Inductively Sequential Term-Graph Rewrite Systems*

Rachid Echahed

CNRS, LIG
46, avenue Félix Viallet, F-38031 Grenoble, France
Rachid.Echahed@imag.fr

Abstract. Definitional trees have been introduced by Sergio Antoy in order to design an efficient term rewrite strategy which computes needed outermost redexes. In this paper, we consider the use of definitional trees in the context of term-graph rewriting. We show that, unlike the case of term rewrite systems, the strategies induced by definitional trees do not always compute needed redexes, in presence of term-graph rewrite systems. We then define a new class called inductively sequential term-graph rewrite systems (istGRS) for which needed redexes are still provided by definitional trees. Systems in this class are not confluent in general. We give additional syntactic criteria over istGRS's which ensure the confluence property with respect to the set of admissible term-graphs.

1 Introduction

Many declarative languages are based on term rewrite systems (TRS). There are good reasons for that, they actually benefit from a solid logical foundations (equational logic, model-theory, proof methods) as well as very efficient implementation techniques. Term rewrite systems have been used also as a unifying computational model for declarative languages sharing both functional and logic features with very efficient operational semantics [5].

However, real-life programs, very often, deal with complex data-structures built by means of pointers (e.g., circular lists, doubly-linked lists, etc.). Such data-structures can be modeled as term-graphs [6,19] and are sometimes mandatory for efficiency reasons, namely time and space complexity of algorithms. Term rewriting constitutes a computational model which is Turing-complete and thus can encode theoretically any transformation over term-graphs, but such encodings are in general cumbersome and too costly. Thus term-graphs appear as a good trade-off to use rewrite systems to compute with general data-structures without using all the machinery specific to graph transformations [20,13,14]. In recent works, e.g. [4,3,18] term-graph rewriting has also been considered as a means to implement naturally, in declarative languages, the call-time choice semantics introduced in [16].

* This work has been partly funded by the project ARROWS of the French *Agence Nationale de la Recherche.*

H. Ehrig et al. (Eds.): ICGT 2008, LNCS 5214, pp. 84–98, 2008.

A new class of term-graph rewrite systems (hereafter, noted tGRS) has been introduced recently in [8]. This class is a conservative extension of those introduced in [9,10]. It provides some features dedicated especially to pointer rewriting such as *redirection of pointers* or *node constraints* (see section 2). These features allow one to write, in a rule-based language, algorithms with efficient space complexity such as *in-situ list reversal* or those manipulating node constraints such as the *length of circular lists*. Such algorithms were not possible to encode directly in previous works regarding term-graph rewriting such as [19,9,10].

The new features of tGRSs are very appealing. That is why, we intend to pursue our efforts in investigating the class of tGRSs. In [8], a categorical approach has been proposed, [12] present a discussion about the use of term-graphs with priorities as a means to overcome the non-confluence issues and [11] presents the first general and complete narrowing procedure which is able to synthesize solutions with circular data-structures.

The present paper is a first step towards the conception of efficient rewrite strategies in presence of subclasses of tGRSs. We particularly consider the use of *Definitional Trees* introduced by Antoy in his seminal paper [1]. Definitional trees have been successfully used in defining efficient strategies either in term rewriting and narrowing [1,2,5], graph rewriting and graph narrowing [9,10,4]. We show that, the strategies induced by Definitional trees do not compute needed redexes in general. Then, we define a particular class of term-graph rewrite systems for which the induced strategies are efficient and compute needed redexes.

The paper is organized as follows. The next section defines the class of term-graph rewrite systems that we consider. In section 3 we show some negative results regarding the use of definitional trees and introduce the class of inductively sequential tGRSs, for which Definitional trees help to compute needed redexes. In section 4 we show the confluence property for a subclass of inductively sequential term-graph rewrite systems. Section 5 concludes the paper.

2 Preliminary Definitions

In this section we define a class of term-graph rewrite systems, denoted tGRS. We define the shape of its rules and the process of rewriting. The right-hand sides of the rules consist of sequences of actions. These actions are intended to decompose the transformation of graphs into consecutive *atomic* actions.

Definition 1 (Signature). *A* many-sorted signature $\Sigma = \langle S, \Omega \rangle$ *consists of a set* S *of sorts and an* S*-indexed family of sets of operation symbols* $\Omega = \uplus_{s \in S} \Omega_s$ *with* $\Omega_s = \uplus_{w \in S^*} \Omega_{w \to s}$*. We shall write* $f : s_1 \dots s_n \to s$ *whenever* $f \in \Omega_{s_1 \dots s_n \to s}$ *and say that* f *is of* sort s *and* rank $s_1 \dots s_n$*. A* constructor-based signature Σ *is a triple* $\Sigma = \langle S, \mathcal{C}, \mathcal{D} \rangle$ *such that* S *is a set of sorts,* $\mathcal{C}$ *is an* S*-indexed family of sets of* constructor symbols*,* $\mathcal{D}$ *is an* S*-indexed family of sets of* defined operations*,* $\mathcal{C} \cap \mathcal{D} = \emptyset$ *and* $\langle S, \mathcal{C} \uplus \mathcal{D} \rangle$ *is a signature.*

A term-graph is defined in this paper as a set of nodes and edges between the nodes [6]. Each node may be labeled with an operation symbol or not. A node

which is not labeled will act as a variable. Let $\mathcal{N} = \uplus_{s \in S} \mathcal{N}_s$, be an S-indexed family of countable sets of *nodes*. $\mathcal{N}$ is supposed to be fixed throughout the rest of the paper.

