
g_1 \arrow[d, dashed, "\varepsilon"'] & & g_2 \arrow[d, dashed, "\varepsilon"] \\
g'_1 \arrow[rr, phantom, "\sim"] & & g'_2
\end{tikzcd}$$

Figure 10: Confluence : Lemma 2

$g'_2 = g_2$).
From now on, we assume that $n_1 \neq n_2$. We consider two cases, depending on the position of n_1 and n_2 in g : either n_1 and n_2 are disjoint (i.e., $n_1 \| n_2$) or there exists a path from n_1 to n_2 (i.e., $n_1 \prec n_2$). The case $n_2 \prec n_1$ is symmetric.

Case 1 : n_1 and n_2 are disjoint (i.e., $n_1 \| n_2$) :
By definition of g_1, $g_{1|n_2} = (g[n_1 \leftarrow h_1(r_1)])_{|n_2}$. Since $n_1 \| n_2$, there exists no path from n_2 to n_1 in g, and there exists a path from a root of g to n_2 which does not go through n_1, so by Proposition 2 (persistence), $(g[n_1 \leftarrow h_1(r_1)])_{|n_2} = g_{|n_2}$, i.e., $g_{1|n_2} = g_{|n_2}$. By hypothesis, h_2 is a homomorphism from l_2 to $g_{|n_2}$, so h_2 is also a homomorphism from l_2 to $g_{1|n_2}$. In other words, l_2 matches g_1 at node n_2. Hence, g_1 can be rewritten with the rule $l_2 \to r_2$ into the graph $g'_1 = g_1[n_2 \leftarrow h_2(r_2)]$, that is to say $g'_1 = (g[n_1 \leftarrow h_1(r_1)])[n_2 \leftarrow h_2(r_2)]$. For the same reason, g_2 can be rewritten into $g'_2 = (g[n_2 \leftarrow h_2(r_2)])[n_1 \leftarrow h_1(r_1)]$. We now prove that $g'_1 = g'_2$. Let us verify that n_1 is not a node of $h_2(r_2)$ and n_2 is not a node of $h_1(r_1)$ in order to use Proposition 6 (commutativity). The graph $h_2(r_2)$ is composed of nodes of r_2 and of nodes introduced by h_2, that is to say nodes of $g_{|n_2}$ (since h_2 is a homomorphism from l_2 to $g_{|n_2}$). By hypothesis, $\mathcal{N}_g \cap \mathcal{N}_{r_2} = \emptyset$, so n_1 is not a node of r_2. Moreover, n_1 is not a node of $g_{|n_2}$ because n_1 and n_2 are disjoint. So n_1 is not a node of $h_2(r_2)$. With the similar reasoning, we can prove that n_2 is not a node of $h_1(r_1)$. Thus, by Proposition 6, $g'_1 = g'_2$.

Case 2 : There exists a path from n_1 to n_2 (i.e., $n_1 \prec n_2$) (see Figure 11) :
Since $n_1 \prec n_2$, n_2 is a node of $g_{|n_1}$. By hypothesis $h_1(l_1) = g_{|n_1}$, so n_2 is a node of $h_1(l_1)$. The fact that n_2 is a node of $h_1(l_1)$ stems from the existence of at least one variable node p_1 of l_1 such that n_2 is a node of $g_{|h_1(p_1)}$. The existence of p_1 is due to the fact that l_1 is a pattern, and $h_1(\mathcal{R}\mathrm{oot}_{l_1}) = n_1 \neq n_2$. There may be other variable nodes in l_1 like the node p_1, Let $P = \{p_1, p_2 \ldots p_k\}$ be the set of variable nodes p_i of l_1 such that n_2 is a node of $g_{|h_1(p_i)}$. There are two sub-cases to study, depending on $P \cap \mathcal{N}_{r_1} = \emptyset$ or not, i.e.,whether or not at least one variable node of P belongs to the right-hand side r_1.

Case 2.1 : ($P \cap \mathcal{N}_{r_1} \neq \emptyset$), i.e., there exists a node $p_k \in P$ which is a common variable node of r_1 and l_1 :
This sub-case is represented in Figure 11. By definition of P, n_2 is a node of $g_{|h_1(p_k)}$. By hypothesis, p_k is a variable node of r_1, thus n_2 is a node of the graph $h_1(r_1)$. n_2 is also a node of $g_1 = g[n_1 \leftarrow h_1(r_1)]$. Indeed, $h_1(r_1)$ is not modified by the pointer redirection from n_1 to $\mathcal{R}\mathrm{oot}_{h_1(r_1)}$ because $n_1 \notin h_1(r_1)$ (cf. lemma 1).

Let us show that $g_{1|n_2} = g_{|n_2}$. Since $n_1 \notin \mathcal{N}_{h_1(r_1)}$ and $n_2 \in \mathcal{N}_{h_1(r_1)}$, by proposition 3 (embedding), $g_{1|n_2} = (h_1(r_1))_{|n_2}$. Since n_2 is a node of $g_{|n_1}$ and $h_1(l_1) = g_{|n_1}$, we obtain $g_{|n_2} = (h_1(l_1))_{|n_2}$. n_2 being a common node of $h_1(l_1)$ and $h_1(r_1)$, we get $(h_1(l_1))_{|n_2} = (h_1(r_1))_{|n_2}$. Hence $g_{1|n_2} = g_{|n_2}$.

By hypothesis, h_2 is a homomorphism from l_2 to $g_{|n_2}$, so h_2 is also a homomorphism from l_2 to $g_{1|n_2}$ since $g_{|n_2} = g_{1|n_2}$. In other words, l_2 matches g_1 at node n_2. Therefore, g_1 can be rewritten with the rule $l_2 \to r_2$ into the graph $g'_1 = g_1[n_2 \leftarrow h_2(r_2)]$, that is to say $g'_1 = (g[n_1 \leftarrow h_1(r_1)])[n_2 \leftarrow h_2(r_2)]$.

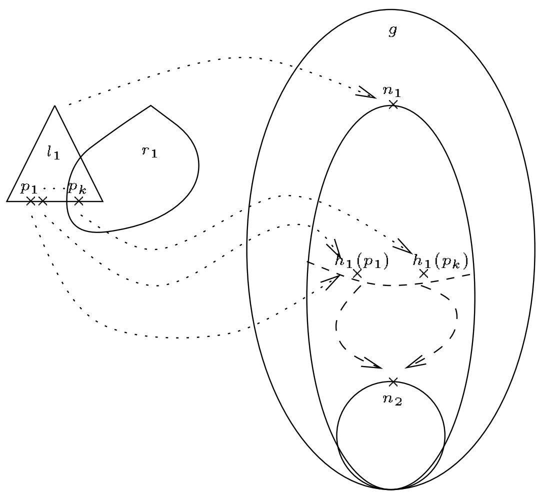

Figure 11:

We have just defined g'_1. We now define g'_2. By hypothesis, $n_1 \prec n_2$ and then $n_2 \not\prec n_1$ since g is admissible. Thus n_1 remains a node of $g_2 = g[n_2 \leftarrow h_2(r_2)]$. In addition, there exists a matcher $h'_1 : l_1 \to g_{2|n_1}$ because l_1 is a pattern and the construction of g_2 is obtained from g by a pointer redirection (replacement) at n_2, where n_2 is labeled with a defined operation. Hence, l_1 matches g_2 at node n_1. So g_2 can be rewritten into $g'_2 = g_2[n_1 \leftarrow h'_1(r_1)]$.

By definitions of h'_1 and h_1 we deduce that $h'_1(l_1) = h_1(l_1)[n_2 \leftarrow h_2(r_2)]$. Indeed, $h'_1(l_1) =$ (by definition of h'_1) $g_{2|n_1} =$ (By definition of g_2) $g[n_2 \leftarrow h_2(r_2)]_{|n_1} =$ (from $n_1 \prec n_2$) $g_{|n_1}[n_2 \leftarrow h_2(r_2)] =$ (By definition of h_1) $h_1(l_1)[n_2 \leftarrow h_2(r_2)]$. Then, it is easy to see that $h'_1(r_1) = h_1(r_1)[n_2 \leftarrow h_2(r_2)]$. Thus, $g'_2 = (g[n_2 \leftarrow h_2(r_2)])[n_1 \leftarrow (h_1(r_1)[n_2 \leftarrow h_2(r_2)])]$.

Finally, g_1 rewrites into $g'_1 = (g[n_1 \leftarrow h_1(r_1)])[n_2 \leftarrow h_2(r_2)]$ and g_2 rewrites into $g'_2 = (g[n_2 \leftarrow h_2(r_2)])[n_1 \leftarrow (h_1(r_1)[n_2 \leftarrow h_2(r_2)])]$. As n_1 is not a node of $h_2(r_2)$ since $n_1 \prec n_2$, by proposition 5 (associativity), we obtain $g'_1 = g'_2$.

Case 2.2 : $(P \cap \mathcal{N}_{r_1} = \emptyset)$. There is no variable node of P which appears in r_1 :
In this case, n_2 is not a node of $h_1(r_1)$. But n_2 may remain a node of $g[n_1 \leftarrow h_1(r_1)]$ if there exists a path from a root of g to n_2 which does not go through n_1. Hence, there are again two sub-cases.

Case 2.2.1 : Every path from a root of g to n_2 goes through n_1 :
Then n_2 is no more a node of $g_1 = g[n_1 \leftarrow h_1(r_1)]$: it is deleted by replacement since $P \cap \mathcal{N}_{r_1} = \emptyset$. Concerning g_2, by hypothesis $n_1 \prec n_2$ and $n_2 \not\prec n_1$ Which implies that n_1 is a node of $g_2 = g[n_2 \leftarrow h_2(r_2)]$. Moreover, there exists a homomorphism $h'_1 : l_1 \to g_{2|n_1}$ (for the same reason as in Case 2.1). Hence, l_1 matches g_2 at node n_1 and so g_2 can be rewritten into $g'_2 = g_2[n_1 \leftarrow h'_1(r_1)]$. By hypothesis, n_2 is not a node of $h_1(r_1)$, so $h'_1(r_1) = h_1(r_1)$. Thus, g_2 can be rewritten into $g'_2 = (g[n_2 \leftarrow h_2(r_2)])[n_1 \leftarrow h_1(r_1)]$. As n_2 is not a node of $h_1(r_1)$ and every path from a root of g to n_2 goes through n_1, by proposition 7 (weak dominance), we obtain $(g[n_2 \leftarrow h_2(r_2)])[n_1 \leftarrow h_1(r_1)] = g[n_1 \leftarrow h_1(r_1)]$, that is to say $g_2 \to_{[n_1, l_1 \to r_1]} g_1$. Finally, let $g'_1 = g'_2 = g_1$ and the lemma holds.

