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. Echahe, 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. [PV93, 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 $F(a, a, x) \rightarrow x$ 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

![Graph Rewrite System](image)

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 (1, x, r)) -> 1
right-tree (bin-tree (1, 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 \(\Sigma = (S, \Omega)\) consists of a set \(S\) of sorts and an \(S\)-indexed family \(\Omega = \{\Omega_s\}_{s \in S}\) with \(\Omega_s = \{\omega_{w, a}\}_{w, a \in S \times S}\) of sets of operation symbols. We shall write \(f : s_1 \ldots s_n \rightarrow s\) whenever \(f \in \Omega_{s_1 \ldots \ast_n}\), and say that \(f\) is of sort \(s\), rank \(s_1 \ldots s_n\) and profile \(s_1 \ldots s_n, s\). □

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 \(X = \{\mathcal{X}_s\}_{s \in S}\) be an \(S\)-indexed family of countable sets of variables and \(N = \{\mathcal{N}_s\}_{s \in S}\), an \(S\)-indexed family of countable sets of nodes. Both \(X\) and \(N\) are fixed throughout the rest of the paper.

Definition 2 (Graph)
A (rooted) graph \(g\) over \((\Sigma, N, X)\) is a tuple \(g = (N_g, \mathcal{L}_g, S_g, \mathrm{Root}_g)\) such that:
1. $\mathcal{N}_g$ is the set of nodes of $g$, i.e., $\mathcal{N}_g = \bigcup_{s \in S} (\mathcal{N}_g)_s$ 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., $\mathcal{L}_g = \bigcup_{s \in S} (\mathcal{L}_g)_s$ 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., $\mathcal{S}_g = \bigcup_{s \in S} (\mathcal{S}_g)_s$ 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. $\text{Roots}_g$ is a nonempty set of distinguished nodes of $g$, called its roots: $\text{Roots}_g \subseteq \mathcal{N}_g$.

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 $\text{Root}_g$ (i.e., $\text{Roots}_g = \{\text{Root}_g\}$). We write $V(g)$ for the set of variables appearing in $g$. \hfill $\square$

**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 $\text{Root}_G = \{x_1\}$. Nodes of $G$ are $\mathcal{N}_G = \{x_1, \ldots, x_5\}$. Its labeling function is defined by $\mathcal{L}_G(x_1) = \mathcal{L}_G(x_5) = \text{cons}$, $\mathcal{L}_G(x_2) = +$, $\mathcal{L}_G(x_3) = \text{succ}$, et $\mathcal{L}_G(x_4) = 0$. Its successor function is defined by $\mathcal{S}_G(x_1) = x_2.x_5$, $\mathcal{S}_G(x_2) = x_3.x_3$, $\mathcal{S}_G(x_3) = x_4$, $\mathcal{S}_G(x_4) = \varepsilon$, $\mathcal{S}_G(x_5) = x_3.x_1$. The right-hand graph, $T$ represents the classical rewriting rule $\text{succ}(x) + y \to \text{succ}(x+y)$ (cf. definition 20). $T$ has two roots $\text{Roots}_T = \{11, r_1\}$.

![Figure 2: Two examples of graphs.](image)

**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: $u_0 = n_1$, $u_k = n_2$, $k \leq 1$, $u_i \in N_g$ 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 $P_g(n_1, n_2)$ 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 $P_g(n_1, n_2) \neq \emptyset$.

By definition of a graph $g$, the connectivity condition says that each node is either a root or is accessible from a root: $\forall n \in N_g, n \notin \text{Roots}_g \Rightarrow \exists r \in \text{Roots}_g$ such that $P_g(r, n) \neq \emptyset$. \hfill \qed

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 $\Omega \cup X$ (resp. $N$). The linear notation of a graph is defined by:

- **Graph** ::= $\text{Node} | \text{Node} + \text{Graph}$
- **Node** ::= $n:A(\text{Node}, \ldots, \text{Node}) | n$

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 +. \hfill \qed

**Example 2** With this syntax, the graphs $G$ and $T$ of figure 2 are written:

$G = x1: \text{cons}(x2:+(x3: \text{succ}(x4:0), x3), x5: \text{cons}(x3, x1)), T = 11:+(12: \text{succ}(13:x), 14:y) + r1: \text{succ}(r2:+(13, 14))$.

**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:

- $N_{g_{|P}}$ is the set of nodes of $g$ accessible from nodes of $P$, i.e., $N_{g_{|P}} = P \cup \{n \in N_g | \exists r \in P; P_g(r, n) \neq \emptyset\}$.
- For all nodes $n$ of $g_{|P}$, $L_{g_{|P}}(n) = L_g(n)$, i.e., $L_{g_{|P}}$ is the restriction of $L_g$ to $N_{g_{|P}}$.
- For all nodes $n$ of $g_{|P}$, $S_{g_{|P}}(n) = S_g(n)$, i.e., $S_{g_{|P}}$ is the restriction of $S_g$ to $N_{g_{|P}}$.
- $\text{Roots}_{g_{|P}} = P$.

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 $G_{|x2} = x2:+(x3: \text{succ}(x4:0), x3)$ and the subgraph of $T$ rooted by $\{13, 14\}$ is $T_{|\{13, 14\}} = 13:x + 14:y$.

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.
Otherwise, the graph obtained after pointer redirection would not be connected. Finally, we ignore all uninteresting nodes such as $x_2$ (garbage collection). The result of the replacement by $D$ of the subgraph of $G$ at node $x_2$ is denoted $G[x_2 \leftarrow D]$.

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 $n \in N_{g_1} \cap N_{g_2}$, $L_{g_1}(n) = L_{g_2}(n)$ and $S_{g_1}(n) = S_{g_2}(n)$.

2. Any variable appearing both in $g_1$ and $g_2$ labels a same node: For all $x \in V(g_1) \cap V(g_2)$, if $L_{g_1}(n) = x$ and $L_{g_2}(m) = x$, then $n = m$.

**Example 4** Consider the graphs $g_1, g_2$ and $g_3$ defined by $g_1 = x_1:f(x_2:0), g_2 = x_2:f(x_1:0)$ and $g_3 = x_3:0$. The node $x_1$ 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:

- $N_{g_1 + g_2} = N_{g_1} \cup N_{g_2}$.
- $L_{g_1 + g_2} = L_{g_1} \cup^1 L_{g_2}$.
- $S_{g_1 + g_2} = S_{g_1} \cup^1 S_{g_2}$.
- $\text{Roots}_{g_1 + g_2} = \text{Roots}_{g_1} \cup \text{Roots}_{g_2}$.

---

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 $L_{g_1 \oplus g_2}$ and $S_{g_1 \oplus g_2}$ 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 $\rho : N_g \rightarrow N_g$ 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 $\rho$ as the graph $\rho(g)$ such that all nodes having $p$ as successor in $g$ have $q$ as successor in $\rho(g)$:

- $N_{\rho(g)} = N_g$.
- $L_{\rho(g)} = L_g$.
- For all nodes $n$, if $S_g(n) = \varepsilon$, then $S_{\rho(g)}(n) = \varepsilon$, else, if $S_g(n) = n_1 \ldots n_k$, then $S_{\rho(g)}(n) = \rho(n_1) \ldots \rho(n_k)$.
- $\text{Roots}_{\rho(g)} = \{p\} \cup \rho(\text{Roots}_g)$, that is to say if $\text{Roots}_g = \{n_1, \ldots, n_k\}$, then $\text{Roots}_{\rho(g)} = \{p, \rho(n_1), \ldots, \rho(n_k)\}$.

□

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 $p \in \text{Roots}_{\rho(g)}$. When $p \notin N_g$, $\rho(g) = g$.

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 $\text{Root}_u$. 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 $\rho$ be the pointer redirection, defined on $H$, such that $\rho(p) = \text{Root}_u$ and $\rho(n) = n$ for all $n \neq p$. Let $H' = \rho(H)$ and $Q = \rho(\text{Roots}_g)$.

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 $D = r1:\text{succ}(r2:+(x4:0,x3:\text{succ}(x4)))$ of the subgraph rooted by $x2$ in the graph $G = x1:\text{cons}(x2:+(x3:\text{succ}(x4:0),x3),x5:\text{cons}(x3,x1))$. The result is the graph $G[x2 \leftarrow D] = x1:\text{cons}(r1:\text{succ}(r2:+(x4:0,x3:\text{succ}(x4))),x5:\text{cons}(x3,x1))$.