Definition 2 (Term-Graph)
A term-graph g *over* $\langle \Sigma, \mathcal{N} \rangle$ *is a tuple* $g = \langle \mathcal{N}_g, \mathcal{N}_g^\Omega, \mathcal{L}_g, \mathcal{S}_g \rangle$ *such that :*

1. $\mathcal{N}_g$ *is the set of nodes of* g*, i.e.,* $\mathcal{N}_g = \uplus_{s \in S} (\mathcal{N}_g)_s$ *with* $(\mathcal{N}_g)_s \subseteq \mathcal{N}_s$.
2. $\mathcal{N}_g^\Omega$ *is the subset of labeled nodes of* g, $\mathcal{N}_g^\Omega \subseteq \mathcal{N}_g$
3. $\mathcal{L}_g$*, the* labeling function of g*, is an* S*-indexed family of functions associating an operation symbol to each labeled node of* g*, i.e.,* $\mathcal{L}_g = \uplus_{s \in S} (\mathcal{L}_g)_s$ *with* $(\mathcal{L}_g)_s : (\mathcal{N}_g^\Omega)_s \to \Omega_s$.
4. $\mathcal{S}_g$*, the* successor function of g*, is an* S*-indexed family of functions associating a (possibly empty) string of nodes to each labeled node of* g*, i.e.,* $\mathcal{S}_g = \uplus_{s \in S} (\mathcal{S}_g)_s$ *with* $(\mathcal{S}_g)_s : (\mathcal{N}_g^\Omega)_s \to \mathcal{N}_g^*$ *such that for every node* $n \in (\mathcal{N}_g)_s$ *:*
   - *if* $(\mathcal{L}_g)_s(n) = f$ *with* $f : s_1 \dots s_k \to s$*, then there exist* $n_1, \dots, n_k \in \mathcal{N}_g$ *such that* $(\mathcal{S}_g)_s(n) = n_1 \dots n_k$ *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).*

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

We write $\mathrm{ar}(n)$ *for the arity of node* n *which is equal to the length of* $\mathcal{S}_g(n)$*. A rooted term-graph, denoted by* g^n*, is a term-graph* g *with a distinguished node* n *(*$n \in \mathcal{N}_g$*) called the root of* g*.* n *will be denoted by* $\mathcal{R}oot_g$*. Let* g *be a term-graph and* n *and* m *two nodes of* g *(*$n, m \in \mathcal{N}_g$*), we write* $n \rightsquigarrow_g m$ *iff* $m \in \mathcal{S}_g(n)$*. We will say that node* m *is reachable in* g *from node* n *iff* $n \overset{*}{\rightsquigarrow}_g m$*. A rooted term-graph* g^n *is a* constructor-rooted term-graph *if and only if the root* n *is labeled by a constructor (i.e.* $\mathcal{L}_g(n) \in \mathcal{C}$*). A rooted term-graph* g^n *is a* constructor term-graph *if and only if every reachable node* m *from the root* n *(*$n \overset{*}{\rightsquigarrow}_g m$*),* m *is either labeled by a constructor symbol (*$\mathcal{L}_g(m) \in \mathcal{C}$*) or* m *is not labeled (*$m \notin \mathcal{N}_g^\Omega$*).*

In the sequel, we will assume that all formulae we are considering are well-sorted, and thus drop subscripts related to the many-sorted framework.

As the formal definition of term-graphs is not very convenient to write examples, we recall below the linear notation [6] of term-graphs. In the following grammar, the variable A (resp. n) ranges over the set Ω (resp. $\mathcal{N}$):

TermGraph ::= Node | Node + TermGraph
Node ::= n:A(Node,...,Node) | n:• | n

The root of a rooted term-graph defined by means of a linear expression is the first node of the expression. n:• means that node n is not labeled. + stands for the union of graph definitions.

Example 1. Let G_1^a be the graph (see Fig.1) defined by $G_1^a = \langle \mathcal{N}_{G_1^a}, \mathcal{N}_{G_1^a}^\Omega, \mathcal{L}_{G_1^a}, \mathcal{S}_{G_1^a} \rangle$ such that:

- $\mathcal{N}_{G_1^a} = \{a, b, c, d, e\}$
- $\mathcal{N}_{G_1^a}^\Omega = \{a, b, c, e\}$

– $\mathcal{L}_{G_1^a}(a) = succ; \mathcal{L}_{G_1^a}(b) = f; \mathcal{L}_{G_1^a}(c) = g; \mathcal{L}_{G_1^a}(e) = h$
– $\mathcal{S}_{G_1^a}(a) = b; \mathcal{S}_{G_1^a}(b) = ce; \mathcal{S}_{G_1^a}(c) = de; \mathcal{S}_{G_1^a}(e) = b$