Case 2.2.2 : There exists a path C from a root of g to n_2 such that $n_1 \notin C$:
Then n_2 is not a node of $h_1(r_1)$ $(P \cap \mathcal{N}_{r_1} = \emptyset)$, but n_2 remains a node of $g_1 = g[n_1 \leftarrow h_1(r_1)]$, since the path C is not modified by the replacement at node n_1. There is no path from n_2 to n_1 and there exists a path from a root of g to n_2 which does not go through n_1, so by Proposition 2 (persistence), $(g[n_1 \leftarrow h_1(r_1)])_{|n_2} = g_{|n_2}$, i.e., $g_{1|n_2} = g_{|n_2}$. By hypothesis, l_2 matches $g_{|n_2}$, so

l_2 matches $g_{1|n_2}$. Hence, g_1 can be rewritten into $g'_1 = g_1[n_2 \leftarrow h_2(r_2)]$, that is to say $g'_1 = (g[n_1 \leftarrow h_1(r_1)])[n_2 \leftarrow h_2(r_2)]$. Concerning g_2, by hypothesis, $n_1 \prec n_2$ and $n_2 \not\prec n_1$ which implies that n_1 is a node of $g_2 = g[n_2 \leftarrow h_2(r_2)]$. Moreover, there exists a homomorphism $h'_1 : l_1 \to g_{2|n_1}$ (for the same reason as in Case 2.1). Hence, l_1 matches g_2 at node n_1 and so g_2 can be rewritten into $g'_2 = g_2[n_1 \leftarrow h'_1(r_1)]$. By hypothesis, n_2 is not a node of $h_1(r_1)$, so $h'_1(r_1) = h_1(r_1)$. Thus, g_2 can be rewritten into $g'_2 = (g[n_2 \leftarrow h_2(r_2)])[n_1 \leftarrow h_1(r_1)]$. Finally, n_1 is not a node of $h_2(r_2)$ (since $n_1 \prec n_2$ in g) and n_2 is not a node of $h_1(r_1)$ (by hypothesis), so by proposition 6 (commutativity), $g'_1 = g'_2$.

□

An immediate consequence of Lemma 2 is the following proposition.

Proposition 11 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS and g_1, g_2, g'_1 and g'_2 four admissible graphs such that g_1 and g_2 are equal up to renaming of nodes ($g_1 \sim g_2$), $g_1 \to g'_1$ and $g_2 \to g'_2$. Then there exist two admissible graphs g''_1 and g''_2 such that $g'_1 \xrightarrow{\varepsilon} g''_1$, $g'_2 \xrightarrow{\varepsilon} g''_2$ and $g''_1 \sim g''_2$. (see Figure 12).

$$\begin{tikzcd}
& g_1 \arrow[dl] & \sim & g_2 \arrow[dr] & \\
g'_1 \arrow[dr, dashed, "\varepsilon"'] & & & & g'_2 \arrow[dl, dashed, "\varepsilon"] \\
& g''_1 & \sim & g''_2 &
\end{tikzcd}$$

Figure 12:

Proof Since $g_1 \to g'_1$ and $g_1 \sim g_2$, by Proposition 10, there exists an admissible graph d such that $g_2 \to d$ and $g'_1 \sim d$. Since $g_2 \to d$ and $g_2 \to g'_2$, by Lemma 2, there exist two admissible graphs d' and g''_2 such that $d \xrightarrow{\varepsilon} d'$, $g'_2 \xrightarrow{\varepsilon} g''_2$ and $d' \sim g''_2$. Since $g'_1 \sim d$ and $d \xrightarrow{\varepsilon} d'$, by Proposition 10, there exists an admissible graph g''_1 such that $g'_1 \xrightarrow{\varepsilon} g''_1$ and $g''_1 \sim d'$. Finally, since $g''_1 \sim d'$ and $d' \sim g''_2$, we conclude that $g''_1 \sim g''_2$.

□

We now generalize Lemma 2.

Proposition 12 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS and g, g_1 and g_2 three admissible graphs. If $g \xrightarrow{*} g_1$ and g rewrites in at most one step to g_2 ($g \xrightarrow{\varepsilon} g_2$), then there exist two admissible graphs g'_1 and g'_2 such that $g_1 \xrightarrow{\varepsilon} g'_1$, $g_2 \xrightarrow{*} g'_2$ and $g'_1 \sim g'_2$. (see Figure 13).

$$\begin{tikzcd}
& g \arrow[dl, "*"'] \arrow[dr, "\varepsilon"] & \\
g_1 \arrow[d, dashed, "\varepsilon"'] & & g_2 \arrow[d, dashed, "*"] \\
g'_1 & \sim & g'_2
\end{tikzcd}$$

Figure 13:

$$\begin{tikzcd}
 & & & g \arrow[dl] \arrow[drrr] & & & \\
 & & d \arrow[dll, "n"'] \arrow[dr, "\varepsilon"] & & A & & g_2 \arrow[dl, "\varepsilon"'] \\
g_1 \arrow[dd, "\varepsilon"'] & HR & & d_1 \arrow[ddl, "*"'] & \sim_\alpha & d_2 \arrow[ddl, "*"'] & \\
 & & & C & & & \\
g_1' & \sim_\beta & d_1' & \sim_\gamma & g_2' & &
\end{tikzcd}$$

Figure 14:

Proof This proof is done by induction on the length of the derivation $g \overset{*}{\to} g_1$ and it is sketched in figure 14. For a null derivation, the proposition is straightforward (i.e., $g_1' = g_2' = g_2$). We suppose it is true for length n. Let g, g_1 and g_2 be three admissible graphs such that $g \overset{n+1}{\to} g_1$ and $g \overset{\varepsilon}{\to} g_2$. If $g = g_2$, the proposition is straightforward (i.e., $g_1' = g_2' = g_1$), so we suppose $g \to g_2$. There exists a graph d such that $g \to d \overset{n}{\to} g_1$. By Lemma 2, there exist two graphs d_1 and d_2 such that $d \overset{\varepsilon}{\to} d_1$, $g_2 \overset{\varepsilon}{\to} d_2$ and $d_1 \sim_\alpha d_2$ (see A in Figure 14). Since $d \overset{n}{\to} g_1$ and $d \overset{\varepsilon}{\to} d_1$, by induction hypothesis, there exists two graphs g_1' and d_1' such that $g_1 \overset{\varepsilon}{\to} g_1'$, $d_1 \overset{*}{\to} d_1'$ and $g_1' \sim_\beta d_1'$ (see HR in Figure 14). Since $d_1 \overset{*}{\to} d_1'$ and $d_1 \sim_\alpha d_2$, by Proposition 10, there exists a graph g_2' such that $d_2 \overset{*}{\to} g_2'$ and $d_1' \sim_\gamma g_2'$ (see C in Figure 14). Finally, $g_1' \sim_\beta d_1'$ and $d_1' \sim_\gamma g_2'$, so $g_1' \sim_{\gamma \circ \beta} g_2'$, i.e., g_1' and g_2' are equal up to renaming of nodes.

□

The following proposition is a second generalization of Lemma 2.

Proposition 13 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS and g, g_1 and g_2 three admissible graphs. If $g \overset{*}{\to} g_1$ and $g \overset{*}{\to} g_2$, then there exist two admissible graphs g_1' and g_2' such that $g_1 \overset{*}{\to} g_1'$, $g_2 \overset{*}{\to} g_2'$ and $g_1' \sim g_2'$. (see Figure 15).

$$\begin{tikzcd}
 & g \arrow[dl, "*"] \arrow[dr, "*"'] & \\
g_1 \arrow[d, dashed, "*"'] & & g_2 \arrow[d, dashed, "*"] \\
g_1' & \sim & g_2'
\end{tikzcd}$$

Figure 15:

Proof Let g, g_1 and g_2 be three admissible graphs such that $g \overset{*}{\to} g_1$ and $g \overset{*}{\to} g_2$. We prove by induction on the length of the derivation $g \overset{*}{\to} g_2$ that there exist two graphs g_1' and g_2' such that $g_1 \overset{*}{\to} g_1'$, $g_2 \overset{*}{\to} g_2'$ and $g_1' \sim g_2'$. This proof is sketched in figure 16. For a null derivation, the proposition is straightforward (i.e., $g_1' = g_2' = g_1$). We suppose it is true for length n. Let g, g_1 and g_2 be three admissible graphs such that $g \overset{*}{\to} g_1$ and $g \overset{n+1}{\to} g_2$. Then there exists a graph d such that $g \to d \overset{n}{\to} g_2$. By Proposition 12, there exist two graphs d_1 and d_2 such that $g_1 \overset{\varepsilon}{\to} d_1$, $d \overset{*}{\to} d_2$ and $d_1 \sim_\alpha d_2$ (see A in Figure 16). Since $d \overset{*}{\to} d_2$ and $d \overset{n}{\to} g_2$, by induction hypothesis, there exist two graphs d_2' and g_2' such that $d_2 \overset{*}{\to} d_2'$, $g_2 \overset{*}{\to} g_2'$ and $d_2' \sim_\beta g_2'$ (see HR in Figure 16). Since $d_2 \overset{*}{\to} d_2'$ and $d_1 \sim_\alpha d_2$, by Proposition 10, there exists a graph g_1' such that $d_1 \overset{*}{\to} g_1'$ and $g_1' \sim_\gamma d_2'$ (see C in

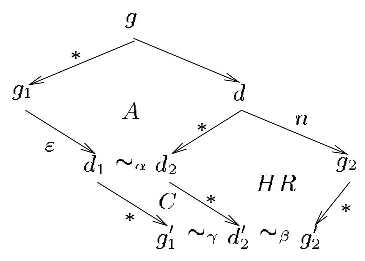

Figure 16:

Figure 16). Finally, $g_1' \sim_\gamma d_2'$ and $d_2' \sim_\beta g_2'$, so $g_1' \sim_{\beta\circ\gamma} g_2'$, i.e., g_1' and g_2' are equal up to renaming of nodes.