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

**Example 6** Let $g = 11:+(12:0,13:0)$, $d = r1:+(11:+(12:0,13:0),r2:0)$, and $p = 11$. Then the reader may check that $g[p \leftarrow d] = r1:+(r1,x2:0)$, 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 $d$.  

6
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., \( P_\prec(n_1, n_2) \neq \emptyset \). 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 \( n_1 \parallel n_2 \), iff there exists no path from \( n_1 \) to \( n_2 \) or from \( n_2 \) to \( n_1 \) in \( g \), i.e., \( (n_1 \preceq n_2 \lor n_2 \preceq n_1) \), or equivalently, \( P_\parallel(n_1, n_2) = P_\parallel(n_2, n_1) = \emptyset \). 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 \( g|_{n_1} \parallel g|_{n_2} \).

**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 \( g|_{n_1} \leftarrow u \parallel g|_{n_2} \). This proposition holds in particular when \( n_1 \parallel n_2 \).

**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 \( n_1 \not\in \mathcal{N}_u \) and \( n_2 \in \mathcal{N}_u \), then \( g|_{n_1} \leftarrow u \parallel u|_{n_2} \).

From the example 6, it appears that an expression such as \( g|_{n_1} \leftarrow u_1 \parallel g|_{n_2} \leftarrow u_2 \) 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 \( g = x_1:f(x_2:g(x_3:0)) \), \( u_1 = x_4:1 \) and \( u_2 = x_5:h(x_2:g(x_3:0)) \) be three compatible term graphs. Then \( g|_{x_3} \leftarrow u_1 = x_1:f(x_2:g(x_4:1)) \). This graph is no more compatible with \( u_2 \), since \( x_2 \) has not the same successors in \( g|_{x_3} \leftarrow u_1 \) as in \( u_2 \). The problem is that \( x_3 \) 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 \( n_1 \not\in \mathcal{N}_u_2 \), then \( g|_{n_1} \leftarrow u_1 \) and \( u_2 \) are compatible, thus \( g|_{n_1} \leftarrow u_1 \parallel u_2 \) 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 \( S_{g+u_1}(p) = S_{u_2}(p) \) but \( S_{g[n_1\leftarrow u_1]}(p) \neq S_{u_2}(p) \). Suppose \( S_{g+u_1}(p) = p_1 \ldots p_k \). Let \( \rho \) be the pointer redirection such that \( \rho(n_1) = \text{Root}_{u_1} \) and \( \rho(n) = n \) for all \( n \neq n_1 \). Then, by definition of replacement, \( S_{g[n_1\leftarrow u_1]}(p) = \rho(p_1) \ldots \rho(p_k) \). Since \( S_{u_2}(p) = p_1 \ldots p_k \) and \( S_{g[n_1\leftarrow u_1]}(p) \neq S_{u_2}(p) \), 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 \( S_{u_2}(p) = p_1 \ldots p_k \), 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 \( n_1 \not\in \mathcal{N}_u_2 \). 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 \( \rho_1(n_1) = \text{Root}_{u_1} \) and \( \rho_2(n_2) = \text{Root}_{u_2} \) 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
\( (g[n_1 \leftarrow u_1][n_2 \leftarrow u_2] = (\rho_2(\rho_1(g + u_1 + u_2)))|_{\mathcal{N}_{u_2}}) \).

2. Let \( \rho_3(n_1) = \rho_2(\text{Root}_{u_1}) \) 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
\( (g[n_2 \leftarrow u_2][n_1 \leftarrow u_1[n_2 \leftarrow u_2]] = (\rho_3(\rho_2(g + u_1 + u_2)))|_{\mathcal{N}_{u_2}}) \).

3. Finally, we can show that for all \( p \in \mathcal{N}_{g+n_1+u_2} \), \( \rho_2(\rho_1(p)) = \rho_3(\rho_2(p)) \). This proof is done by analysis of the different cases for \( p \).

\[ \square \]

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 \( u_1 \). \( \square \)

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.,
\( \forall r \in \text{Root}_{u_2}, \forall C \in \mathcal{P}_g(r, n_1), n_2 \in C \). 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. \( \square \)

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 \( h = \psi_{a \in \mathcal{A}} h_a, h_a: (\mathcal{N}_{g_1})_a \to (\mathcal{N}_{g_2})_a \) 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 \( n \in (\mathcal{N}_{g_1})_a, (\mathcal{L}_{g_1})_a(n) \notin \mathcal{X}, \)

   - \( (\mathcal{L}_{g_1})_a(h_a(n)) = (\mathcal{L}_{g_2})_a(n) \).
   - \( (\mathcal{S}_{g_2})_a(h_a(n)) = \psi_{a \in \mathcal{A}}(\mathcal{S}_{g_1})_a(n) \), that is to say for all non variable nodes \( n \), if \( (\mathcal{S}_{g_1})_a(n) = \varepsilon \), then \( (\mathcal{S}_{g_2})_a(h_a(n)) = \varepsilon \), else if \( (\mathcal{S}_{g_1})_a(n) = n_1 \ldots n_k \), where \( n_i \in (\mathcal{N}_{g_1})_{a_i} \) for all \( i \in 1..k \), then \( (\mathcal{S}_{g_2})_a(h_a(n)) = h_{a_1}(n_1) \ldots h_{a_k}(n_k) \).

2. For all nodes \( n \in (\mathcal{N}_{g_1})_a, (\mathcal{L}_{g_2})_a(n) \notin \mathcal{X}, h_a(n) \in (\mathcal{N}_{g_2})_a \).

3. The images of roots of \( g_1 \) are the roots of \( g_2 \): \( \text{Roots}_{g_2} = h(\text{Roots}_{g_1}) \), i.e., if \( \text{Roots}_{g_1} = \{ n_1, \ldots, n_k \} \), where \( n_i \in (\mathcal{N}_{g_1})_{a_i} \) for all \( i \in 1..k \), then \( \text{Roots}_{g_2} = \{ h_{a_i}(n_i) | i \in 1..k \} \).

8
We say that \( h \) is a variable-preserving homomorphism if \( h \) is a homomorphism between two graphs with variables treated as constants, i.e., \( \mathcal{L}_{g_1}(h(n)) = \mathcal{L}_{g_2}(n) \) 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 \( L = 11:4(12:succ(13:x),14:y) \) to the subgraph of \( G \) rooted by \( x_2 \). Formally, \( h \) is a mapping from \( \mathcal{N}_L \) to \( \mathcal{N}_{G[x_2]} \) defined by \( h(11) = x_2, h(12) = h(14) = x_3 \) and \( h(13) = x_4 \). \( h \) is not a variable-preserving homomorphism since \( \mathcal{L}_L(13) = x \) whereas \( \mathcal{L}_G(h(13)) = \mathcal{L}_G(x_4) \neq x \).

![Homomorphism Example](image)

Figure 4: Example of homomorphism.

Given three graphs \( g_1, g_2 \) and \( g_3 \) and two homomorphisms \( h_1 : g_1 \rightarrow g_2 \) and \( h_2 : g_2 \rightarrow g_3 \), by the composition of \( h_1 \) and \( h_2 \), denoted \( h_2 \circ h_1 \), we mean the mapping \( h : \mathcal{N}_{g_1} \rightarrow \mathcal{N}_{g_3} \) 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, \( h : g_1 \rightarrow g_2 \) 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 \( h : l \rightarrow g|_n \). \( h \) 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 \( x_2 \).