G_1^a could also be written using the linear notation as follows:

$$G_1^a = a : succ(b : f(c : g(d : \bullet, e : h(b)), e))$$

Definition 3 (Homomorphism). *Let g_1^n and g_2^m be two rooted term-graphs. A* homomorphism *h from g_1^n to g_2^m is a mapping $h : \mathcal{N}_{g_1^n} \rightarrow \mathcal{N}_{g_2^m}$ which preserves the root, the labeled nodes and the labeling and successor functions, i.e., $h(n) = m$, $h(\mathcal{N}^{\Omega}_{g_1^n}) \subseteq \mathcal{N}^{\Omega}_{g_2^m}$, and for each labeled node, p, in g_1^n, $\mathcal{L}_{g_2^m}(h(p)) = \mathcal{L}_{g_1^n}(p)$ and $\mathcal{S}_{g_2^m}(h(p)) = h^*(\mathcal{S}_{g_1^n}(p))$ where h^* denotes the extension of h to strings (of nodes) defined by $h^*(p_1 \ldots p_k) = h(p_1) \ldots h(p_k)$.*

Notice that homomorphisms, as defined above, can map unlabeled nodes to labeled ones.

Definition 4 (Actions). *An action has one of the following forms. We omit to give sort constraints which are quite straightforward and thus we assume that all constructions are well-sorted.*

– *a **node definition** or **node labeling** $\alpha : f(\alpha_1, \ldots, \alpha_n)$ where $\alpha, \alpha_1, \ldots, \alpha_n$ are nodes and f is a label of rank $s_1, \ldots, s_n$. This means that α is labeled by f and $\alpha_1 \ldots \alpha_n$ are the successor nodes of α ($\mathcal{S}(\alpha) = \alpha_1 \ldots \alpha_n$).*
– *an **edge redirection** or **local redirection** $\alpha \gg_i \beta$ where α, β are nodes and $i \in \{1, \ldots, \mathrm{ar}(\mathcal{L}(\alpha))\}$. This is an edge redirection and means that the target of the i^{th} edge outgoing α is redirected to point to the node β.*
– *a **global redirection** $\alpha \gg \beta$ where α and β are nodes. This means that all edges pointing to α are redirected to point to the node β.*

The result of applying an action a to a term-graph g is denoted by $a[g]$ and is defined as the following term-graph g':

– *If $a = \alpha : f(\alpha_1, \ldots, \alpha_n)$ then $\mathcal{N}_{g'} = \mathcal{N}_g \cup \{\alpha, \alpha_1, \ldots, \alpha_n\}$, $\mathcal{L}_{g'}(\alpha) = f$, $\mathcal{L}_{g'}(\beta) = \mathcal{L}_g(\beta)$ if $\beta \neq \alpha$, and $\mathcal{S}_{g'}(\alpha) = \alpha_1, \ldots, \alpha_n$, $\mathcal{S}_{g'}(\beta) = \mathcal{S}_g(\beta)$ if $\beta \neq \alpha$. $\cup$ denotes classical union.*
– *If $a = \alpha \gg_i \beta$ then $\mathcal{N}_{g'} = \mathcal{N}_g$, $\mathcal{L}_{g'} = \mathcal{L}_g$, and if $\mathcal{S}_g(\alpha) = \alpha_1, \ldots, \alpha_i, \ldots, \alpha_n$ then $\mathcal{S}_{g'}(\alpha) = \alpha_1, \ldots, \alpha_{i-1}, \beta, \alpha_{i+1}, \ldots, \alpha_n$ and for any node γ we have $\mathcal{S}_{g'}(\gamma) = \mathcal{S}_g(\gamma)$ iff $\gamma \neq \alpha$. If α does not occur in $\mathcal{N}_g$, then $g' = g$.*
– *If $a = \alpha \gg \beta$ then $\mathcal{N}_{g'} = \mathcal{N}_g$, $\mathcal{L}_{g'} = \mathcal{L}_g$ and for all nodes δ such that $\mathcal{S}_g(\delta) = \alpha_1, \ldots, \alpha_n$ then $\mathcal{S}_{g'}(\delta) = \alpha'_1, \ldots, \alpha'_n$ such that for i in $1..n$, $\alpha'_i = \beta$ if $\alpha_i = \alpha$, and $\alpha'_i = \alpha_i$ if $\alpha_i \neq \alpha$. If α does not occur in $\mathcal{N}_g$, then $g' = g$.*

The application of an action a to a rooted term-graph g^n is a rooted term-graph g'^m such that $g' = a[g]$ and root m is defined as follows:

– *$m = n$ if a is not of the form $n \gg p$.*
– *$m = p$ if a is of the form $n \gg p$.*