$\square$

We are now ready to prove the confluence of cGRS w.r.t the class of admissible graphs :

Proof of theorem 1 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS and g_1, g_2, g_3 and g_4 four admissible graphs such that $g_1 \sim g_2$, $g_1 \overset{*}{\to} g_1'$ and $g_2 \overset{*}{\to} g_2'$. Since $g_1 \overset{*}{\to} g_1'$ and $g_1 \sim g_2$, by Proposition 10, there exists an admissible graph d such that $g_2 \overset{*}{\to} d$ and $g_1' \sim d$. Since $g_2 \overset{*}{\to} d$ and $g_2 \overset{*}{\to} g_2'$, by Proposition 13, there exist two admissible graphs d' and g_2'' such that $d \overset{*}{\to} d'$, $g_2' \overset{*}{\to} g_2''$ and $d' \sim g_2''$. Since $g_1' \sim d$ and $d \overset{*}{\to} d'$, by Proposition 10, there exists an admissible graph g_1'' such that $g_1' \overset{*}{\to} g_1''$ and $g_1'' \sim d'$. Finally, since $g_1'' \sim d'$ and $d' \sim g_2''$, we conclude that $g_1'' \sim g_2''$.

5 Confluence modulo bisimilarity

Another interesting question is the confluence modulo bisimilarity of the graph rewriting relation with respect to admissible graphs. Beyond its theoretical interest, we need this result to establish the completeness of narrowing in the framework of cGRSs [EJ97].

Definition 25 (Confluence modulo bisimilarity)
Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. We say that *the rewriting relation $\overset{*}{\to}$ is confluent modulo bisimilarity w.r.t the class of admissible graphs* iff for all admissible graphs g_1, g_2, g_1' and g_2' such that g_1 and g_2 are bisimilar ($g_1 \doteq g_2$), $g_1 \overset{*}{\to} g_1'$ and $g_2 \overset{*}{\to} g_2'$. Then there exist two admissible graphs g_1'' and g_2'' such that $g_1' \overset{*}{\to} g_1''$, $g_2' \overset{*}{\to} g_2''$ and $g_1'' \doteq g_2''$. (see Figure 17). $\square$

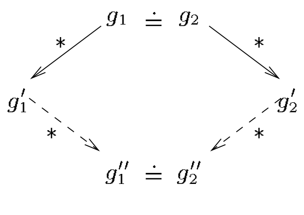

Figure 17: Confluence modulo bisimilarity

Theorem 2 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. Then the rewriting relation $\overset{*}{\to}$ is confluent modulo bisimilarity w.r.t the class of admissible graphs.

The rest of this section is dedicated to the proof of Theorem 2. We must first establish a few technical results. The following proposition states that if two graphs are bisimilar and the left-hand side of a rule matches one of them, then it also matches the other.

Proposition 14 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g_1 and g_2 two admissible graphs and $l \to r$ a rewrite rule of $\mathcal{R}$.

1. If there exists a homomorphism $h : g_1 \to g_2$ and l matches g_1 at the root, then l matches g_2 at the root.
2. If there exists a *variable-preserving* homomorphism $h : g_1 \to g_2$ and l matches g_2 at the root, then l matches g_1 at the root.
3. If g_1 and g_2 are bisimilar and l matches g_1 at the root, then l matches g_2 at the root.

Proof The first part of the proposition is straightforward : if there exists a homomorphism $h_1 : l \to g_1$ and a homomorphism $h : g_1 \to g_2$, then by composition, $h \circ h_1$ is a homomorphism from l to g_2, and thus l matches g_2 at the root.

Let us prove the second part of the proposition. By definition of a cGRS, l is a pattern, so l is a finite linear tree. Hereafter, we use this property, since the proof is done by induction on the height of l, i.e., the maximal length of paths from $\mathcal{R}\mathrm{oot}_l$ to the leaves of l.

Base case: Let l be a linear tree of height $\eta = 0$, g_1 and g_2 two admissible graphs and $h : g_1 \to g_2$ a variable-preserving homomorphism such that l matches g_2 at the root. Since $\eta = 0$, l is either a variable or a constant. If l is a variable, l trivial matches g_1 at the root. If l is a constant, then g_2 is the same constant because $h_2 : l \to g_2$ is a homomorphism. As $h : g_1 \to g_2$ is a variable-preserving homomorphism, g_1 is also the same constant. Hence l matches g_1 at the root.

Induction step: Let $\eta > 0$. We suppose that for every linear tree l of height less or equal to η, for all admissible graphs g_1 and g_2 and variable-preserving homomorphism $h : g_1 \to g_2$, if l matches g_2 at the root, then l matches g_1 at the root. Let l be a linear tree of height $\eta + 1$, g_1 and g_2 two admissible graphs and $h : g_1 \to g_2$ a variable-preserving homomorphism, such that there exists a homomorphism $h_2 : l \to g_2$. We build a homomorphism $h_1 : l \to g_1$.
l is not a variable nor a constant, since it is of height $\eta + 1$. So its root is labeled with a non constant operation f. Since $h_2 : l \to g_2$ is a homomorphism, the root of g_2 is also labeled with f. And as $h : g_1 \to g_2$ is a variable-preserving homomorphism, the root of g_1 is also labeled with f. We now suppose $\mathcal{S}_l(\mathcal{R}\mathrm{oot}_l) = n_1 \ldots n_k$, $\mathcal{S}_{g_1}(\mathcal{R}\mathrm{oot}_{g_1}) = u_1 \ldots u_k$ and $\mathcal{S}_{g_2}(\mathcal{R}\mathrm{oot}_{g_2}) = v_1 \ldots v_k$. For all $i \in 1..k$, $h_{|u_i}$ is a variable-preserving homomorphism from $g_{1|u_i}$ to $g_{2|v_i}$. For all $i \in 1..k$, $l_{|n_i}$ is a linear tree of height less or equal to η and $h_{2|n_i}$ is a homomorphism from $l_{|n_i}$ to $g_{2|v_i}$. So by induction hypothesis, there exists a matcher $\phi_i : l_{|n_i} \to g_{1|u_i}$ for all $i \in 1..k$. Let $h_1 : \mathcal{N}_l \to \mathcal{N}_{g_1}$ be the function such that $h_1(\mathcal{R}\mathrm{oot}_l) = \mathcal{R}\mathrm{oot}_{g_1}$ and for all $i \in 1..k$, for all $n \in \mathcal{N}_{l_{|n_i}}$, $h_1(n) = \phi_i(n)$. We now prove that h_1 is a homomorphism from l to g_1. First, l is a linear tree, so for all $i, j \in 1..k$, if $i \neq j$, then $\mathcal{N}_{(l_{|n_i})} \cap \mathcal{N}_{(l_{|n_j})} = \emptyset$ and $\mathcal{V}(l_{|n_i}) \cap \mathcal{V}(l_{|n_j}) = \emptyset$. Thus, for all nodes n of l, $h_1(n)$ is defined once and only once, i.e., h_1 is a well defined function. Second, h_1 preserves the labeling function of l, since $\mathcal{L}_{g_1}(h_1(\mathcal{R}\mathrm{oot}_l)) = \mathcal{L}_{g_1}(\mathcal{R}\mathrm{oot}_{g_1}) = \mathcal{L}_l(\mathcal{R}\mathrm{oot}_l) = f$, and every ϕ_i preserves the labeling function of $l_{|n_i}$. Third, h_1 preserves the successor function of l, since $\mathcal{S}_{g_1}(\mathcal{R}\mathrm{oot}_{g_1}) = u_1 \ldots u_k = \phi_1(n_1) \ldots \phi_k(n_k) = h_1(\mathcal{S}_l(\mathcal{R}\mathrm{oot}_l))$, and every ϕ_i preserves the successor function of $l_{|n_i}$. So we conclude that $h_1 : l \to g_1$ is a homomorphism and thus, l matches g_1 at the root.

Finally, we prove the last part of the proposition. Let g_1 and g_2 be two bisimilar graphs such that l matches g_1 at the root. By definition of bisimilarity, there exist a graph g and two variable-preserving homomorphisms $\mu_1 : g_1 \to g$ and $\mu_2 : g_2 \to g$. l matches g_1 at the root, so l matches g at the root, with the first part of this proposition. Then by the second part of this proposition, we obtain l matches g_2 at the root □

Note : The proposition above would not hold if the left-hand sides of the rules of a cGRS were not finite linear trees. Consider for instance the rule `x1:+(x2:0,x2)` $\rightarrow$ `x2`. Its left-hand side l does not match the graph $g_1 =$ `y1:+(y2:0,y3:0)`, since there is no homomorphism from l to g_1. Consequently, g_1 cannot be rewritten. However, l and g_1 are bisimilar and l can be rewritten into `x2:0`. Consider now the rule `x1:+(x2:x,x3:s(x3))` $\rightarrow$ `x3`. Its left-hand side matches the graph $g_3 =$ `y1:+(x2:0,x3:s(x3))` but it does not match the graph $g_4 =$ `z1:+(z2:0,z3:s(z4:s(z3)))`, whereas g_3 and g_4 are bisimilar.

We now establish a key result for the proof of confluence modulo bisimilarity of our graph rewriting relation. A similar proposition is given in [AK96] (proposition 5.13) (for a different rewriting relation, as already said).

Lemma 3 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g_1 and g_2 two admissible graphs and $h : g_1 \rightarrow g_2$ a *variable-preserving* homomorphism. If there exists a graph g'_2 such that $g_2 \rightarrow g'_2$, then there exist an admissible graph g'_1 and a variable-preserving homomorphism $h' : g'_1 \rightarrow g'_2$ such that $g_1 \xrightarrow{+} g'_1$. (see Figure 18).

$$\begin{tikzcd}
g_1 \arrow[r, "h"] \arrow[d, dashed, "+"] & g_2 \arrow[d] \\
g'_1 \arrow[r, dotted, "h'"] & g'_2
\end{tikzcd}$$

Figure 18:

Proof We assume $g_2 \rightarrow_{[q,l\rightarrow r]} g'_2$. By definition, there exists a homomorphism $\psi : l \rightarrow g_{2|q}$, and $g'_2 = g_2[q \leftarrow \psi(r)]$.