Finally, a precise definition of a rewriting step requires the notion of “application” of a homomorphism \( h : g_1 \rightarrow g_2 \) 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 \( h : L \rightarrow G[x_2] \) of example 8. Let \( R = r1:succ(r2:*(13:x,14:y)) \). Computing \( h(R) \) consists in replacing all subgraphs shared
between $R$ and $L$ by their corresponding subgraphs in $G_{|x|2}$. Here, $R$ and $L$ share subgraphs $13\cdot x$ and $14\cdot y$. Their corresponding graphs by $h$ in $G$ are respectively $x4\cdot 0$ and $x3\cdot \text{succ}(x4\cdot 0)$. Hence, we obtain a graph $h(R) = r1\cdot \text{succ}(r2\cdot +(x4\cdot 0, x3\cdot \text{succ}(x4)))$, 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 $B(g, g') = \{ p \in N_g \cap N_{g'} \mid p \in \text{Roots}_{g} \text{ or } \exists q \in (N_g - N_{g'}), p \in S_q(g) \}$. □

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

**Definition 14** (Application of a homomorphism on a graph)

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

$$h(g) = g[p_1 \leftarrow g_2[p_1] \ldots p_k \leftarrow g_2[p_k]]$$

Thanks to proposition 6, the condition $B \cap N_{g_2} = \emptyset$ is sufficient to insure the well definedness of the expression $g[p_1 \leftarrow g_2[p_1] \ldots p_k \leftarrow g_2[p_k]]$. □

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

### 2.4 Graph equalities

Homomorphisms are also used to define equality relations on graphs. We already used the syntactic equality, denoted $=\cdot$, 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 $\phi : g_1 \rightarrow g_2$ 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 $\alpha$-conversion in $\lambda$-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 $g_1 \equiv g_2$, if there exists a graph $g'$, and two variable-preserving homomorphisms $h_1 : g_1 \rightarrow g'$ and $h_2 : g_2 \rightarrow g'$. □

**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].
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 \( \Sigma \), we mean a triple \( \Sigma = (S, \mathcal{C}, \mathcal{D}) \) 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 \( \langle S, \mathcal{C} \cup \mathcal{D} \rangle \) 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 \( n \in N_g \), if \( L_g(n) \in \mathcal{D} \), then \( P_g(n, n) = \emptyset \). [\( \square \)]

**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. [\( \square \)]
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 \textit{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.

\textbf{Definition 20}  (Pattern) 
A term graph \( g \) is a \textit{pattern} if:

1. It is operation-rooted : \( \mathcal{L}_g(\text{Root}_g) \in \mathcal{D} \).
2. All its other nodes are labeled either with a constructor symbol or with a variable : For all \( n \in \mathcal{N}_g \), if \( n \neq \text{Root}_g \), then \( \mathcal{L}_g(n) \in \mathcal{X} \) or \( \mathcal{L}_g(n) \in \mathcal{C} \).
3. There is no cycle including the root : \( \mathcal{P}_g(\text{Root}_g, \text{Root}_g) = \emptyset \).
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 \( n \neq \text{Root}_g \), then \( \text{cardinal}(\{ m \in \mathcal{N}_g \mid n \in \mathcal{S}_g(m) \}) = 1 \).

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. \( \square \)

\textbf{Example 16}  \( 11 : +(12:0,13:0) \) is a pattern, whereas \( 11 : +(12:0,12) \) is not.

\textbf{Definition 21}  (Admissible rewrite rule) 
An \textit{admissible rewrite rule} is a rewrite rule \( e = l \rightarrow r \) 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. \( \mathcal{V}(r) \subseteq \mathcal{V}(l) \).

\( \square \)

We say that two admissible rewrite rules are \textit{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.

\textbf{Definition 22}  (cGRS) 
A \textit{constructor-based graph rewriting system (cGRS)} is a pair \( SP = (\Sigma, \mathcal{R}) \) such that \( \Sigma = (S, \mathcal{C}, \mathcal{D}) \) 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. \( \square \)

\textbf{Example 17}  Figure 7 shows the set of admissible rewrite rules associated to the \textit{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].
Definition 23 (Rewriting step) Let \( SP = (\Sigma, R) \) 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 \( R \) and \( p \) a node of \( g_1 \). We say that \( g_1 \) \textit{rewrites to} \( g_2 \) \textit{at node} \( p \) \textit{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 \textit{redex} of \( g_1 \) rooted by \( p \). A graph is in \textit{normal form} if it has no redex. A constructor graph is necessarily in normal form. \( \rightarrow^* \) denotes the reflexive closure of \( \rightarrow \) and we say that \( g_1 \) is rewritten \textit{in at most one step}\(^2\) into \( g_2 \) whenever \( g_1 \rightarrow^* g_2 \). \( \rightarrow^* \) denotes the transitive closure of \( \rightarrow \) and \( \rightarrow^* \), its transitive and reflexive closure. We speak of a \textit{derivation} from \( g_1 \) to \( g_2 \) whenever \( g_1 \rightarrow^* g_2 \).

Example 18 Consider the graph \( G \) and the rule \( T = L \rightarrow R \) of figure 2, where \( L = 11:* (12: succ(13:x,14:y)) \) and \( R = r1 : succ(r2:* (13:x,14:y)) \). 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, \( G \rightarrow_{[x2, l \rightarrow R]} G' \).

We now prove the well definedness of a rewriting step.

Proposition 8 Let \( SP = (\Sigma, R) \) be a cGRS, \( g_1 \) an admissible graph, \( e = l \rightarrow r \) a variant of a rewrite rule of \( R \) and \( p \) 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, \( l \) 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, \( B(l,r) \subseteq \mathcal{N}_{l \rightarrow r} \) so \( B(l,r) \cap \mathcal{N}(g_1|_p) = \emptyset \). Hence \( h(r) \) is well defined. Let \( B(l,r) = \{p_1, \ldots, p_k\} \). 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.

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.

\[^2\text{This notation is introduced in [Hue80]. In [Klo92], it is written } \rightarrow^\ast.\]
Lemma 1  Let $SP = (\Sigma, \mathcal{R})$ 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 $\text{Root}_u \not\subset p$ in $g + u$. Thus, the pointer redirection from $p$ to $\text{Root}_u$ does not modify $u$ to create a cycle on a node labeled with a defined operation. □

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

Proof  Let $h : l \rightarrow g_1[p]$ 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 $P = B(l, r) = \{p_1, \ldots, p_k\}$ then $h(r) = r[p_1 \leftarrow (g_1[p_1])[h(p_1)] \cdots [p_k \leftarrow (g_1[p_k])[h(p_k)]]$. 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 $p \notin \mathcal{N}_g$, and $\mathcal{N}_r \cap \mathcal{N}_g = \emptyset$. Thus if $p$ appears in $h(r)$, then $p$ is a node of $(g_1[p])[h(p)]$ for some $i$.

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

Example 19  Consider the following rewrite rule $11:+\langle 12:0, 13:x \rangle \rightarrow r1:+\langle 11, r2:0 \rangle$. 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 $x1:+\langle x2:0, x3:0 \rangle$ rewrites to $r1:+\langle r1, r2:0 \rangle$. 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 $SP = (\Sigma, \mathcal{R})$ be a cGRS. We say that the rewriting relation $\xrightarrow{\ast}$ 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.
Theorem 1 Let \( SP = \langle \Sigma, \mathcal{R} \rangle \) be a cGRS. Then the rewriting relation \( \rightarrow \) 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 \( g_1 \rightarrow g'_1 \) (resp. \( g_1 \overset{\varepsilon}{\rightarrow} g'_1 \)), then there exists an admissible graph \( g'_2 \) such that \( g_2 \rightarrow g'_2 \) (resp. \( g_2 \overset{\varepsilon}{\rightarrow} g'_2 \)) and \( g'_1 \) and \( g'_2 \) are equal up to renaming of nodes \( (g_1 \sim_\beta g'_2) \). (see Figure 9).

![Figure 9:](image)

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 \( g \rightarrow g_1 \) and \( g \rightarrow g_2 \), then there exist two graphs \( g'_1 \) and \( g'_2 \) such that \( g_1 \rightarrow g'_1, g_2 \rightarrow g'_2 \) and \( g'_1 \) and \( g'_2 \) are equal up to renaming of nodes \( (g'_1 \sim_\beta g'_2) \). (see Figure 10).

**Proof** This proof is done by case analysis. Let \( g, g_1 \) and \( g_2 \) be three admissible graphs such that \( g \rightarrow_{[n_1, l_1 \rightarrow r_1]} g_1 \) and \( g \rightarrow_{[n_2, l_2 \rightarrow r_2]} g_2 \). By definition of a rewriting step, there exist a matcher \( h_1 : l_1 \rightarrow g|_{n_1} \) such that \( g_1 = g[n_1 \leftarrow h_1(r_1)] \) and a matcher \( h_2 : l_2 \rightarrow g|_{n_2} \) such that \( g_2 = g[n_2 \leftarrow h_2(r_2)] \).

If \( n_1 = n_2 \), then necessarily \( l_1 \rightarrow r_1 \) is a variant of \( l_2 \rightarrow r_2 \) 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
$g'_2 = g_2$.

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., $n_1 \parallel n_2$) 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., $n_1 \parallel n_2$):

By definition of $g_1$, $g_{1|n_2} = (g[n_1 \leftarrow h_1(r_1)])_{n_2}$. Since $n_1 \parallel n_2$, 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 $l_2 \rightarrow r_2$ 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, $N_2 \cap N_{r_1} = \emptyset$, 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, $g'_2 = g'_2$.