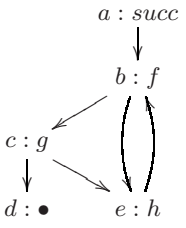

Fig. 1. Term-graph G_1^a

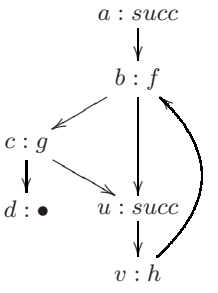

Fig. 2. Term-graph G_2^a

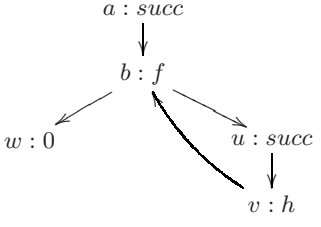

Fig. 3. Term-graph G_3^a

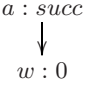

Fig. 4. Term-graph G_4^a

The application of a sequence of actions u to a (rooted) term-graph g is defined inductively as follows : $u[g] = g$ if u is the empty sequence and $u[g] = u'[a[g]]$ if $u = a; u'$ where ; is the concatenation operation.

Example 2
Let G_1^a be the graph defined in *Example1* (see Fig.1).
Let G_2^a be the graph (see Fig.2) $G_2^a = a : succ(b : f(c : g(d : \bullet, u : succ(v : h(b))), u))$
Let G_3^a be the graph (see Fig.3) $G_3^a = a : succ(b : f(w : 0, u : succ(v : h(b))))$
Let G_4^a be the graph (see Fig.4) $G_4^a = a : succ(w : 0)$

Below we give some examples of the application of actions on the graphs above. The first line shows the application of the actions $v : h(b)$; $u : succ(v)$; $e \gg u$ on the term-graph G_1^a. The second line shows the application of the actions $w : 0$; $c \gg w$ on the term-graph G_2^a. The last line shows the application of the action $b \gg w$ on the term-graph G_3^a.

$v : h(b)$; $u : succ(\ v)$; $e \gg u$ $\ [G_1^a] = u : succ(v)$; $e \gg u[G_1^a + v : h(b)] = e \gg u$ $\ [G_1^a + u : succ(v : h(b))] = G_2^a + e : h(b)$

$w : 0$; $c \gg w$ $\ [G_2^a] = c \gg w$ $\ [G_2^a + w : 0] = G_3^a + c : g(d : \bullet, u)$

$b \gg w$ $\ [G_3^a] = G_4^a + b : f(w, u : succ(v : h(w)))$

Definition 5 (Node Constraint). *A* node constraint *is a (possibly empty) conjunction of disequations between nodes:* $\bigwedge_{i=1}^{n}(\alpha_i \not\equiv \beta_i)$. *A substitution* $\sigma : \mathcal{N} \to \mathcal{N}$ *is a* solution *of a constraint* $\phi = \bigwedge_{i=1}^{n}(\alpha_i \not\equiv \beta_i)$ *iff for any* $i \in [1..n]$, *we have* $\sigma(\alpha_i) \neq \sigma(\beta_i)$. *We denote by* $sol(\phi)$ *the set of solutions of* ϕ.

Notice that we do not use equality constraints. Such equalities may be encoded directly into term-graphs.

Definition 6 (Rule, system)
A term-graph rewrite rule *is an expression of the form* $[l \mid c] \to r$ *where* r *is a sequence of actions,* c *is a constraint and* l *is a rooted term-graph s.t. for any node* α *occurring in* l, *we have* $\mathcal{R}oot_l \overset{*}{\rightsquigarrow}_l \alpha$ *(i.e. any node occurring in the left-hand side must be reachable from the root* $\mathcal{R}oot_l$*). A rule* ρ_2 *is said to be a variant of a rule* ρ_1 *iff* ρ_2 *is obtained from* ρ_1 *by (one-one) renaming all the nodes in* ρ_1. *A* term-graph rewrite system *is a set of rewrite rules.*

Example 3. We first define an operation, *sameloc*, which tests whether two arguments are located at the same place or not. Such operation is sometimes used to enhance the implementation of equality in declarative languages.

$r : sameloc(n : \bullet, n) \to q : true; r \gg q$
$[r : sameloc(n : \bullet, m : \bullet) \mid n \not\equiv m] \to q : false; r \gg q$

As a second example, we define below the operation *length* which deals with cyclic data-structures. $length(p : \bullet)$ computes the number of elements of any, possibly circular, list matched by node p.

$r : length(p : \bullet) \to r' : length'(p, p); r \gg r'$
$r : length'(p_1 : nil, p_2 : \bullet) \to r' : 0; r \gg r'$
$r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_2) \to r' : s(0); r \gg r'$
$[r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \not\equiv p_3] \to r' : s(q : \bullet); q : length'(p_2, p_3); r \gg r'$

Pointers help very often to enhance the efficiency of algorithms. In the following, we define the operation *reverse* which performs the so-called "in-situ list reversal".

$2\ o : reverse(p : \bullet) \to o' : reverse'(p, q : nil); o \gg o'$
$o : reverse'(p_1 : cons(n : \bullet, q : nil), p_2 : \bullet) \to p_1 \gg_2 p_2; o \gg p_1$
$o : reverse'(p_1 : cons(n : \bullet, p_2 : cons(m : \bullet, p_3 : \bullet), p_4 : \bullet) \to p_1 \gg_2 p_4; o \gg_1 p_2; o \gg_2 p_1$

The last example illustrates the encoding of classical term rewrite systems. We define the addition on naturals as well as the function *double* with their usual meanings.