Let $P = h^{-1}(q) = \{p_1, p_2, \ldots, p_n\}$. All nodes of P are pairwise disjoint : otherwise, if $p_i \prec p_j$ and $i \neq j$, then there exists a path $C = [p_i, n_1, \ldots, n_k, p_j]$ in g_1 ; h being a homomorphism, $h(C) = [h(p_i), h(u_1), \ldots, h(u_k), h(p_j)]$ is a path of g_2 ; but $h(p_i) = h(p_j) = q$, so there exists a cycle on q in g_2, which is impossible since q is labeled with a defined operation and g_2 is admissible. Therefore all nodes of P are pairwise disjoint.

Let $\{l_i \rightarrow r_i \mid i \in 1..n\}$ be a set of n variants of $l \rightarrow r$; let α_i be the isomorphism from the graph $l_i \rightarrow r_i$ to the graph $l \rightarrow r$ (i.e., α_i is the isomorphism from $l_i + r_i$ to $l + r$ which renames all nodes and variables of $l_i + r_i$). ψ is a homomorphism from l to $g_{2|q}$ and $h_{|p_i}$ is a homomorphism from $g_{1|p_i}$ to $g_{2|q}$ for all $i \in 1..n$. So by proposition 14, there exists a homomorphism from l to $g_{1|p_i}$. $l_i \rightarrow r_i$ is a variant of $l \rightarrow r$ by isomorphism α_i and there exists a homomorphism from l to $g_{1|p_i}$, so there exist a homomorphism ϕ_i from l_i to $g_{1|p_i}$. Hence, for all $i \in 1..n$, g_1 can be rewritten at node p_i with rule $l_i \rightarrow r_i$ into $g_1[p_i \leftarrow \phi_i(r_i)]$. All positions in P are pairwise disjoint, so by proposition 6 (commutativity), we can perform all these rewriting steps in any order and we obtain a graph $g'_1 = g_1[p_1 \leftarrow \phi_1(r_1)] \ldots [p_n \leftarrow \phi_n(r_n)]$. We clearly have $g_1 \xrightarrow{+} g'_1$.

Now, let us prove that there exists a variable-preserving homomorphism $h' : g'_1 \rightarrow g'_2$. We are going to decompose the construction of g'_1 and g'_2 in elementary stages (following the definition of replacement) and prove that there exists a variable-preserving homomorphism at each stage.
First stage, let $G_1 = g_1 + \phi_1(r_1) + \ldots + \phi_n(r_n)$ and $G_2 = g_2 + \psi(r)$. There exists a homomorphism $\mu : G_1 \rightarrow G_2$ such that for all nodes $n \in \mathcal{N}_{g_1}$, $\mu(n) = h(n)$ and for all $i \in 1..k$, for all nodes $n \in (\mathcal{N}_{r_i} - \mathcal{N}_{l_i})$, $\mu(n) = \alpha_i(n)$.

Second stage, we perform the multiple pointer redirection ρ_1 on G_1 and the pointer redirection ρ_2 on G_2, where $\rho_1 = \{p_1 \mapsto \mathcal{R}\text{oot}_{\phi_1(r_1)}, \ldots, p_n \mapsto \mathcal{R}\text{oot}_{\phi_n(r_n)}\}$ and $\rho_2 = \{q \mapsto \mathcal{R}\text{oot}_{\psi(r)}\}$. We must prove the μ is still a homomorphism from $\rho_1(G_1)$ to $\rho_2(G_2)$. First, μ is a function from $\mathcal{N}_{G_1}$ to $\mathcal{N}_{G_2}$, so μ is also a function from $\mathcal{N}_{\rho_1(G_1)}$ to $\mathcal{N}_{\rho_2(G_2)}$. Second, μ preserves the labeling function of G_1, so it does on $\rho_1(G_1)$. Third, μ preserves the successor function of $\rho_1(G_1)$: we must verify that $\mathcal{S}_{\rho_2(G_2)}(\mu(n)) = \mu(\mathcal{S}_{\rho_1(G_1)}(n))$, for all node $n \in \mathcal{N}_{\rho_1(G_1)}$. As $\mathcal{N}_{\rho_1(G_1)} = \mathcal{N}_{G_1}$, it is equivalent to prove that for all nodes $n \in \mathcal{N}_{G_1}$, $\rho_2(\mathcal{S}_{G_2}(\mu(n))) = \mu(\rho_1(\mathcal{S}_{G_1}(n)))$. Since μ is a homomorphism from G_1 to G_2, $\rho_2(\mathcal{S}_{G_2}(\mu(n))) = \rho_2(\mu(\mathcal{S}_{G_1}(n)))$. So, let us prove that for all nodes $n \in \mathcal{N}_{G_1}$, $\mu(\rho_1(\mathcal{S}_{G_1}(n))) = \rho_2(\mu(\mathcal{S}_{G_1}(n)))$. This result holds if $\rho_2(\mu(n)) = \mu(\rho_1(n))$ for all nodes n of G_1. There are two cases for n. First case, if $n \notin P$. Then $\rho_1(n) = n$ and so $\mu(\rho_1(n)) = \mu(n)$. As $n \notin P$, $\mu(n) \neq q$, so $\rho_2(\mu(n)) = \mu(n)$. Hence, $\mu(\rho_1(n)) = \rho_2(\mu(n))$. Second case, if $n \in P$. Then there exists an $i \in 1..k$ such that $n = p_i$. So $\rho_1(n) = \rho_1(p_i) = \mathcal{R}\text{oot}_{\phi_i(r_i)} = \phi_i(\mathcal{R}\text{oot}_{r_i})$. So $\mu(\rho_1(n)) = \mu(\phi_i(\mathcal{R}\text{oot}_{r_i}))$. Moreover, $\rho_2(\mu(n)) = \rho_2(\mu(p_i)) = \rho_2(q) = \mathcal{R}\text{oot}_{\psi(r)} = \psi(\mathcal{R}\text{oot}_r)$. By definition of μ, $\mu(\phi_i(\mathcal{R}\text{oot}_{r_i})) = \psi(\mathcal{R}\text{oot}_r)$. Hence, $\mu(\rho_1(n)) = \rho_2(\mu(n))$ and we obtain that μ preserves the successor function of $\rho_1(G_1)$. Finally, $\mu(\mathcal{R}\text{oots}_{\rho_1(G_1)}) = \mathcal{R}\text{oots}_{\rho_2(G_2)}$. So we conclude that μ is still a homomorphism from $\rho_1(G_1)$ to $\rho_2(G_2)$.
Third and last stage, let $g_1' = \rho_1(G_1)_{|\rho_1(\mathcal{R}\text{oots}_{g_1})}$ and $g_2' = \rho_2(G_2)_{|\rho_2(\mathcal{R}\text{oots}_{g_2})}$. Let $h' = \mu_{|\mathcal{R}\text{oot}_{g_1'}}$. It is clear that h' is a variable-preserving homomorphism from g_1' to g_2'.

□

The following proposition is a generalization of Lemma 3.

Proposition 15 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g_1 and g_2 two admissible graphs and $h : g_1 \to g_2$ a *variable-preserving* homomorphism. If there exists a graph g_2' such that $g_2 \xrightarrow{*} g_2'$, then there exist a graph g_1' and a variable-preserving homomorphism $h' : g_1' \to g_2'$ such that $g_1 \xrightarrow{*} g_1'$. (see Figure 19).

$$\begin{tikzcd}
g_1 \arrow[r, "h"] \arrow[d, dashed, "*"] & g_2 \arrow[d, "*"] \\
g_1' \arrow[r, dotted, "h'"] & g_2'
\end{tikzcd}$$

Figure 19:

Proof This proof is done by induction on the length of the derivation $g_2 \xrightarrow{*} g_2'$. For a null derivation, the proposition is straightforward (i.e., $g_1' = g_1$, $g_2' = g_2$, $h' = h$). We suppose the proposition is true for length n. Let g_1 and g_2 be two admissible graphs and $h : g_1 \to g_2$ a variable-preserving homomorphism such that $g_2 \xrightarrow{n+1} g_2'$. Then, there exists a graph d' such that $g_2 \to d' \xrightarrow{n} g_2'$. Since $g_2 \to d'$ and there exists a variable-preserving homomorphism $h : g_1 \to g_2$, by Lemma 3, there exist a graph d and a variable-preserving homomorphism $j : d \to d'$ such that $g_1 \xrightarrow{+} d$. Since there exists a variable-preserving homomorphism $j : d \to d'$ and $d' \xrightarrow{n} g_2'$, by induction hypothesis, there exist a graph g_1' and a variable-preserving homomorphism $h' : g_1' \to g_2'$ such that $d \xrightarrow{*} g_1'$. Finally, $g_1 \xrightarrow{+} d \xrightarrow{*} g_1'$, so $g_1 \xrightarrow{*} g_1'$.

□

We now establish a second key result for the proof of confluence modulo bisimilarity of cGRSs.

Lemma 4 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g_1 and g_2 two admissible graphs and $h : g_1 \to g_2$ a *variable-preserving* homomorphism. If there exists a graph g_1' such that $g_1 \to g_1'$, then there exist two admissible graphs g_1'' and g_2' and a variable-preserving homomorphism $h' : g_1'' \to g_2'$ such that $g_1' \xrightarrow{*} g_1''$ and $g_2 \to g_2'$. (see Figure 20).

$$\begin{tikzcd}
 & g_1 \arrow[r, "h"] \arrow[dl] & g_2 \arrow[dd, dashed] \\
g_1' \arrow[dr, dashed, "*"'] & & \\
 & g_1'' \arrow[r, dotted, "h'"] & g_2'
\end{tikzcd}$$

Figure 20:

Proof This proof is sketched in Figure 21. We assume g_1 rewrites to g_1' at node p_1 with rule

$$\begin{tikzcd}
 & g_1 \arrow[r, "h"] \arrow[dl] \arrow[dd, "+"] & g_2 \arrow[dd] \\
g_1' \arrow[d, "*"'] \arrow[r, phantom, "B"] & {} \arrow[r, phantom, "A"] & {} \\
g_1'' \arrow[r, phantom, "\sim_\alpha"] & d \arrow[r, "j"] & g_2'
\end{tikzcd}$$

Figure 21:

$l \to r$. Let $q = h(p_1)$ and $P = h^{-1}(q) = \{p_1, \ldots p_n\}$ (i.e., P is the set of the antecedents of q by h). Since there exists a homomorphism from l to $g_{1|p_1}$ (i.e., the matcher) and $h_{|p_1}$ is a homomorphism from $g_{1|p_1}$ to $g_{2|q}$, then by composition, there exists a homomorphism from l to $g_{2|q}$. So there exists a graph g_2' such that $g_2 \to_{[q, l\to r]} g_2'$. Since $g_2 \to_{[q,l\to r]} g_2'$ and there exists a variable-preserving homomorphism $h : g_1 \to g_2$, by Lemma 3, there exists a graph d and a variable-preserving homomorphism $j : d \to g_2'$ such that $g_1 \xrightarrow{+} d$ (see A in Figure 21). From the proof of Lemma 3, the nodes of P are pairwise disjoint in g_1 and all rewriting steps of the derivation $g_1 \xrightarrow{+} d$ are performed at the nodes of P. Therefore, since p_1 is a node of P and $g_1 \to_{[p_1, l\to r]} g_1'$, there exists a graph g_1'' such that $g_1' \xrightarrow{*} g_1''$ and $g_1'' \sim_\alpha d$ (see B in Figure 21). The nodes used to perform the rewriting derivation $g_1' \xrightarrow{*} g_1''$ are those of $P - \{p_1\}$, that is to say $\{p_2, \ldots, p_n\}$. Finally, the isomorphism of renaming α is a variable-preserving homomorphism from g_1'' to d and j is a variable-preserving homomorphism from d to g_2', so $\alpha \circ j$ is a variable-preserving homomorphism from g_1'' to g_2'.