**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 $p_{l_1}$ of $l_1$ such that $n_2$ is a node of $g_{h_1(r_1)}$. The existence of $p_{l_1}$ is due to the fact that $l_1$ is a pattern, and $h_1(\text{Root}_{l_1}) = n_2 \neq n_2$. There may be other variable nodes in $l_1$ like the node $p_{l_1}$. Let $P = \{p_1, p_2, \ldots, p_k\}$ be the set of variable nodes $p_{l_1}$ of $l_1$ such that $n_2$ is a node of $g_{h_1(r_1)}$. There are two sub-cases to study, depending on $P \cap N_{r_1} = \emptyset$ or not, i.e., whether or not at least one variable node of $P$ belongs to the right-hand side of $r_1$.

**Case 2.1**: $(P \cap N_{r_1} = \emptyset)$, 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 $\text{Root}_{h_1(r_1)}$ because $n_1 \notin h_1(r_1)$ (cf. lemma 1).

Let us show that $g_{1|n_2} = g_{n_2}$. Since $n_1 \notin N_{h_1(r_1)}$ and $n_2 \in N_{h_1(r_1)}$, 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 $l_2 \rightarrow r_2$ 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 10: Confluence : Lemma 2
We have just defined $g'$. We now define $g''$. By hypothesis, $n_1 \prec n_2$ and then $n_2 \neq n_1$ 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 $h'_1 : l_1 \rightarrow g_{[n_1]}$, 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 $g'' = g_2[n_1 \leftarrow h'_1(r_1)]$.

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, $h'_1(l_1) = (\text{by definition of } h'_1) g_{[n_1]} = (\text{from } n_1 \prec n_2) g_{[n_1]}[n_2 \leftarrow h_2(r_2)] = (\text{By definition of } h_1) h_1(l_1)[n_2 \leftarrow h_2(r_2)]$. Then, it is easy to see that $h'_1(r_1) = h_1(r_1)[n_2 \leftarrow h_2(r_2)]$. Thus, $g'' = (g[n_2 \leftarrow h_2(r_2)][n_1 \leftarrow (h_1(r_1)[n_2 \leftarrow h_2(r_2)])]$.

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

**Case 2.2 :** $(P \cap \mathcal{N}_r = \emptyset)$. 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 $P \cap \mathcal{N}_r = \emptyset$. Concerning $g_2$, by hypothesis $n_1 \prec n_2$ and $n_2 \neq n_1$ Which implies that $n_1$ is a node of $g_2 = g[n_2 \leftarrow h_2(r_2)]$. Moreover, there exists a homomorphism $h'_1 : l_1 \rightarrow g_{[n_1]}$ (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 $g'' = g_2[n_1 \leftarrow h'_1(r_1)]$. 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 $g'' = (g[n_2 \leftarrow h_2(r_2)][n_1 \leftarrow h_1(r_1)]$. 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 $g_2 \rightarrow_{[n_1, l_1 \rightarrow r_1]} g_1$. Finally, let $g'_1 = g'' = g_1$ 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 $h_1(r_1)$ ($P \cap \mathcal{N}_r = \emptyset$), 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, \( n_1 < n_2 \) and \( n_2 \neq n_1 \) which implies that \( n_1 \) is a node of \( g_2 = g[n_2 \leftarrow h_2(r_2)] \). Moreover, there exists a homomorphism \( h'_1 : l_1 \rightarrow g_2[n_1] \) (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 \( n_1 < n_2 \) in \( g \)) and \( n_2 \) is not a node of \( h_1(r_1) \) (by hypothesis), so by proposition 6 (commutativity), \( g'_1 = g'_2 \).

\[ \square \]

An immediate consequence of Lemma 2 is the following proposition.

**Proposition 11** Let \( SP = (\Sigma, R) \) 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 \( (g_1 \sim g_2) \). \( g_1 \rightarrow g'_1 \) and \( g_2 \rightarrow g'_2 \). Then there exist two admissible graphs \( g''_1 \) and \( g''_2 \) such that \( g'_1 \rightarrow g''_1 \), \( g'_2 \rightarrow g''_2 \) and \( g''_1 \sim g''_2 \). (see Figure 12).

![Figure 12:](image)

**Proof** Since \( g_1 \rightarrow g'_1 \) and \( g_1 \sim g_2 \), by Proposition 10, there exists an admissible graph \( d \) such that \( g_2 \rightarrow d \) and \( g'_1 \sim d \). Since \( g_2 \rightarrow d \) and \( g_2 \rightarrow g'_2 \), by Lemma 2, there exist two admissible graphs \( d' \) and \( g''_2 \) such that \( d \rightarrow d' \), \( g'_2 \rightarrow g''_2 \) and \( d' \sim g''_2 \). Since \( g'_1 \sim d \) and \( d \rightarrow d' \), by Proposition 10, there exists an admissible graph \( g''_1 \) such that \( g'_1 \rightarrow g''_1 \) and \( g''_1 \sim d' \). Finally, since \( g''_1 \sim d' \) and \( d' \sim g''_2 \), we conclude that \( g''_1 \sim g''_2 \).

\[ \square \]

We now generalize Lemma 2.  

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

![Figure 13:](image)
Proof  This proof is done by induction on the length of the derivation \(g \rightarrow g_1\) 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 \(g \rightarrow^n g_1\) and \(g \rightarrow^\omega g_2\). If \(g = g_2\), the proposition is straightforward (i.e., \(g_1' = g_2' = g_1\)), so we suppose \(g \rightarrow g_2\). There exists a graph \(d\) such that \(g \rightarrow d \rightarrow^n g_1\). By Lemma 2, there exist two graphs \(d_1\) and \(d_2\) such that \(d \rightarrow d_1\), \(g_2 \rightarrow d_2\) and \(d_1 \sim_d d_2\) (see \(A\) in Figure 14). Since \(d \rightarrow g_1\) and \(d \rightarrow d_1\), by induction hypothesis, there exist two graphs \(g_1'\) and \(d_1'\) such that \(g_1 \rightarrow g_1', d_1 \rightarrow d_1'\) and \(g_1' \sim_{\beta} d_1'\) (see \(HR\) in Figure 14). Since \(d_1 \rightarrow d_1'\) and \(d_1 \sim_d d_2\), by Proposition 10, there exists a graph \(g_2'\) such that \(d_2 \rightarrow g_2'\) and \(d_1' \sim_d g_2'\) (see \(C\) in Figure 14). Finally, \(g_1' \sim_{\beta} d_1'\) and \(d_1' \sim g_2'\), so \(g_1' \sim_{\beta \gamma} g_2'\), 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 \(SP = (\Sigma, \mathcal{R})\) be a cGRS and \(g, g_1\) and \(g_2\) three admissible graphs. If \(g \rightarrow g_1\) and \(g \rightarrow g_2\), then there exist two admissible graphs \(g_1'\) and \(g_2'\) such that \(g_1 \rightarrow g_1', g_2 \rightarrow g_2'\) and \(g_1' \sim g_2'\). (see Figure 15).

![Figure 15](image)

Proof  Let \(g, g_1\) and \(g_2\) be three admissible graphs such that \(g \rightarrow g_1\) and \(g \rightarrow g_2\). We prove by induction on the length of the derivation \(g \rightarrow g_2\) that there exist two graphs \(g_1'\) and \(g_2'\) such that \(g_1 \rightarrow g_1', g_2 \rightarrow g_2'\) 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 \(g \rightarrow g_1\) and \(g \rightarrow^n g_2\). Then there exists a graph \(d\) such that \(g \rightarrow d \rightarrow^n g_2\). By Proposition 12, there exist two graphs \(d_1\) and \(d_2\) such that \(g_1 \rightarrow d_1\), \(d \rightarrow d_2\) and \(d_1 \sim_d d_2\) (see \(A\) in Figure 16). Since \(d \rightarrow d_2\) and \(d \rightarrow g_2\), by induction hypothesis, there exist two graphs \(d_2'\) and \(g_2'\) such that \(d_2 \rightarrow d_2'\), \(g_2 \rightarrow g_2'\) and \(d_2' \sim_{\beta} g_2'\) (see \(HR\) in Figure 16). Since \(d_2 \rightarrow d_2'\) and \(d_1 \sim_d d_2\), by Proposition 10, there exists a graph \(g_1'\) such that \(d_1 \rightarrow g_1'\) and \(g_1' \sim g_2'\) (see \(C\) in
Finally, \( g'_1 \sim_\gamma d'_2 \) and \( d'_2 \sim_\beta g'_2 \), so \( g'_1 \sim_\beta \gamma g'_2 \), 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 \( SP = (\Sigma, R) \) be a cGRS and \( g_1, g_2, g_3 \) and \( g_4 \) four admissible graphs such that \( g_1 \sim g_2, g_1 \to g'_1 \) and \( g_2 \to g'_2 \). 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 Proposition 13, there exist two admissible graphs \( d' \) and \( g'_3 \) such that \( d \to d' \), \( g'_1 \to g'_3 \) and \( d' \to g'_3 \). Since \( g'_1 \sim d \) and \( d \to d' \), by Proposition 10, there exists an admissible graph \( g'_4 \) such that \( g'_1 \to g'_4 \) and \( g'_4 \sim d' \). Finally, since \( g'_1 \sim d' \) and \( d' \sim g'_4 \), we conclude that \( g'_1 \sim g'_4 \).