$r : +(n : 0, m : \bullet) \to r \gg m$
$r : +(n : succ(p : \bullet), m : \bullet) \to q : succ(k : +(p, m)); r \gg q$
$r : double(n : \bullet) \to q : +(n, n); r \gg q$

Definition 7 (Matching). *Let $[l \mid c] \rightarrow r$ be a rewrite rule and g^n a rooted term-graph. We say that the left-hand side $[l \mid c]$ matches the term-graph g^n at node p, and denoted by $[l \mid c] \leq g^p$ iff p is reachable from n (i.e. $n \overset{*}{\leadsto}_g p$) and there exists a homomorphism, also called* matcher*, h from l to g^p, i.e. $h : \mathcal{N}_l \rightarrow \mathcal{N}_g$ such that $h(\mathcal{R}oot_l) = p$ and h is a solution of constraint c, i.e., $h \in sol(c)$.*

Definition 8 (Rewrite Step). *Let ρ be the rewrite rule $[l \mid c] \rightarrow r$ and g^n be a rooted term-graph. We say that g^n rewrites to g_1^m at node p by using the rule ρ iff there exists a matcher $h : l \rightarrow g^p$ which is a solution of constraint c and $g_1^m = h(r)[g^n]$. We write $g^n \rightarrow_{[p,\,[l|c]\rightarrow r]} g_1^m$, $g^n \rightarrow_p g^m$ or simply $g^n \rightarrow g^m$.*

Example 4. Let f, g and h be three defined operations specified by the following rewrite rules:

$n : f(p : 0, q : \bullet) \rightarrow n \gg p$
$n : g(p : \bullet, q : succ(m : \bullet)) \rightarrow w : 0; n \gg w$
$n : h(p : \bullet) \rightarrow u : succ(v : h(p)); n \gg u$

Let G_1^a, G_2^a, G_3^a and G_4^a be the graphs defined in Example 2. We recall their definitions below.

$G_1^a = a : succ(b : f(c : g(d : \bullet, e : h(b)), e))$
$G_2^a = a : succ(b : f(c : g(d : \bullet, u : succ(v : h(b))), u))$
$G_3^a = a : succ(b : f(w : 0, u : succ(v : h(b))))$
$G_4^a = a : succ(w : 0)$

From the rules given in this example, we can get the following derivation. Notice that we did not report the nodes which are not reachable from the roots of the considered term-graphs.

$$G_1^a \rightarrow_e G_2^a \rightarrow_c G_3^a \rightarrow_b G_4^a$$

3 Inductively Sequential Term-Graph Rewrite Systems

Inductively sequential term rewrite systems have been introduced by Antoy in [1]. Such systems are defined over constructor-based signatures. The left-hand sides of the rules are patterns of the form $f(k_1, \cdots, k_n)$ where f is a defined symbol and the sub-terms (i.e., the k_i's) are constructor terms. By definition, the rules of an inductively sequential term rewrite system are stored in data-structures called definitional trees. Thanks to these data-structures, several efficient rewriting and narrowing strategies have been devised (e.g. [1,5]).

In this section we consider a subclass of tGRSs which consists of systems that can be stored within definitional trees.

Definition 9 (Definitional tree). *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a tGRS such that Σ is a constructor-based signature. A tree $\mathcal{T}$ is a* partial definitional tree*, or* pdt*, with pattern $[\pi \mid C]$ iff one of the following cases holds:*

- $\mathcal{T} = rule([\pi \mid C] \to r)$, *where* $[\pi \mid C] \to r$ *is a variant of a rule of* $\mathcal{R}$.
- $\mathcal{T} = position.branch([\pi \mid C], o, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, *where* o *is a non-labeled node of* π, o *is of sort* s, $c_1, \ldots, c_k$ $(k > 0)$ *are different constructors of the sort* s *and for all* $j \in 1..k$, $\mathcal{T}_j$ *is a pdt with pattern* $[\pi_j|C]$, *such that* π_j *is obtained from* π *by applying an action which labels the node* o *with constructor* c_j, *i.e.,* $\pi_j = o : c_j(o_1 : \bullet, \ldots, o_n : \bullet)[\pi]$, *where* n *is the number of arguments of* c_j *and* $o_1, \ldots, o_n$ *are new nodes.*
- $\mathcal{T} = share.branch([\pi \mid C], \mathcal{T}_1, \mathcal{T}_2)$, *where* $\mathcal{T}_1$ *is a pdt with pattern* $[\pi \mid C \wedge n \neq m]$ *such that* n *and* m *are nodes occurring in* π *and the constraint* $n \neq m$ *does not occur in* C *and* $\mathcal{T}_2$ *is a pdt with pattern* $[\pi' \mid C]$ *such that* π' *is obtained from* π *by collapsing the two nodes* n *and* m *(and their successors). I.e.* π' *is obtained by encoding the constraint* $n \doteq m$ *into* π.