□

The following proposition is a generalization of lemma 4.

Proposition 16 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g_1 and g_2 two admissible graphs and $h : g_1 \to g_2$ a *variable-preserving* homomorphism. If there exists a graph g_1' such that $g_1 \xrightarrow{*} g_1'$, then there exist two admissible graphs g_1'' and g_2' and a variable-preserving homomorphism $h' : g_1'' \to g_2'$ such that $g_1' \xrightarrow{*} g_1''$ and $g_2 \xrightarrow{*} g_2'$. (see Figure 22).

$$\begin{tikzcd}
 & g_1 \arrow[r, "h"] \arrow[dl, "*"'] & g_2 \arrow[dd, dashed, "*"] \\
g'_1 \arrow[dr, dashed, "*"'] & & \\
 & g''_1 \arrow[r, dotted, "h'"] & g'_2
\end{tikzcd}$$

Figure 22:

Proof This proof is done by induction on the length of the derivation $g_1 \overset{*}{\to} g'_1$ and it is sketched in Figure 23. For a null derivation, the proposition is straightforward (i.e., $g''_1 = g_1$, $g'_2 = g_2$

$$\begin{tikzcd}
 & & & & g_1 \arrow[r, "h"] \arrow[dll, "n"'] & g_2 \arrow[dd, "*"] \\
 & & d \arrow[dll] \arrow[drr, "*"] & & HR & \\
g'_1 \arrow[dr, "*"'] & & B & & d_1 \arrow[r, "j"] \arrow[dl, "\varepsilon"] & d_2 \arrow[ddd, "\varepsilon"] \\
 & f_1 \arrow[ddr, "*"'] \arrow[rr, phantom, "\sim_\alpha"] & & f_2 \arrow[ddr, "*"] & C & \\
 & & D & & & \\
 & & g''_1 \arrow[rr, phantom, "\sim_\beta"] & & e \arrow[r, "j'"] & g'_2
\end{tikzcd}$$

Figure 23:

and $h' = h$). We suppose the proposition is true for length n. Let g_1 and g_2 be two admissible graphs and $h : g_1 \to g_2$ a variable-preserving homomorphism such that there exists a graph g'_1 with $g_1 \overset{n+1}{\to} g'_1$. Then, there exists a graph d such that $g_1 \overset{n}{\to} d \to g'_1$. Since $h : g_1 \to g_2$ is a variable-preserving homomorphism and $g_1 \overset{n}{\to} d$, by induction hypothesis, there exists two graphs d_1 and d_2 and a variable-preserving homomorphism $j : d_1 \to d_2$ such that $d \overset{*}{\to} d_1$ and $g_2 \overset{*}{\to} d_2$ (see HR in Figure 23). Since $d \to g'_1$ and $d \overset{*}{\to} d_1$, by Proposition 12, there exists two graphs f_1 and f_2 such that $g'_1 \overset{*}{\to} f_1$, $d_1 \overset{\varepsilon}{\to} f_2$, and $f_1 \sim_\alpha f_2$ (see B in Figure 23). Since there exists a variable-preserving homomorphism $j : d_1 \to d_2$ and $d_1 \overset{\varepsilon}{\to} f_2$, we can deduce by using Lemma 4, the existence of two graphs e and g'_2 and a variable-preserving homomorphism $j' : e \to g'_2$ such that $f_2 \overset{*}{\to} e$ and $d_2 \overset{\varepsilon}{\to} g'_2$ (see C in Figure 23). Since $f_2 \overset{*}{\to} e$ and $f_1 \sim_\alpha f_2$, by Proposition 10, there exists a graph g''_1 such that $f_1 \overset{*}{\to} g''_1$ and $g''_1 \sim_\beta e$ (see D in Figure 23). Finally, $g''_1 \sim_\beta e$, so β is a variable-preserving homomorphism from g''_1 to e. Moreover, $j' : e \to g'_2$ is a variable-preserving homomorphism. So $j' \circ \beta : g''_1 \to g'_2$ is a variable-preserving homomorphism.

□

We are now ready to prove Theorem 2.

Proof of theorem 2 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. Let g_1 and g_2 be two admissible and bisimilar graphs ($g_1 \doteq g_2$) such that there exist two graphs g'_1 and g'_2 with $g_1 \overset{*}{\to} g'_1$ and $g_2 \overset{*}{\to} g'_2$. We prove that there exist two admissible graphs g''_1 and g''_2 such that $g'_1 \overset{*}{\to} g''_1$, $g'_2 \overset{*}{\to} g''_2$ and $g''_1 \doteq g''_2$. This proof is direct thanks to the previous propositions. It is sketched in Figure 24. By definition of bisimilarity, there exist a graph g, a variable-preserving homomorphism $h_1 : g_1 \to g$ and a variable-preserving homomorphism $h_2 : g_2 \to g$ (see A in Figure 24). Since $g_1 \overset{*}{\to} g'_1$ and there exists a

$$\begin{tikzcd}
& g_1 \arrow[dl, "*"'] \arrow[drr, "h_1"] & & \overset{\doteq}{A} & & g_2 \arrow[dll, "h_2"'] \arrow[dr, "*"] & \\
g_1' \arrow[d, "*"'] & & B & g \arrow[dl, "*"'] \arrow[dr, "*"] & B' & & g_2' \arrow[d, "*"] \\
f_1 \arrow[rr, "h_1'"] \arrow[dr, "*"'] & D & d_1 \arrow[d, "*"'] & C & d_2 \arrow[d, "*"] & D' & f_2 \arrow[ll, "h_2'"'] \arrow[dl, "*"] \\
& g_1'' \arrow[r, "h_1''"'] & d_1' \arrow[rr, phantom, "\sim_\alpha"] & & d_2' & g_2'' \arrow[l, "h_2''"] &
\end{tikzcd}$$

Figure 24:

variable-preserving homomorphism $h_1 : g_1 \to g$, by Proposition 16, there exist two graphs f_1 and d_1 and a variable-preserving homomorphism $h_1' : f_1 \to d_1$ such that $g_1' \xrightarrow{*} f_1$ and $g \xrightarrow{*} d_1$ (see B in Figure 24). For the same reason, there exist two graphs f_2 and d_2 and a variable-preserving homomorphism $h_2' : f_2 \to d_2$ such that $g_2' \xrightarrow{*} f_2$ and $g \xrightarrow{*} d_2$ (see B' in Figure 24). Since $g \xrightarrow{*} d_1$ and $g \xrightarrow{*} d_2$, by Theorem 1, there exist two graphs d_1' and d_2' such that $d_1 \xrightarrow{*} d_1'$, $d_2 \xrightarrow{*} d_2'$ and $d_1' \sim_\alpha d_2'$ (see C in Figure 24). Since $d_1 \xrightarrow{*} d_1'$ and there exists a variable-preserving homomorphism $h_1' : f_1 \to d_1$, by Proposition 15, there exist a graph g_1'' and a variable-preserving homomorphism $h_1'' : g_1'' \to d_1'$ such that $f_1 \xrightarrow{*} g_1''$ (see D in Figure 24). For the same reason, there exist a graph g_2'' and a variable-preserving homomorphism $h_2'' : g_2'' \to d_2'$ such that $f_2 \xrightarrow{*} g_2''$ (see D' in Figure 24). Finally, $\alpha \circ h_1''$ is a variable-preserving homomorphism from g_1'' to d_2' and h_2'' is a variable-preserving homomorphism from g_2'' to d_2', so by definition of bisimilarity, $g_1'' \doteq g_2''$.

6 Needed graph rewriting

In this section, we define an optimal rewriting strategy for admissible graphs. We first introduce some vocabulary.