\section{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 \( SP = (\Sigma, R) \) be a cGRS. We say that the rewriting relation \( \to \) 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 \( (g_1 \equiv g_2) \), \( g_1 \to g'_1 \) and \( g_2 \to g'_2 \). Then there exist two admissible graphs \( g'_1 \) and \( g'_2 \) such that \( g'_1 \to g'_1 \), \( g'_2 \to g'_2 \) and \( g'_1 \equiv g'_2 \). (see Figure 17).

\[ \square \]

Figure 17: Confluence modulo bisimilarity

**Theorem 2** Let \( SP = (\Sigma, R) \) be a cGRS. Then the rewriting relation \( \to \) 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 \( \mathcal{SP} = (\Sigma, \mathcal{R}) \) be a cGRS, \( g_1 \) and \( g_2 \) two admissible graphs and \( l \rightarrow r \) a rewrite rule of \( \mathcal{R} \).

1. If there exists a homomorphism \( h : g_1 \rightarrow g_2 \) and \( l \) matches \( g_1 \) at the root, then \( l \) matches \( g_2 \) at the root.

2. If there exists a variable-preserving homomorphism \( h : g_1 \rightarrow g_2 \) 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 \( h_1 : l \rightarrow g_1 \) and a homomorphism \( h : g_1 \rightarrow g_2 \), 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 \( \text{Root}_1 \) 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 \( h : g_1 \rightarrow g_2 \) 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 \( h_2 : l \rightarrow g_2 \) is a homomorphism. As \( h : g_1 \rightarrow g_2 \) 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 \( \eta \), for all admissible graphs \( g_1 \) and \( g_2 \) and variable-preserving homomorphism \( h : g_1 \rightarrow g_2 \), 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 \( h : g_1 \rightarrow g_2 \) a variable-preserving homomorphism, such that there exists a homomorphism \( h_2 : l \rightarrow g_2 \). We build a homomorphism \( h_1 : l \rightarrow g_1 \).

\( l \) 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 \( h_2 : l \rightarrow g_2 \) is a homomorphism, the root of \( g_2 \) is also labeled with \( f \). And as \( h : g_1 \rightarrow g_2 \) is a variable-preserving homomorphism, the root of \( g_1 \) is also labeled with \( f \). We now suppose \( S_l(\text{Root}_{t_1}) = n_1 \ldots n_k \), \( S_{g_1}(\text{Root}_{g_1}) = u_1 \ldots u_k \) and \( S_{g_2}(\text{Root}_{g_2}) = v_1 \ldots v_k \). For all \( i \in 1..k \), \( h_{l_{[n_i]}} \) is a variable-preserving homomorphism from \( g_{1_{[n_i]}} \) to \( g_{2_{[v_i]}} \). For all \( i \in 1..k \), \( l_{[n_i]} \) is a linear tree of height less or equal to \( \eta \) and \( h_{g_2_{[u_i]}} \) is a homomorphism from \( g_{1_{[n_i]}} \) to \( g_{2_{[v_i]}} \). So by induction hypothesis, there exists a matcher \( \phi_i : h_{l_{[n_i]}} \rightarrow g_{1_{[n_i]}} \) for all \( i \in 1..k \). Let \( h_1 : N_l \rightarrow N_{g_1} \) be the function such that \( h_1(\text{Root}_{t_1}) = \text{Root}_{g_1} \) and for all \( i \in 1..k \), for all \( n \in N_{l_{[n_i]}} \), \( h_1(n) = \phi_i(n) \). 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 \( N_{l_{[n_i]}} \cap N_{l_{[n_j]}} = \emptyset \) and \( N_{g_1} \cap N_{g_2} = \emptyset \). Thus, for all nodes \( n \) of \( l \), \( h_1(n) \) is defined only when \( n \) is a well defined function. Second, \( h_1 \) preserves the labeling function of \( l \), since \( L_{g_1}(h_1(\text{Root}_{t_1})) = L_{g_1}(\text{Root}_{g_1}) = L_{g_1}(\text{Root}_{t_1}) = f \), and every \( \phi_i \) preserves the labeling function of \( l_{[n_i]} \). Third, \( h_1 \) preserves the successor function of \( l \), since \( S_{g_1}(\text{Root}_{g_1}) = u_1 \ldots u_k = \phi_i(n_1) \ldots \phi_k(n_k) = h_1(S_{l}(\text{Root}_{l})) \), and every \( \phi_i \) preserves the successor function of \( l_{[n_i]} \). So we conclude that \( h_1 : l \rightarrow g_1 \) 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 \( \mu_1 : g_1 \rightarrow g \) and \( \mu_2 : g_2 \rightarrow g \). \( l \) 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.

\( \square \)
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 \( x_1 : \ast (x_2 : 0, x_2) \rightarrow x_2 \). Its left-hand side \( l \) does not match the graph \( g_1 = y_1 : \ast (y_2 : 0, y_3 : 0) \), 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 \( x_2 : 0 \). Consider now the rule \( x_1 : \ast (x_2 : x, x_3 : s(x_3)) \rightarrow x_3 \). Its left-hand side matches the graph \( g_3 = y_1 : \ast (x_2 : 0, x_3 : s(x_3)) \) but it does not match the graph \( g_4 = z_1 : \ast (z_2 : 0, z_3 : s(z_4 : s(z_3))) \), 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 \( SP = (\Sigma, \mathcal{R}) \) 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 \( g_1 \rightarrow^* g'_1 \). (see Figure 18).

![Figure 18](image)

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

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 \( \{l_i \rightarrow r_i | i \in 1..n\} \) be a set of \( n \) variants of \( l \rightarrow r \); let \( \alpha_i \) be the isomorphism from the graph \( l_i \rightarrow r_i \) to the graph \( l \rightarrow r \) (i.e., \( \alpha_i \) is the isomorphism from \( l_i + r_i \) to \( l + r \) which renames all nodes and variables of \( l_i + r_i \) ). \( \psi \) is a homomorphism from \( l \rightarrow g_{2|p_i} \) and \( h_{|p_i} \) is a homomorphism from \( g_{1|p_i} \) to \( g_{2|p_i} \) 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 \( \alpha_i \) and there exists a homomorphism from \( l \rightarrow g_{1|p_i} \), so there exist a homomorphism \( \phi_i \) 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 \( g_1[p_i : \phi_i(r_i)] \). 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 \( g'_1 = g_1[p_1 : \phi_1(r_1)] \ldots [p_n : \phi_n(r_n)] \). We clearly have \( g_1 \rightarrow^* g'_1 \).