We write $pattern(\mathcal{T})$ *to denote the pattern argument of a pdt.*
A definitional tree $\mathcal{T}$ *of a defined operation* f *is a* finite *pdt with a pattern of the form* $[p : f(o_1 : \bullet, \ldots, o_n : \bullet) \mid true]$, *also denoted by* $p : f(o_1 : \bullet, \ldots, o_n : \bullet)$, *where* n *is the number of arguments of* f, $p, o_1, \ldots, o_n$ *are new nodes, and for every rule* $[l \mid C] \to r$ *of* $\mathcal{R}$, *with* l *of the form* $f(g_1, \ldots, g_n)$, *there exists a leaf* $rule([l' \mid C'] \to r')$ *of* $\mathcal{T}$ *such that* $[l' \mid C'] \to r'$ *is a variant of* $[l \mid C] \to r$.

Example 5. We consider the auxiliary operation $length'$ defined in Example 3. We recall first its rules and provide a definitional tree for it. We give only the pattern or the rule for every node of the tree.

(Rule1) $r : length'(p_1 : nil, p_2 : \bullet) \to r' : 0; r \gg r'$

(Rule2) $r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_2) \to r' : s(0); r \gg r'$

(Rule3) $[r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \neq p_3] \to r' : s(q : \bullet); q : length'(p_2, p_3); r \gg r'$

Readers familiar with classical definitional trees [1] should notice the introduction of a new kind of nodes called *share.branch*. In the context of term-graph rewriting, sharing of data-structures plays an important role which cannot be handled easily in the framework of term (tree) rewriting. The addition of the nodes *share.branch* still ensures the property of non overlapping of the patterns situated at the leaves of a definitional tree. We can easily prove the following statement.

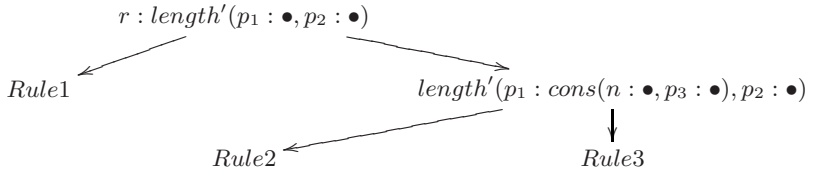

Fig. 5. A definitional tree of operation $length'$

Proposition 1. *Let $\mathcal{T}$ be a definitional tree of a defined operation f. Let $[l_1 \mid c_1] \to r_1$ and $[l_2 \mid c_2] \to r_2$ be two different rules of $\mathcal{T}$. Then, the left-hand sides $[l_2 \mid c_2]$ and $[l_1 \mid c_1]$ do not overlap. I.e., there exist no term-graph g, and matchers $h_1 : l_1 \to g$ and $h_2 : l_2 \to g$ which fulfil respectively constraints c_1 and c_2.*

Hereafter, we define the rewrite strategy Φ induced by definitional trees. We start by the following technical definition of constructor paths.

Definition 10 (Constructor Path). *We will say that a node p is reachable from a node n_0 in a term-graph g through a* constructor path *iff there exists a path in g, say $n_0 \rightsquigarrow_g n_1 \rightsquigarrow_g \ldots \rightsquigarrow_g n_k \rightsquigarrow_g p$ such that, for all $i \in 0..k$, $\mathcal{L}_g(n_j)$ is a constructor symbol ($\in \mathcal{C}$).*

Definition 11 (A term-graph rewrite strategy). *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a tGRS such that Σ is constructor-based and the rules of every defined operation are stored in a definitional tree. Let g^n be a rooted term-graph. Let p be a reachable node from the root n through a constructor path in g^n such that p is labeled by a defined operation f and let $\mathcal{T}_f$ be a definitional tree of f. Φ is the partial function defined by $\Phi(g^n) = \varphi(g^p, \mathcal{T}_f)$.*

Below, we define the partial function φ. Let g^n be a rooted term-graph such that $\mathcal{L}_{g^n}(n) \in \mathcal{D}$ (i.e. the root n is labeled with a defined operation) and $\mathcal{T}$ a pdt such that $pattern(\mathcal{T}) \leq g^n$. When it is defined, the value $\varphi(g^n, \mathcal{T})$ is a pair (p, R) such that the term-graph g^n can be reduced at node p using the rule R. More precisely, $\varphi(g^n, \mathcal{T})$ is defined as follows:

$$\varphi(g^n, \mathcal{T}) = \begin{cases} (n, [\pi' \mid C'] \to r') & \text{if } \mathcal{T} = rule([\pi \mid C] \to r) \text{ and} \\ & \quad [\pi' \mid C'] \to r' \text{ is a variant of } [\pi \mid C] \to r \text{ ;} \\ \varphi(g^n, \mathcal{T}_i) & \text{if } \mathcal{T} = share.branch([\pi \mid C], \mathcal{T}_1, \mathcal{T}_2) \text{ for} \\ & \quad \text{the unique } i \text{ such that } pattern(\mathcal{T}_i) \leq g^n \text{ and } i \in 1..2; \\ \varphi(g^n, \mathcal{T}_i) & \text{if } \mathcal{T} = position.branch([\pi \mid C], o, \mathcal{T}_1, \ldots, \mathcal{T}_k) \text{ for} \\ & \quad \text{the unique } i \text{ such that } pattern(\mathcal{T}_i) \leq g^n \text{ and } i \in 1..k; \\ (p, R) & \text{if } \mathcal{T} = position.branch([\pi \mid C], o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad [\pi \mid C] \text{ matches } g^n \text{ at the root } n \text{ by} \\ & \quad \text{homomorphism } h : \pi \to g, \\ & \quad h(o) \text{ is labeled with a defined operation } f \text{ (in } g), \\ & \quad \mathcal{T}' \text{ is a definitional tree of } f \text{ and} \\ & \quad \varphi(g^{h(o)}, \mathcal{T}') = (p, R). \end{cases}$$

Example 6. We illustrate the use of the strategy Φ. We consider again the operations and the rules given in Example 4.

(R1) $n : f(p : 0, q : \bullet) \to n \gg p$
(R2) $n : g(p : \bullet, q : succ(m : \bullet)) \to w : 0; n \gg w$
(R3) $n : h(p : \bullet) \to q : succ(m : h(p)); n \gg q$

First, we provide a definitional tree for each operation.

$\mathcal{T}_f = position.branch(n : f(p : \bullet, q : \bullet), p, rule(R1))$
$\mathcal{T}_g = position.branch(n : g(p : \bullet, q : \bullet), q, rule(R2))$
$\mathcal{T}_h = rule(R3)$

The following derivation given in Example 4 is developed by the strategy Φ.

$$G_1^a \rightarrow_e G_2^a \rightarrow_c G_3^a \rightarrow_b G_4^a$$

One can easily verify the following equalities:

$\Phi(G_1^a) = (e,\text{R3})$
$\Phi(G_2^a) = (c,\text{R2})$
$\Phi(G_3^a) = (b,\text{R1})$

The aim of the definition of the strategy Φ is to compute needed nodes to be contracted during the transformation of a term-graph. We define below the notions of needed nodes and outermost nodes in the framework of term-graph rewriting.

Definition 12 (needed node, outermost redex). *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a tGRS such that Σ is constructor-based. Let g_1^n and g_2^m be two term-graphs and $B = g_1^n \xrightarrow{*} g_2^m$ a rewrite derivation. A node q labeled with a defined operation in g_1^n and reachable from the root n* is a residual node by *B if q remains reachable from the root m in g_2^m. Then, we call* descendant of *g_1^q the rooted term-graph g_2^q. A node q in g is* needed *iff in every rewrite 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^n is an* outermost node *of g^n iff $q = n$ or q is reachable from n through a constructor path. A redex u rooted by q in g^n is an* outermost redex *iff $q = n$ or q is reachable from n through a path $p_0 \rightsquigarrow_{g^n} p_1 \rightsquigarrow_{g^n} \ldots \rightsquigarrow_{g^n} p_k$ such that $p_0 = n$, $p_k = q$ and g^{p_i} is not a redex for all $i \in 0..(k-1)$.*

Unlike the case of terms, we show in the following proposition that, in general, the strategy Φ does not compute needed nodes when it is applied on term-graphs. We will give later in Definition 13 sufficient conditions which ensure the neededness of the nodes computed by the strategy Φ.

Proposition 2. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a tGRS such that Σ is constructor-based and the rules of every defined operation are stored in a definitional tree. Let g^n be a rooted term-graph. Then,*

1. *the computation of $\phi(g^n)$ may be infinite.*
2. *if $\phi(g^n) = (p, R)$, the node p is not needed in general.*
3. *if $\phi(g^n)$ is not defined, g^n can still have a constructor normal form.*

Proof. The proof is given by counter examples. Let us consider the following tGRS which satisfies the conditions of the proposition.

$$r : f_1(p : 0) \rightarrow r \gg p$$
$$r : f_1(p : succ(p' : \bullet)) \rightarrow r \gg p$$
$$r : h_1(p : 0, q : succ(n : \bullet)) \rightarrow q \gg p$$
$$r : g_1(p : 0) \rightarrow r \gg p$$

1. Let E^n be the term-graph $n : f_1(m : f_1(n))$. Then, by definition of the strategy ϕ, $\phi(E^n) = \varphi(E^n, \mathcal{T}_{f_1})$, for some definitional tree, $\mathcal{T}_{f_1}$, of f_1. By definition of φ and from the patterns of the rules defining f_1, one can easily verify that $\phi(E^n)$ does not halt.
2. Let $G^n = n : succ(r : succ(p : f_1(q : succ(s : h_1(u : 0, r)))))$. We can easily verify that $\phi(G^n) = (p, r_1 : f_1(p_1 : succ(p'_1 : \bullet)) \rightarrow r_1 \gg p_1)$. However, the node p is not needed in G^n since one may obtain the desired normal form $n : succ(u : 0)$ after one rewrite step performed at node s.
3. Let us consider the graph $H^n = n : succ(r : succ(p : g_1(q : succ(s : h_1(u : 0, r)))))$. Then, $\phi(H^n) = \varphi(H^p, \mathcal{T}_{g_1})$ is not defined for any definitional tree $\mathcal{T}_{g_1}$. However, if we rewrite H^n at node s we get a constructor normal form $H^n_1 = n : succ(u : 0)$.