Definition 26 (Strategy)
Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS. A *(sequential) graph rewriting strategy* is a partial function $\mathcal{S}$ which takes an admissible graph g and returns a pair (p, R) such that p is a node of g, R is a rewrite rule of $\mathcal{R}$ and g can be rewritten at node p with rule R. We write $g \to_{\mathcal{S}} g'$ and speak of an *$\mathcal{S}$-step* from g to g' whenever $\mathcal{S}(g) = (p, R)$ and $g \to_{[p,R]} g'$. $\xrightarrow{*}_{\mathcal{S}}$ denotes the reflexive and transitive closure of $\to_{\mathcal{S}}$ and we speak of an *$\mathcal{S}$-derivation* from g to g' whenever $g \xrightarrow{*}_{\mathcal{S}} g'$.
A strategy $\mathcal{S}$ is *c-normalizing* iff for all admissible graph g admitting a constructor normal form c, if $g \xrightarrow{*} c$, then there exists a constructor graph c' such that $g \xrightarrow{*}_{\mathcal{S}} c'$ and $c' \sim c$.
A strategy $\mathcal{S}$ is *c-hyper-normalizing* iff for all admissible graphs g admitting a constructor normal form c, any derivation D starting with g which uses infinitely many times $\mathcal{S}$-steps ends with a constructor normal form c' such that $c \sim c'$. □

Definition 27 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g and g' two admissible graphs and $B = g \xrightarrow{*} g'$ a rewriting derivation. A node q labeled with a defined operation in g *is a residual node by B* if q remains a node of g'. Then, we call *descendant of $g_{|q}$* the subgraph $(g')_{|q}$. A redex u rooted by q in g is a *needed redex* iff in every rewriting derivation from g to a constructor normal form, a descendant of $g_{|q}$ is rewritten at its root q. A node q labeled with a defined operation in g is an *outermost node* of g iff $q \in \mathcal{R}\mathrm{oots}_g$ or there exist a root $r \in \mathcal{R}\mathrm{oots}_g$ and a path $C = [p_0, \ldots, p_k]$ from r to q such that $p_0 = r$, $p_k = q$ and p_i is a node of g labeled with a constructor symbol for all $i \in 0..(k-1)$. A redex u rooted by q in g is an *outermost redex* iff $q \in \mathcal{R}\mathrm{oots}_g$ or there exist a root

$r \in \mathcal{R}oots_g$ and a path $C = [p_0, \ldots, p_k]$ from r to q such that $p_0 = r$, $p_k = q$ and $g_{|p_i}$ is not a redex for all $i \in 0..(k-1)$. □

Example 20 Let $g =$ `x1:cons(x2:*(x3:0,x4:+(x3,x3)),x5:cons(x4,x6:nil))`. By definition, $g_{|\texttt{x2}}$ and $g_{|\texttt{x4}}$ are outermost needed redexes. However, there exist a path from `x2` to `x4`. Notice that $g_{|\texttt{x4}}$ does not vanish when $g_{|\texttt{x2}}$ is rewritten.

Remark : The notions of outermost node and outermost redex are well-defined in the framework of admissible graphs : if p and q are two nodes of an admissible graph labeled with defined operations and such that there exist a path from p to q (i.e., p is *outer* than q), then there is no path from q to p.

Our graph rewriting strategy is based on definitional trees and is a conservative extension to admissible graphs of Antoy's term rewriting strategy [Ant92]. A definitional tree is a hierarchical structure whose leaves are the rules of a cGRS used to define a given operation. In the following definition, *branch* and *rule* are uninterpreted symbols, used to construct the nodes of a definitional tree.

Definition 28 (Definitional tree)
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 s, $c_1, \ldots, c_k$ $(k > 0)$ are different constructors of the sort s and for all $j \in 1..k$, $\mathcal{T}_j$ is a pdt with pattern $\pi[o \leftarrow p : c_j(o_1 : X_1, \ldots, o_n : X_n)]$, such that n is the number of arguments of c_j, $X_1, \ldots, X_n$ are new variables and $p, o_1, \ldots, o_n$ are new nodes.

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

Example 21 Consider the following cGRS which defines the sum of positive integers :

```
x1:+(x2:0,x3:x) -> x3
x4:+(x5:succ(x6:x),x7:y) -> x8:succ(x9:+(x6,x7))
```

The definitional tree $\mathcal{T}_+$ of the operation `+` is

$$\begin{array}{ll} \mathcal{T}_+ = branch(& \texttt{l1:+(l2:X,l3:Y)}, \\ & \texttt{l2}, \\ & rule(\texttt{l4:+(l5:0,l6:U)} \to \texttt{l6}), \\ & rule(\texttt{l7:+(l8:succ(l9:V),l10:W)} \to \texttt{l11:succ(l12:+(l9,l10))})) \end{array}$$

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

Definition 29 (Admissible graph rewriting strategy)
Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an inductively sequential cGRS and g an admissible graph. Φ is the partial function defined by $\Phi(g) = \varphi(g_{|p}, \mathcal{T}_f)$, where p is an outermost node of g labeled with a defined operation f and $\mathcal{T}_f$ is a definitional tree of f.
Let g be an admissible operation-rooted term graph and $\mathcal{T}$ a pdt such that $pattern(\mathcal{T}) \leq g$. We define the partial function φ :

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

□

Example 22 Consider the admissible graph $g =$ `x1:cons(x2:+(x3:+(x4:0,x4),x3),x1)`. Then

$$\begin{array}{rcl} \Phi(g) & = & \varphi(\ \texttt{x2:+(x3:+(x4:0,x4),x3)}, \mathcal{T}_{+}) \\ & = & \varphi(\ \texttt{x2:+(x3:+(x4:0,x4),x3)}, \\ & & \quad branch(\texttt{l1:+(l2:X,l3:Y)}, \texttt{l2}, \ldots)) \\ & = & \varphi(\ \texttt{x3:+(x4:0,x4)}, \\ & & \quad branch(\texttt{l1:+(l2:X,l3:Y)}, \texttt{l2}, rule(\texttt{l4:+(l5:0,l6:U)} \to \texttt{l6}), \ldots)) \\ & = & \quad (\texttt{x3}, \texttt{v1:+(v2:0,v3:Z)} \to \texttt{v3}) \end{array}$$

Lemma 5 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an inductively sequential cGRS, f a defined operation, $\mathcal{T}_f$ a definitional tree of f, and g an admissible term graph whose root is labeled with f. If $\varphi(g, \mathcal{T}_f) = (p, R)$, then (i) in every rewriting derivation from g to a constructor-rooted term graph, a descendant of $g_{|p}$ is rewritten at the root, in one or more steps, into a constructor-rooted term graph ; (ii) $g_{|p}$ is a redex of g matched by the left-hand side of R ; (iii) $g_{|p}$ is an outermost redex of g. If $\varphi(g, \mathcal{T})$ is not defined, then g cannot be rewritten into a constructor-rooted term graph.

Proof In the sequel, $\sqsubset$ denotes the Nœtherian ordering defined by : $(g_1, \mathcal{T}_1) \sqsubset (g_2, \mathcal{T}_2)$ if and only if either the graph g_1 has fewer occurrences of defined operations than the graph g_2, or else $g_1 = g_2$ and $\mathcal{T}_1$ is a proper subtree of $\mathcal{T}_2$. The proof is by Nœtherian induction on $\sqsubset$. We consider the different cases of the definition of φ.

Base case : consider $(g, \mathcal{T})$ where g is an admissible operation-rooted term graph, $\mathcal{T} = rule(\pi \to r)$ and $\pi \to r$ is a variant of a rule of $\mathcal{R}$ such that $\pi \leq g$. By definition of φ, $\varphi(g, \mathcal{T})$ is defined and $\varphi(g, \mathcal{T}) = (\mathcal{R}\mathrm{oot}_g, \pi' \to r')$, where $\pi' \to r'$ is a variant of $\pi \to r$. (i) Since g is operation-rooted, every rewriting derivation from g to a constructor-rooted term graph must contain a rewriting step at node $\mathcal{R}\mathrm{oot}_g$. So in any rewriting derivation from g to a constructor-rooted term graph, a descendant of $g_{|\mathcal{R}\mathrm{oot}_g}$ must be rewritten at the root into a constructor-rooted term graph. (ii) By definition of φ, $\pi \leq g$ and since π' is a variant of π, $\pi' \leq g$. So $g_{|\mathcal{R}\mathrm{oot}_g}$ is a redex matched by the left-hand side of $\pi' \to r'$. (iii) It is obvious that $g_{|\mathcal{R}\mathrm{oot}_g}$ is an outermost redex of g.

Induction step : consider $(g, \mathcal{T})$ where g is an admissible operation-rooted term graph, $\mathcal{T} = branch(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, 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 assume that $\mathit{pattern}(\mathcal{T}) \leq g$, i.e., there exists a homomorphism $h : \pi \to g$. There are three cases to consider.

Case 1 : $g_{|h(o)}$ is constructor-rooted, say $\mathcal{L}_g(h(o)) = c$. Then, there are two possibilities. If there exists an $i \in 1..k$ such that $\mathit{pattern}(\mathcal{T}_i) \leq g$, then by definition of φ, $\varphi(g, \mathcal{T}) = \varphi(g, \mathcal{T}_i)$. By induction hypothesis, all the claims hold already for $\varphi(g, \mathcal{T}_i)$, so they also hold for $\varphi(g, \mathcal{T})$. Else, there exists no $i \in 1..k$ such that $\mathit{pattern}(\mathcal{T}_i) \leq g$, then the operation $\mathcal{L}_g(\mathcal{R}\mathrm{oot}_g)$ is incompletely defined by $\mathcal{R}$. So g cannot be rewritten into a constructor-rooted term graph and $\varphi(g, \mathcal{T})$ is not defined.

Case 2 : $g_{|h(o)}$ is operation-rooted, say $\mathcal{L}_g(h(o)) = f$. Let $\mathcal{T}'$ be a definitional tree of f. Then, either $\varphi(g_{|h(o)}, \mathcal{T}')$ is defined, or not.

First sub-case, suppose $\varphi(g_{|h(o)}, \mathcal{T}') = (p, R)$, where p is a node of $g_{|h(o)}$ and R is a variant of a rule of $\mathcal{R}$. By the definition of φ, $\varphi(g, \mathcal{T}) = (p, R)$. (i) By the definition of φ, $\mathit{pattern}(\mathcal{T}) \leq g$. By the definition of a definitional tree, for every rule $l \to r$ of $\mathcal{R}$ whose left-hand side may match g at the root, there exists a leaf $\mathit{rule}(l' \to r')$ of $\mathcal{T}$, where $l' \to r'$ is a variant of $l \to r$. By construction of a definitional tree, if $l \to r$ is a rule represented by a leaf of $\mathcal{T}$, then $\mathit{pattern}(\mathcal{T}) \leq l$, i.e., there exists a homomorphism $h' : \pi \to l$. By hypothesis, $\mathcal{T}$ is a *branch* pdt and not a *rule* pdt so $\mathit{pattern}(\mathcal{T}) < l$. This implies that l has a constructor symbol at node $h'(o)$. However, the case being considered assumes that g has a defined operation at node $h(o)$. Hence, there is no rule of $\mathcal{R}$ whose left-hand side matches g at the root, i.e., g is not a redex. Moreover, in any rewriting derivation from g that includes a rewriting step at the root, a descendant of $g_{|h(o)}$ must be rewritten into a constructor-rooted term. Since g is operation-rooted, in any rewriting derivation from g to a constructor-rooted term graph, a descendant of g is rewritten at the root, and consequently, a descendant of $g_{|h(o)}$ is rewritten into a constructor-rooted term graph. By the induction hypothesis, in any rewriting derivation from $g_{|h(o)}$ to a constructor-rooted term graph, a descendant of $(g_{|h(o)})_{|p}$ is rewritten into a constructor-rooted term graph. Since p is a node of $g_{|h(o)}$, then by Proposition 1, $(g_{|h(o)})_{|p} = g_{|p}$. So in any rewriting derivation from g to a constructor-rooted term graph, a descendant of $g_{|p}$ is rewritten into a constructor-rooted term graph. (ii) By induction hypothesis, $g_{|p}$ is a redex of $g_{|h(o)}$ matched by the left-hand side of R, thus $g_{|p}$ is a redex of g matched by the left-hand side of R. (iii) By the definition of φ, $\pi \leq g$, i.e., there exists a homomorphism $h : \pi \to g$. By the definition of a definitional tree, π is a pattern and o is a node of π. These conditions imply that there exists a path from $\mathcal{R}\mathrm{oot}_g$ to $h(o)$ along which all nodes are labeled with constructor symbols, except $\mathcal{R}\mathrm{oot}_g$ and $h(o)$. In cGRS, redexes occur only at nodes labeled with defined operations. We have just proved that g is not a redex. Thus, there is no redex in g above $h(o)$. By induction hypothesis, $g_{|p}$ is an outermost redex in $g_{|h(o)}$. So $g_{|p}$ is also an outermost redex in g.