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 \( p_1 \) on \( G_1 \) and the pointer redirection \( p_2 \) on \( G_2 \), where \( p_1 = \{ p_1 \mapsto \text{Root}_{\phi_1(r_1)}, \ldots, p_n \mapsto \text{Root}_{\phi_n(r_n)} \} \) and \( p_2 = \{ q \mapsto \text{Root}_{\psi(r)} \} \). We must prove the \( \mu \) is still a homomorphism from \( p_1(G_1) \) to \( p_2(G_2) \). First, \( \mu \) is a function from \( \mathcal{N}_{G_1} \) to \( \mathcal{N}_{G_2} \), so \( \mu \) is also a function from \( \mathcal{N}_{p_1(G_1)} \) to \( \mathcal{N}_{p_2(G_2)} \). Second, \( \mu \) preserves the labeling function of \( G_1 \), so it does on \( p_1(G_1) \). Third, \( \mu \) preserves the successor function of \( p_1(G_1) \) : we must verify that \( \mathcal{S}_{p_2(G_2)}(\mu(n)) = \mu(\mathcal{S}_{p_1(G_1)}(n)) \), for all node \( n \in \mathcal{N}_{p_1(G_1)} \). As \( \mathcal{N}_{p_2(G_2)} = \mathcal{N}_{G_2} \), it is equivalent to prove that for all nodes \( n \in \mathcal{N}_{G_1} \), \( \mathcal{S}_{p_2(G_2)}(\mu(n)) = \mu(p_1(\mathcal{S}_{p_1(G_1)}(n))) \). Since \( \mu \) is a homomorphism from \( G_1 \) to \( G_2 \), \( p_2(\mathcal{S}_{G_2}(\mu(n))) = \mu(p_2(\mathcal{S}_{G_1}(n))) \). So, let us prove that for all nodes \( n \in \mathcal{N}_{G_1} \), \( \mu(p_1(\mathcal{S}_{p_1(G_1)}(n))) = \mu(p_2(\mathcal{S}_{G_1}(n))) \). This result holds if \( \mu(p_2(\mu(n))) = \mu(p_1(\mu(n))) \) for all nodes \( n \) of \( G_1 \). There are two cases for \( n \). First case, if \( n \notin P \). Then \( p_1(n) = n \) and so \( \mu(p_1(n)) = \mu(n) \). Hence, \( \mu(p_1(n)) = \mu(p_2(\mu(n))) \). Second case, if \( n \in P \). Then there exists an \( i \in 1..k \) such that \( n = p_i \). So \( p_1(n) = p_1(p_i) = \text{Root}_{\phi_1(r_i)} = \phi_i(\text{Root}_{r_i}) \). So \( \mu(p_1(n)) = \mu(\phi_i(\text{Root}_{r_i})) \). Moreover, \( p_2(\mu(n)) = \mu(p_2(\mu(n))) = \mu(\phi_i(\text{Root}_{r_i})) = \phi_i(\text{Root}_{r_i}) \). By definition of \( \mu \), \( \mu(\phi_i(\text{Root}_{r_i})) = \psi(\text{Root}_{r_i}) \). Hence, \( \mu(p_1(n)) = \mu(p_2(\mu(n))) \) and we obtain that \( \mu \) preserves the successor function of \( p_1(G_1) \). Finally, \( \mu(\text{Root}_{p_1(G_1)}) = \text{Root}_{p_2(G_2)} \). So we conclude that \( \mu \) is still a homomorphism from \( p_1(G_1) \) to \( p_2(G_2) \).

Third and last stage, let \( g'_1 = p_1(G_1)|_{p_1(\text{Root}_{r_i})} \) and \( g'_2 = p_2(G_2)|_{p_2(\text{Root}_{r_i})} \). Let \( h' = \mu|_{\text{Root}_{r_i}} \). 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 \( SP = (\Sigma, \mathcal{R}) \) 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 \( g_2 \xrightarrow{h'} g'_2 \), then there exist a graph \( g'_1 \) and a variable-preserving homomorphism \( h' : g'_1 \rightarrow g'_2 \) such that \( g_1 \xrightarrow{h'} g'_1 \). (see Figure 19).

![Figure 19:](image)

**Proof** This proof is done by induction on the length of the derivation \( g_2 \xrightarrow{h'} g'_2 \). 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 \( h : g_1 \rightarrow g_2 \) a variable-preserving homomorphism such that \( g_2 \xrightarrow{h'_{n+1}} g'_2 \). Then, there exists a graph \( d' \) such that \( g_2 \xrightarrow{h'_{n+1}} g'_2 \). Since \( g_2 \xrightarrow{h'} g'_2 \) and there exists a variable-preserving homomorphism \( h : g_1 \rightarrow g_2 \), by Lemma 3, there exist a graph \( d \) and a variable-preserving homomorphism \( j : d \rightarrow d' \) such that \( g_1 \xrightarrow{h'} d \). Since there exists a variable-preserving homomorphism \( j : d \rightarrow d' \) and \( d \xrightarrow{h'_{n+1}} g_2 \), by induction hypothesis, there exist a graph \( g'_1 \) and a variable-preserving homomorphism \( h' : g'_1 \rightarrow g'_2 \) such that \( d \xrightarrow{h'_{n+1}} g'_1 \). Finally, \( g_1 \xrightarrow{h'_{n+1}} d \xrightarrow{h'_{n+1}} g'_1 \), so \( g_1 \xrightarrow{h'_{n+1}} g'_1 \).

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 $h : g_1 \rightarrow g_2$ a variable-preserving homomorphism. If there exists a graph $g'_1$ such that $g_1 \rightarrow g'_1$, then there exist two admissible graphs $g''_1$ and $g'_2$ and a variable-preserving homomorphism $h' : g''_1 \rightarrow g'_2$ such that $g'_1 \overset{h}{\rightarrow} g''_1$ and $g_2 \rightarrow g'_2$. (see Figure 20).

![Figure 20:](image)

**Proof** This proof is sketched in Figure 21. We assume $g_1$ rewrites to $g'_1$ at node $p_1$ with rule $l \rightarrow r$. 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 $g_2 \rightarrow_{\{q, l \rightarrow r\}} g'_2$. Since $g_2 \rightarrow_{\{q, l \rightarrow r\}} g'_2$ and there exists a variable-preserving homomorphism $h : g_1 \rightarrow g_2$, by Lemma 3, there exists a graph $d$ and a variable-preserving homomorphism $j : d \rightarrow g'_2$ such that $g_1 \overset{j}{\rightarrow} d$ (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 $g'_1 \overset{j}{\rightarrow} d$ are performed at the nodes of $P$. Therefore, since $p_1$ is a node of $P$ and $g_1 \rightarrow_{\{p_1, l \rightarrow r\}} g'_1$, there exists a graph $g''_1$ such that $g'_1 \overset{\alpha}{\rightarrow} g''_1$ and $g''_1 \overset{j}{\rightarrow} d$ (see $B$ in Figure 21). The nodes used to perform the rewriting derivation $g''_1 \rightarrow g''_1$ are those of $P - \{p_1\}$, that is to say $\{p_2, \ldots, p_n\}$. Finally, the isomorphism of renaming $\alpha$ 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 $g'_2$.

![Figure 21:](image)

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 $h : g_1 \rightarrow g_2$ a variable-preserving homomorphism. If there exists a graph $g'_1$ such that $g_1 \rightarrow g'_1$, then there exist two admissible graphs $g''_1$ and $g'_2$ and a variable-preserving homomorphism $h' : g''_1 \rightarrow g'_2$ such that $g'_1 \overset{h}{\rightarrow} g''_1$ and $g_2 \rightarrow g'_2$. (see Figure 22).
Figure 22:

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

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

We are now ready to prove Theorem 2.

**Proof of theorem 2** Let $SP = (\Sigma, R)$ be a cGRS. Let $g_1$ and $g_2$ be two admissible and bisimilar graphs ($g_1 \equiv g_2$) such that there exist two graphs $g'_1$ and $g'_2$ with $g_1 \xrightarrow{*} g'_1$ and $g_2 \xrightarrow{*} g'_2$. We prove that there exist two admissible graphs $g''_1$ and $g''_2$ such that $g'_1 \xrightarrow{*} g''_1$, $g'_2 \xrightarrow{*} g''_2$ and $g'_1 \equiv g''_1$. 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 $h_1 : g_1 \rightarrow g$ and a variable-preserving homomorphism $h_2 : g_2 \rightarrow g$ (see $A$ in Figure 24). Since $g_1 \xrightarrow{*} g'_1$ and there exists a
variable-preserving homomorphism \( h_1 : g_1 \rightarrow g \), by Proposition 16, there exist two graphs \( f_1 \) and \( d_1 \) and a variable-preserving homomorphism \( h'_1 : f_1 \rightarrow d_1 \) such that \( g'_1 \Rightarrow f_1 \) and \( g \Rightarrow d_1 \) (see B in Figure 24). For the same reason, there exist two graphs \( f_2 \) and \( d_2 \) and a variable-preserving homomorphism \( h'_2 : f_2 \rightarrow d_2 \) such that \( g'_2 \Rightarrow f_2 \) and \( g \Rightarrow d_2 \) (see B' in Figure 24). Since \( g \Rightarrow d_1 \) and \( g \Rightarrow d_2 \), by Theorem 1, there exist two graphs \( d'_1 \) and \( d'_2 \) such that \( d_1 \Rightarrow d'_1 \), \( d_2 \Rightarrow d'_2 \), and \( d'_1 \sim_a d'_2 \) (see C in Figure 24). Since \( d_1 \Rightarrow d'_1 \) and \( d_2 \Rightarrow d'_2 \) and there exists a variable-preserving homomorphism \( h'_1 : f_1 \rightarrow d_1 \), by Proposition 15, there exist a graph \( g''_1 \) and a variable-preserving homomorphism \( h''_1 : g''_1 \rightarrow d'_1 \) such that \( f_1 \Rightarrow g''_1 \) (see D in Figure 24). For the same reason, there exist a graph \( g''_2 \) and a variable-preserving homomorphism \( h''_2 : g''_2 \rightarrow d'_2 \) such that \( f_2 \Rightarrow g''_2 \) (see D' in Figure 24). Finally, \( a \circ h''_1 \) is a variable-preserving homomorphism from \( g''_1 \) to \( d'_1 \) and \( h''_2 \) is a variable-preserving homomorphism from \( g''_2 \) to \( d'_2 \), so by definition of bisimilarity, \( g''_1 \cong g''_2 \).