Second sub-case, suppose $\varphi(g_{|h(o)}, \mathcal{T}')$ is not defined. Then, $\varphi(g, \mathcal{T})$ is not defined too. Moreover, by induction hypothesis, $g_{|h(o)}$ cannot be rewritten into a constructor-rooted term graph. However, as before, g is not a redex and in any rewriting derivation from g to a constructor-rooted term graph, a descendant of $g_{|h(o)}$ must be rewritten into a constructor-rooted term graph. So g cannot be rewritten into a constructor-rooted term graph.

Case 3 : $g_{|h(o)}$ is a variable. Then $\varphi(g, \mathcal{T})$ is not defined. On the other hand, by the definition of φ, $\pi \leq g$. Any rule whose left-hand side may match g at the root is represented by a leaf of $\mathcal{T}$. If $l \to r$ is a rule represented by a leaf of $\mathcal{T}$, then by construction of a definitional tree, $\mathit{pattern}(\mathcal{T}) \leq l$, i.e., there exists a homomorphism $h' : \pi \to l$. Thus, by the definition of a definitional tree, l has a constructor symbol at node $h'(o)$. However, the case being considered assumes that g has a variable at node $h(o)$. Hence, g is not a redex. Since in any rewriting derivation from g that includes a rewriting step at the root, a descendant of $g_{|h(o)}$ must be rewritten into a constructor-rooted term and $g_{|h(o)}$ is a variable, g has no constructor normal form.

□

Proposition 17 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRSand g an admissible graph. If $\Phi(g) = (p, R)$, then $g_{|p}$ is an outermost needed redex of g and g can be rewritten at node p with rule R. If $\Phi(g)$ is not defined, then g cannot be rewritten into a constructor graph.

Proof Follows immediately from Lemma 5 and the definition of Φ. □

From the previous proposition, we deduce that if an admissible graph g has a constructor normal form c and $\Phi(g) = (p, R)$, then every derivation from g to c must contain a step at node p with rule R. The following lemma states that given a derivation $D = g \overset{*}{\rightarrow} c$, performing the step computed by $\Phi(g)$ first leads to a new derivation of length less than (or equal to) the length of D.

Lemma 6 Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a cGRS, g an admissible non constructor graph and c a constructor graph, such that there exists a rewriting derivation $g \overset{*}{\rightarrow} c$ of length n. Then, there exists an admissible graph g' such that $g \rightarrow_{\Phi} g'$ and a rewriting derivation $g' \overset{*}{\rightarrow} c'$ of length n' such that $n' < n$ and c' is a constructor graph with $c \sim c'$.

Proof This proof is sketched in Figure 25. By hypothesis, there exists a rewriting derivation

$$\begin{tikzcd}
g \arrow[r, "{[u_1,R_1]}"'] \arrow[dd, "{[p,R]}"] & a_1 \arrow[r, "{[u_2,R_2]}"'] \arrow[d, "{[p,R]}"] & \cdots \arrow[r, "{[u_{k-1},R_{k-1}]}"'] & a_{k-1} \arrow[r, "{[p,R]}"'] \arrow[d, "{[p,R]}"] & a_k \arrow[r, "{[u_{k+1},R_{k+1}]}"'] \arrow[dd, phantom, "\sim"] & \cdots \arrow[r, "{[u_n,R_n]}"'] & c \arrow[dd, phantom, "\sim"] \\
 & b_1 \arrow[d, phantom, "\sim"] & & b_{k-1} \arrow[d, phantom, "\sim"] & & \cdots & \\
g' \arrow[r, "\varepsilon", "{[u_1,R_1]}"'] & a'_1 \arrow[r, "\varepsilon", "{[u'_2,R_2]}"'] & \cdots \arrow[r, "\varepsilon", "{[u'_{k-1},R_{k-1}]}"'] & a'_{k-1} \arrow[r, phantom, "="] & a'_{k-1} \arrow[r, "{[u'_{k+1},R_{k+1}]}"'] & \cdots \arrow[r, "{[u'_n,R_n]}"'] & c'
\end{tikzcd}$$

Figure 25:

from g to a constructor graph c. So by Proposition 17, $(p, R) = \Phi(g)$ exists, g can be rewritten at node p with rule R and in any rewriting derivation from g to c, a descendant of $g_{|p}$ is rewritten into a constructor-rooted term graph. Thus there exists a graph g' such that $g \rightarrow_{[p,R]} g'$ and the derivation B is of the form $B = g \rightarrow_{[u_1,R_1]} a_1 \rightarrow_{[u_2,R_2]} \cdots \rightarrow_{[u_{k-1},R_{k-1}]} a_{k-1} \rightarrow_{[p,R]} a_k \rightarrow_{[u_{k+1},R_{k+1}]} \ldots \rightarrow_{[u_n,R_n]} c$, with $u_i \neq p$ for all $i \in 1..(k-1)$ and $i \in (k+1)..n$.

First, according to Lemma 2, since $g \rightarrow_{[u_1,R_1]} a_1$, $g \rightarrow_{[p,R]} g'$, there exist two graphs b_1 and a'_1 such that $a_1 \overset{\varepsilon}{\rightarrow}_{[p,R]} b_1$, $g' \overset{\varepsilon}{\rightarrow}_{[u_1,R_1]} a'_1$, and $b_1 \sim a'_1$. Since $g_{|p}$ is a needed redex of g, and $p \neq u_1$, p is a node of a_1, so $a_1 \neq b_1$ and $a_1 \rightarrow_{[p,R]} b_1$. More generally, according to Proposition 12, if $g \rightarrow_{[u_1,R_1]} a_1 \ldots \rightarrow_{[u_{k-1},R_{k-1}]} a_{k-1}$ and $g \rightarrow_{[p,R]} g'$, then there exist two graphs b_{k-1}, a'_{k-1} such that $a_{k-1} \overset{\varepsilon}{\rightarrow}_{[p,R]} b_{k-1}$, $g' \overset{\varepsilon}{\rightarrow}_{[u_1,R_1]} \cdots \overset{\varepsilon}{\rightarrow}_{[u'_{k-1},R_{k-1}]} a'_{k-1}$, and $b_{k-1} \sim a'_{k-1}$. In this second derivation, u'_i is the node corresponding to u_i by renaming (when it exists). Since $g_{|p}$ is a needed redex of g and $p \notin \{u_1, \ldots, u_{k-1}\}$, p is a node of a_{k-1} so $a_{k-1} \rightarrow_{[p,R]} b_{k-1}$. The length of the derivation $g \rightarrow_{[u_1,R_1]} \cdots \rightarrow_{[u_{k-1},R_{k-1}]} a_{k-1}$ is exactly $k-1$, whereas the length of the derivation $g' \overset{\varepsilon}{\rightarrow}_{[u_1,R_1]} \ldots \overset{\varepsilon}{\rightarrow}_{[u'_{k-1},R_{k-1}]} a'_{k-1}$ is less or equal to $k-1$. Second, $a_{k-1} \rightarrow_{[p,R]} a_k$ and $a_{k-1} \rightarrow_{[p,R]} b_{k-1}$, hence $a_k \sim b_{k-1}$ and since $a'_{k-1} \sim b_{k-1}$, $a'_{k-1} \sim a_k$. Last, $a_k \rightarrow_{[u_{k+1},R_{k+1}]} \cdots \rightarrow_{[u_n,R_n]} c$ and $a'_{k-1} \sim a_k$, so it is clear that there exists a constructor graph c' such $c \sim c'$ and $a'_{k-1} \rightarrow_{[u'_{k+1},R_{k+1}]} \ldots \rightarrow_{[u'_n,R_n]} c'$. The length of the derivation B is exactly n, whereas the length of the derivation $g' \overset{\varepsilon}{\rightarrow}_{[u_1,R_1]} \cdots \overset{\varepsilon}{\rightarrow}_{[u'_{k-1},R_{k-1}]} a'_{k-1} \rightarrow_{[u'_{k+1},R_{k+1}]} \cdots \rightarrow_{[u'_n,R_n]} c'$ is less or equal to $n-1$.

□

Theorem 3 The strategy Φ is c-hyper-normalizing strategy (and thus c-normalizing).

$$\begin{tikzcd}
g_0 \arrow[r, "*"] \arrow[d, "n_0"] & g'_0 \arrow[r, "\Phi"'] \arrow[d, "n'_0 \leq n_0"] & g_1 \arrow[r, "*"] \arrow[d, "n_1 \leq n'_0 - 1"] & \cdots \arrow[r, "\Phi"'] & C' \arrow[d, "\leq 0"] \\
c_0 \arrow[r, phantom, "\sim"] & c'_0 \arrow[r, phantom, "\sim"] & c_1 \arrow[r, phantom, "\sim"] & \cdots \arrow[r, phantom, "\sim"] & C'
\end{tikzcd}$$

Figure 26:

Proof This proof is sketched in Figure 26. Let g_0 be an admissible graph, c_0 a constructor graph such that there exists a rewriting derivation $g_0 \xrightarrow{*} c_0$ of length n_0. We prove this theorem by induction on n_0. The case $n_0 = 0$ is straightforward. Let $n_0 > 0$ and suppose the theorem is true for all lengths $p < n_0$. Let D be a derivation starting with g which contains infinitely many times Φ-steps. D is of the form $g_0 \xrightarrow{*} g'_0 \rightarrow_\Phi g_1 \xrightarrow{*} g'_1 \rightarrow_\Phi g_2 \xrightarrow{*} \ldots$ By confluence (see Theorem 1), since $g_0 \xrightarrow{*} g'_0$, $g_0 \xrightarrow{*} c_0$ is of length n_0 and c_0 is a constructor normal form, there exists a constructor graph $c'_0 \sim c_0$ and a derivation $g'_0 \xrightarrow{*} c'_0$ of length $n'_0 \leq n_0$. Since $g'_0 \xrightarrow{*} c'_0$ of length n'_0, $g'_0 \rightarrow_\Phi g_1$, and c'_0 is a constructor graph, we deduce by Lemma 6 the existence of a constructor graph $c_1 \sim c'_0$ and a derivation $g_1 \xrightarrow{*} c_1$ of length $n_1 < n'_0$. Let D' be the sub-derivation of D starting with g_1. The derivation $g_1 \xrightarrow{*} c_1$ is of length $n_1 < n_0$ and D' contains infinitely many times Φ-steps (since D contains infinitely many times Φ-steps). So by induction hypothesis, D' ends with a constructor graph $C' \sim c_1$, thus D ends with the constructor graph $C' \sim c_0$.

□

Graph rewriting does not duplicate data. Thus the number of rewriting steps which are necessary to compute a constructor normal form may be optimized. We obtain the following result for the strategy Φ.

Theorem 4 Let g be an admissible graph and c a constructor graph such that there exists a rewriting derivation $g \xrightarrow{*} c$. Then there exists a constructor graph c' with $c' \sim c$ such that the length of the Φ-derivation $g \xrightarrow{*}_\Phi c'$ is less (or equal) to the length of the derivation $g \xrightarrow{*} c$.

Proof By induction on the length n of the derivation $g \xrightarrow{*} c$. The case $n = 0$ is straightforward. Let $n > 0$ and suppose the theorem is true for all lengths $p < n$. Thanks to Lemma 6, there exist an admissible graph g' such that $g \rightarrow_\Phi g'$, a constructor graph $c' \sim c$ and a rewriting derivation $g' \xrightarrow{*} c'$ of length $n' < n$. By induction hypothesis, there exist a constructor graph $c'' \sim c'$ and a Φ-derivation $g' \xrightarrow{*}_\Phi c''$ of length $n'' \leq n'$. The length of the derivation $g \rightarrow_\Phi g' \xrightarrow{*}_\Phi c''$ is $n'' + 1 \leq n' + 1 \leq n$ and $c'' \sim c$. □

7 Conclusion

We characterized the class of admissible graphs. For this class of graphs, we proved the classical confluence as well as the confluence modulo bisimilarity of the graph rewriting relation induced by orthogonal constructor-based GRSs, even in the presence of collapsing rules. We have also presented a new strategy for inductively sequential graph rewriting systems. This strategy has been defined precisely and proved to be c-normalizing and optimal for the class of admissible graphs. The use of definitional trees allows to combine the elegance of neededness with an efficient implementation by pattern-matching. In [KSvEP93], a lazy graph rewriting strategy close to ours is described, namely the annotated functional strategy, which combines the discriminating position strategy [PvE93] and rewriting with priority [BBK87]. At our knowledge, no formal result has been proved regarding this strategy.

The main motivation of this work was the definition of an operational semantics of functional and logic languages based on narrowing. The extension to narrowing of the rewriting strategy presented in this paper can be found in [EJ97].

Acknowledgment

The first author is very indebted to Sergio Antoy and Michael Hanus for the many fruitful discussions.

A Matching

Matching is essential for graph rewriting. We first recall its definition and then we give an algorithm.

Definition 30 (Matching)
Given a graph g, a term graph l and a node n of g, we say that l *matches* g *at node* n, denoted $l \leq g_{|n}$, if there exists a homomorphism $h : l \to g_{|n}$. h is called the *matcher* of l on g at node n.
$\Box$

Here is a matching algorithm : Given two term graphs g_1, g_2, it computes the relation on $\mathcal{N}_{g_1} \times \mathcal{N}_{g_2}$ which induces a function $h : \mathcal{N}_{g_1} \to \mathcal{N}_{g_2}$ such that h is a homomorphism from g_1 to g_2, when it exists, and fails otherwise. It begins with the pair $(\{\mathcal{R}\mathrm{oot}_{g_1} \rhd \mathcal{R}\mathrm{oot}_{g_2}\}, \emptyset)$, and ends with the pair $(\emptyset, h)$ in case of success, or $\Box$ in case of failure. The rules are :

$$\frac{\{n_1 \rhd n_2\} \cup \Delta, h}{\Delta, h \cup \{n_1 \mapsto n_2\}} \quad \text{if } n_1 \notin \mathcal{D}\mathrm{om}(h) \text{ and } \mathcal{L}_{g_1} n_1 \in \mathcal{X} \text{ ;}$$

$$\frac{\{n_1 \rhd n_2\} \cup \Delta, h}{\{u_1 \rhd v_1, \ldots, u_k \rhd v_k\} \cup \Delta, h \cup \{n_1 \mapsto n_2\}} \quad \begin{array}{l} \text{if } n_1 \notin \mathcal{D}\mathrm{om}(h), \\ \mathcal{L}_{g_1}(n_1) \notin \mathcal{X}, \\ \mathcal{L}_{g_1}(n_1) = \mathcal{L}_{g_2}(n_2), \\ \mathcal{S}_{g_1}(n_1) = u_1 \ldots u_k \text{ and} \\ \mathcal{S}_{g_2}(n_2) = v_1 \ldots v_k \text{ ;} \end{array}$$

$$\frac{\{n_1 \rhd n_2\} \cup \Delta, h}{\Box} \quad \begin{array}{l} \text{if } n_1 \notin \mathcal{D}\mathrm{om}(h), \\ \mathcal{L}_{g_1} n_1 \notin \mathcal{X} \text{ and} \\ \mathcal{L}_{g_1}(n_1) \neq \mathcal{L}_{g_2}(n_2) \text{ ;} \end{array}$$

$$\frac{\{n_1 \rhd n_2\} \cup \Delta, h}{\Delta, h} \quad \text{if } n_1 \in \mathcal{D}\mathrm{om}(h) \text{ and } h(n_1) = n_2 \text{ ;}$$

$$\frac{\{n_1 \rhd n_2\} \cup \Delta, h}{\Box} \quad \text{if } n_1 \in \mathcal{D}\mathrm{om}(h) \text{ and } h(n_1) \neq n_2.$$

References

[AEH+96] Marc Andries, Gregor Engels, Annegret Habel, Berthold Hoffmann, Hans-Jörg Kreowski, Sabine Kuske, Detlef Plump, Andy Schürr, and Gabriele Taentzer. Graph transformation for specification and programming. *??*, 1996. In preparation.

[AK96] Z.M. Ariola and J.W. Klop. Equational term graph rewriting. *Fundamenta Informaticae*, 26(3-4), 1996.

[Ant92] S. Antoy. Definitional trees. In *Proc. of the 4th Intl. Conf. on Algebraic and Logic programming*, pages 143–157. Springer Verlag LNCS 632, 1992.

[BBK87] J.C. Baeten, J.A. Bergstra, and J.W. Klop. Term rewriting systems with priorities. In *Proc. of Conference on Rewriting Techniques and Applications*, pages 83–94, Bordeaux, 1987. Springer Verlag LNCS 256.

[BvEG$^+$87] H. Barendregt, M. van Eekelen, J. Glauert, R. Kenneway, M. J. Plasmeijer, and M. Sleep. Term graph rewriting. In *PARLE'87*, pages 141–158. Springer Verlag LNCS 259, 1987.

[Cou93] B. Courcelle. Réécriture de graphes: orientation bibliographique. *Available via URL : http://www.labri.u-bordeaux.fr/ courcell/ActSci.html*, 1993.

[EJ97] R. Echahed and J. C. Janodet. Introducing graphs in functional logic programming languages. Technical report, IMAG, 1997. In preparation.

[ET96] H. Ehrig and G. Taentzer. Computing by graph transformation : A survey and annotated bibliography. *Bulletin of the EATCS*, 59:182–226, June 1996.

[HL91] G. Huet and J.-J. Lévy. Computations in orthogonal term rewriting systems. In J.-L. Lassez and G. Plotkin, editors, *Computational logic: essays in honour of Alan Robinson.* MIT Press, Cambridge, MA, 1991. Previous version: Call by need computations in non-ambiguous linear term rewriting systems, Technical Report 359, INRIA, Le Chesnay, France, 1979.

[HP95] A. Habel and D. Plump. Unification, rewriting, and narrowing on term graphs. *Electronic notes in Theoretical Computer Science 1*, 1995. Available via URL : http://www.elsevier.nl/locate/tcs.

[Hue80] Gérard Huet. Confluent reductions: Abstract properties and application to term rewriting systems. *JACM*, 27(4):797–821, 1980.

[KKSV94] 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. Previous version: Technical Report CS-R9204, CWI, Amsterdam, 1992.

[KKSV95] J. R. Kennaway, J. K. Klop, M. R. Sleep, and F. J. De Vries. Transfinite reduction in orthogonal term rewriting systems. *Information and Computation*, pages 18–38, 1995.

[Klo92] J. W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. II*, pages 1–112. Oxford University Press, 1992. Previous version: Term rewriting systems, Technical Report CS-R9073, Stichting Mathematisch Centrum, Amsterdam, 1990.

[KSvEP93] P.W. Koopman, J.E. Smetsers, M.C. van Eekelen, and M.J. Plasmeijer. Graph rewriting using the annotated functional strategy. In Ronan Sleep, Rinus Plasmeijer, and Marko van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 23, pages 317–332. John Wiley, New York, 1993.

[O'D77] M. J. O'Donnell. *Computing in Systems Described by Equations.* Springer Verlag LNCS 58, 1977.

[PvE93] R. Plasmeijer and M. van Eekelen. *Functional Programming and Parallel Graph Rewriting.* Addison-Wesley, 1993.

[SPvE93] 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.