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 \( SP = (\Sigma, \mathcal{R}) \) be a cGRS. A (sequential) graph rewriting strategy is a partial function \( S \) 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 \( g \Rightarrow_S g' \) and speak of an \( S \)-step from \( g \) to \( g' \) whenever \( S(g) = (p, R) \) and \( g \rightarrow_{[p, R]} g' \). \( 
\Rightarrow_S \) denotes the reflexive and transitive closure of \( \Rightarrow_S \) and we speak of an \( S \)-derivation from \( g \) to \( g' \) whenever \( g \Rightarrow_S g' \).

A strategy \( S \) is e-normalizing iff for all admissible graph \( g \) admitting a constructor normal form \( c \), if \( g \Rightarrow c \), then there exists a constructor graph \( c' \) such that \( g \Rightarrow_S c' \) and \( c' \sim c \).

A strategy \( S \) is e-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 \( S \)-steps ends with a constructor normal form \( c' \) such that \( c \sim c' \).

**Definition 27** Let \( SP = (\Sigma, \mathcal{R}) \) be a cGRS, \( g \) and \( g' \) two admissible graphs and \( B = g \Rightarrow g' \) 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 \( q \in \text{Roots}_g \) or there exist a root \( r \in \text{Roots}_g \) 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 \( q \in \text{Roots}_g \) or there exist a root
Example 20 Let \( g = \text{cons}(x1:*((x3:0,x4:+(x3,x3)),x5:\text{cons}(x4,x6:nil))) \). By definition, \( g|_{x2} \) and \( g|_{x4} \) are outermost needed redexes. However, there exist a path from \( x2 \) to \( x4 \). Notice that \( g|_{x4} \) does not vanish when \( g|_{x2} \) 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 \( SP = (\Sigma, R) \) be a cGRS. A tree \( T \) is a partial definitional tree, or pdt, with pattern \( \pi \) iff one of the following cases holds:

- \( T = \text{rule}(\pi \rightarrow r) \), where \( \pi \rightarrow r \) is a variant of a rule of \( R \).
- \( T = \text{branch}(\pi, o, T_1, \ldots, T_k) \), where \( o \) is a variable node of \( \pi \), \( o \) is of sort \( s \), \( c_1, \ldots, c_k \) \((k > 0)\) are different constructors of the sort \( s \) and for all \( j \in 1..k \), \( T_j \) 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 \( \text{pattern}(T) \) to denote the pattern argument of a pdt.

A definitional tree \( T \) 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 \( l \rightarrow r \) of \( R \), with \( l \) of the form \( f(g_1, \ldots, g_n) \), there exists a leaf \( \text{rule}(l' \rightarrow r') \) of \( T \) such that \( l' \rightarrow r' \) is a variant of \( l \rightarrow r \). 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:

\[
\begin{align*}
x1: +(x2:0,x3:x) &\rightarrow x3 \\
x4:+(x5:\text{succ}(x6:x),x7:y) &\rightarrow x8:\text{succ}(x9:+(x6,x7))
\end{align*}
\]

The definitional tree \( T_+ \) of the operation \( + \) is

\[
T_+ = \text{branch}(11:+(12:X,13:Y), \ 12, \ 13,
\text{rule}(14:+(15:0,16:U) \rightarrow 16),
\text{rule}(17:+(18:\text{succ}(19:V),110:W) \rightarrow 111:\text{succ}(112:+(19,110))))
\]

Now, we are ready to define our graph rewriting strategy noted \( \Phi \). The strategy \( \Phi \) 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 \( R \). \( \Phi \) uses an auxiliary function \( \varphi \) which takes two arguments: an operation-rooted term graph and a pdt of this operation.
Definition 29 (Admissible graph rewriting strategy)
Let \( SP = (\Sigma, \mathcal{R}) \) be an inductively sequential cGRS and \( g \) an admissible graph. \( \Phi \) is the partial function defined by \( \Phi(g) = \varphi(g_{|_{\varphi}}, T_f) \), where \( p \) is an outermost node of \( g \) labeled with a defined operation \( f \) and \( T_f \) is a definitional tree of \( f \).

Let \( g \) be an admissible operation-rooted term graph and \( T \) a pdt such that \( \text{pattern}(T) \leq g \). We define the partial function \( \varphi \):

\[
\varphi(g, T) = \begin{cases} 
(Root_g, \pi' \to r') & \text{if } T = \text{rule}(\pi \to r) \text{ and } \\
\varphi(g, T_i) & \text{if } T = \text{branch}(\pi, o, T_1, \ldots, T_k) \text{ and } \\
(p, R) & \text{if } T = \text{branch}(\pi, o, T_1, \ldots, T_k), \\
& \pi \text{ matches } g \text{ at the root by homomorphism } h : \pi \to g, \\
& h(o) \text{ is labeled with a defined operation } f \text{ (in } g), \\
& T' \text{ is a definitional tree of } f \text{ and } \\
& \varphi(g_{|_{h(o)}}, T') = (p, R).
\end{cases}
\]

Example 22 Consider the admissible graph \( g = x_1 : \text{cons}(x_2 : *(x_3 : *(x_4 : 0, x_4), x_3), x_1) \). Then

\[
\Phi(g) = \varphi( x_2 : *(x_3 : *(x_4 : 0, x_4), x_3), T_+) = \varphi( x_2 : *(x_3 : *(x_4 : 0, x_4), x_3), \\
\text{branch}(11 : +(12 : X, 13 : Y), 12, \ldots) = \varphi( x_3 : +(x_4 : 0, x_4), \\
\text{branch}(11 : +(12 : X, 13 : Y), 12, \text{rule}(14 : +(15 : 0, 16 : U) \to 16), \ldots)) = (x_3, v_1 : +(v_2 : 0, v_3 : Z) \to v_3)
\]

Lemma 5 Let \( SP = (\Sigma, \mathcal{R}) \) be an inductively sequential cGRS, \( f \) a defined operation, \( T_f \) a definitional tree of \( f \), and \( g \) an admissible term graph whose root is labeled with \( f \). If \( \varphi(g, T_f) = (p, R) \), 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 \( \varphi(g, T) \) is not defined, then \( g \) cannot be rewritten into a constructor-rooted term graph.

Proof In the sequel, \( \sqsubset \) denotes the Noetherian ordering defined by : \( (g_1, T_1) \sqsubset (g_2, T_2) \) 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 \( T_1 \) is a proper subtree of \( T_2 \). The proof is by Noetherian induction on \( \sqsubset \). We consider the different cases of the definition of \( \varphi \).

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

**Induction step**: consider \((g, T)\) where \( g \) is an admissible operation-rooted term graph, \( T = \text{branch}(\pi, o, T_1, \ldots, T_k) \), where \( o \) is a variable node of \( \pi \), \( o \) is of sort \( s \), \( c_1, \ldots, c_k \) \((k > 0)\) are different constructors of the sort \( s \) and for all \( j \in 1..k \), \( T_j \) 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 $\text{pattern}(T) \leq g$, i.e., there exists a homomorphism $h : \pi \rightarrow g$. There are three cases to consider.

**Case 1**: $g_{h(o)}$ is constructor-rooted, say $L_2(h(o)) = c$. Then, there are two possibilities. If there exists an $i \in 1..k$ such that $\text{pattern}(T_i) \leq g$, then by definition of $\varphi$, $\varphi(g, T) = \varphi(g, T_i)$. By induction hypothesis, all the claims hold already for $\varphi(g, T_i)$, so they also hold for $\varphi(g, T)$. Else, there exists no $i \in 1..k$ such that $\text{pattern}(T_i) \leq g$, then the operation $L_2(\text{Root}_2)$ is incompletely defined by $R$. So $g$ cannot be rewritten into a constructor-rooted term graph and $\varphi(g, T)$ is not defined.

**Case 2**: $g_{h(o)}$ is operation-rooted, say $L_2(h(o)) = f$. Let $T'$ be a definitional tree of $f$. Then, either $\varphi(g_{h(o)}, T')$ is defined, or not.

First sub-case, suppose $\varphi(g_{h(o)}, T') = (p, R)$, where $p$ is a node of $g_{h(o)}$ and $R$ is a variant of a rule of $R$. By the definition of $\varphi$, $\text{pattern}(T) \leq g$. By the definition of a definitional tree, for every rule $l \rightarrow r$ of $R$ whose left-hand side may match $g$ at the root, there exists a leaf rule $l' \rightarrow r'$ of $T$, where $l' \rightarrow r'$ is a variant of $l \rightarrow r$. By construction of a definitional tree, if $l \rightarrow r$ is a rule represented by a leaf of $T$, then $\text{pattern}(T) \leq l$, i.e., there exists a homomorphism $h' : \pi \rightarrow l$. By hypothesis, $T$ is a branch pdt and not a rule pdt so $\text{pattern}(T) < l$. 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 $R$ whose left-hand side matches $g$ at the root, i.e., $g$ is not a redux. 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 redux of $g_{h(o)}$ matched by the left-hand side of $R$, thus $g_{p}$ is a redux of $g$ matched by the left-hand side of $R$. (iii) By the definition of $\varphi$, $\pi \leq g$, i.e., there exists a homomorphism $h : \pi \rightarrow g$. By the definition of a definitional tree, $\pi$ is a pattern and $o$ is a node of $\pi$. These conditions imply that there exists a path from $\text{Root}_2$ to $h(o)$ along which all nodes are labeled with constructor symbols, except $\text{Root}_2$ and $h(o)$. In cGRS, redexes occur only at nodes labeled with defined operations. We have just proved that $g$ is not a redux. Thus, there is no redux in $g$ above $h(o)$. By induction hypothesis, $g_{p}$ is an outermost redux in $g_{h(o)}$. So $g_{p}$ is also an outermost redux in $g$.

Second sub-case, suppose $\varphi(g_{h(o)}, T')$ is not defined. Then, $\varphi(g, T)$ 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 redux 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 $\varphi(g, T)$ is not defined. On the other hand, by the definition of $\varphi$, $\pi \leq g$. Any rule whose left-hand side may match $g$ at the root is represented by a leaf of $T$. If $l \rightarrow r$ is a rule represented by a leaf of $T$, then by construction of a definitional tree, $\text{pattern}(T) \leq l$, i.e., there exists a homomorphism $h' : \pi \rightarrow l$. 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 redux. 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.

\[ \square \]
Proposition 17  Let \( SP = (\Sigma, \mathcal{R}) \) 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 \( \Phi \).

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 \( D = g \rightarrow c \), 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 \( SP = (\Sigma, \mathcal{R}) \) be a cGRS, \( g \) an admissible non constructor graph and \( c \) a constructor graph, such that there exists a rewriting derivation \( g \rightarrow c \) of length \( n \). Then, there exists an admissible graph \( g' \) such that \( g \rightarrow g' \) and a rewriting derivation \( g' \rightarrow c' \) 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

\[
g \rightarrow [u_1, R_1] a_1 \rightarrow [u_2, R_2] \cdots \rightarrow [u_{k-1}, R_{k-1}] a_{k-1} \rightarrow [p, R] a_k \rightarrow [u_{k+1}, R_{k+1}] \cdots \rightarrow [u_n, R_n] \]

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 \( g \rightarrow \gamma_{[p, R]} g' \) and the derivation \( B \) is of the form \( B = g \rightarrow [u_1, R_1] a_1 \rightarrow [u_2, R_2] \cdots \rightarrow [u_{k-1}, R_{k-1}] a_{k-1} \rightarrow [p, R] a_k \rightarrow [u_{k+1}, R_{k+1}] \cdots \rightarrow [u_n, R_n] c \), with \( u_i \neq p \) for all \( i \in (1..(k - 1)) \) and \( i \in (k + 1) \).

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 \( a_1 \rightarrow \gamma_{[p, R]} b_1, g' \rightarrow \gamma_{[u_1, R_1]} a'_1 \), 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 \( a_1 \sim [p, R] b_1 \). More generally, according to Proposition 12, if \( g \rightarrow [u_1, R_1] a_1 \cdots \rightarrow [u_{k-1}, R_{k-1}] a_{k-1} \) and \( g \rightarrow [p, R] g' \), then there exist two graphs \( b_{k-1}, a'_{k-1} \) such that \( a_{k-1} \rightarrow \gamma_{[p, R]} b_{k-1}, g' \rightarrow \gamma_{[u_1, R_1]} a'_1 \cdots \rightarrow \gamma_{[u_{k-1}, R_{k-1}]} a'_{k-1} \), 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 \( a_{k-1} \sim [p, R] b_{k-1} \). 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 \( g' \rightarrow \gamma_{[u_1, R_1]} a'_{1} \cdots \rightarrow \gamma_{[u_{k-1}, R_{k-1}]} a'_{k-1} \) is less or equal to \( k - 1 \). Second, \( a_{k-1} \rightarrow [p, R] a_k \) and \( a_{k-1} \rightarrow \gamma_{[p, R]} b_{k-1} \), hence \( a_{k-1} \sim b_{k-1} \) and since \( a'_{k-1} \sim b_{k-1}, a'_{k-1} \sim a_{k-1} \) \( a_{k-1} \sim a_{k-1} \) \( a_{k-1} \sim a_{k-1} \), it is clear that there exists a constructor graph \( c' \) such \( c \sim c' \) and \( a'_{k-1} \rightarrow \gamma_{[u_{k+1}, R_{k+1}]} \cdots \rightarrow [u_n, R_n] c' \). The length of the derivation \( B \) is exactly \( n \), whereas the length of the derivation \( g' \rightarrow \gamma_{[u_1, R_1]} a'_{1} \cdots \rightarrow \gamma_{[u_{k-1}, R_{k-1}]} a'_{k-1} \rightarrow [p, R] a_k \rightarrow [u_{k+1}, R_{k+1}] \cdots \rightarrow [u_n, R_n] c' \) is less or equal to \( n - 1 \).

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

30
**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 \( g_0 \rightarrow c_0 \) 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 \( \Phi \)-steps. \( D \) is of the form \( g_0 \rightarrow g'_0 \rightarrow \Phi g_1 \rightarrow g'_1 \rightarrow \Phi g_2 \rightarrow \ldots \). By confluence (see Theorem 1), since \( g_0 \rightarrow g'_0 \), \( g_0 \rightarrow c_0 \) 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 \( g'_0 \rightarrow c'_0 \) of length \( n'_0 \leq n_0 \). Since \( g'_0 \rightarrow c'_0 \) 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 \( g_1 \rightarrow c_1 \) of length \( n_1 < n'_0 \). Let \( D' \) be the sub-derivation of \( D \) starting with \( g_1 \). The derivation \( g_1 \rightarrow c_1 \) is of length \( n_1 < n_0 \) and \( D' \) contains infinitely many times \( \Phi \)-steps (since \( D \) contains infinitely many times \( \Phi \)-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 \( \Phi \).

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

**Proof** By induction on the length \( n \) of the derivation \( g \rightarrow c \). 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 \( g' \rightarrow c' \) of length \( n' < n \). By induction hypothesis, there exist a constructor graph \( c'' \sim c' \) and a \( \Phi \)-derivation \( g' \rightarrow c'' \) of length \( n'' \leq n' \). The length of the derivation \( g \rightarrow \Phi g' \rightarrow c'' \) is \( n'' + 1 \leq n' + 1 \leq n \) and \( c'' \sim c \).

\[ \square \]

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 needness 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 $h : l \rightarrow g|_n$. $h$ is called the matcher of $l$ on $g$ at node $n$.

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 $h : \mathcal{N}_{g_1} \rightarrow \mathcal{N}_{g_2}$ such that $h$ is a homomorphism from $g_1$ to $g_2$, when it exists, and fails otherwise. It begins with the pair ($\{\text{Root}_{g_1} \triangleright \text{Root}_{g_2}\}, \emptyset$), and ends with the pair ($\emptyset, h$) in case of success, or $\emptyset$ in case of failure. The rules are:

$\triangleright_{\mathcal{N}_{g_1} \times \mathcal{N}_{g_2}}$

References