To overcome the issues pointed by Proposition 2, we propose below sufficient syntactic conditions over rewrite rules.

Definition 13 (inductively sequential tGRS). *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be a tGRS. SP is called* inductively sequential *tGRS iff (i) the signature Σ is constructor-based, (ii) the rules which define every defined operation are stored in a definitional tree and (iii) the nodes which can be subject to local or global redirections are the roots of the left-hand sides of the rules. That is to say, for all rules $[\pi \mid C] \rightarrow r$ in $\mathcal{R}$, for all global (respectively, local) redirections of the form $p \gg q$ (respectively, $p \gg_i q$ for some i), occurring in the right-hand side r, we have $p = Root_\pi$.*

Example 7. The rewrite systems given in Example 3 and Example 4 are all inductively sequential but the one which defines the operation *reverse*.

The following proposition summarizes the main properties of φ in presence of inductively sequential term-graph rewrite systems.

Proposition 3. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an inductively sequential tGRS, f a defined operation, $\mathcal{T}_f$ a definitional tree of f, and g^n a rooted term-graph whose root is labeled with f (i.e. $\mathcal{L}_{g^n}(n) = f$). If $\varphi(g^n, \mathcal{T}_f) = (p, R)$, then (i) in every rewrite derivation from g^n to a constructor-rooted term-graph, a descendant of g^p is rewritten at the root p, 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^n. (iv) If $\varphi(g^n, \mathcal{T})$ is not defined, then g^n cannot be rewritten into a constructor-rooted term-graph.*

Theorem 1. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an inductively sequential tGRS, and g^n a rooted term-graph. If $\Phi(g) = (p, R)$, then g^p is an outermost needed redex of g^n and g^n can be rewritten at node p with rule R. If $\Phi(g)$ is not defined, then g^n cannot be rewritten into a constructor term-graph.*

4 Confluence

In this section, we consider the property of confluence which could be of great interest for deterministic computations. Ensuring confluence in presence of term-graph rewrite systems is not an easy task (see e.g., [17]). For example, a rewrite system as simple as the two following rules $f(x) \rightarrow x$ and $g(x) \rightarrow x$ is not confluent. Indeed, the term-graph $n : f(m : g(n))$ can be reduced to two different term-graphs $n : f(n)$ and $m : g(m)$. The two last term-graphs cannot be reduced to a common term-graph. In [9,10], a subclass of circular term-graphs, called admissible term-graphs, has been introduced. It has been shown that, for a large class of term-graph rewrite systems, the rewrite relation induced over admissible term-graphs is confluent. In this section, we generalise that result to the admissible inductively sequential tGRSs.

Definition 14 (admissible rooted term-graph). *[9,10] A rooted term-graph g^n is admissible iff for all nodes m, labeled by a defined operation (i.e., $\mathcal{L}_{g^n}(m) \in \mathcal{D}$), m is not reachable from itself (i.e., m does not belong to a cycle $m \overset{+}{\rightarrow} m$).*

Definition 15 (admissible inductively sequential tGRS). *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an inductively sequential tGRS. SP is called admissible iff for all rules $[\pi \mid C] \rightarrow r$ in $\mathcal{R}$ the following conditions are satisfied*

- *for all global (respectively, local) redirections of the form $p \gg q$ (respectively, $p \gg_i q$ for some i), occurring in the right-hand side r, we have $p = Root_\pi$ and $q \neq Root_\pi$.*
- *for all actions of the form $\alpha : f(\beta_1, \ldots, \beta_n)$, for all $i \in 1..n$, $\beta_i \neq Root_\pi$*
- *the set of actions of the form $\alpha : f(\beta_1, \ldots, \beta_n)$, appearing in r, do not construct a cycle consisting only of newly introduced nodes in r and including a node labeled with a defined operation. If we denote by $\rightsquigarrow_r$ the reachability over the new nodes introduced in r, this condition could be specified as : for all nodes, α, introduced in r and labeled by a defined operation, $\alpha \overset{+}{\rightsquigarrow}_r \alpha$.*
- *Constraint C includes disequations of the form $p \neq q$ where p and q are labeled by constructor symbols.*

Example 8. All the previous inductively sequential systems are admissible or can be modified to fulfil the required conditions. Below we provide an admissible inductively sequential tGRS which defines equality over naturals.

$p : eq(n : \bullet, n) \rightarrow q : true; p \gg q$

$[p : eq(n : 0, m : 0) \mid n \neq m] \rightarrow q : true; p \gg q$

$[p : eq(n : succ(n' : \bullet), m : succ(m' : \bullet)) \mid n \neq m] \rightarrow p \gg_1 n'; p \gg_2 m'$

$p : eq(n : succ(n' : \bullet), m : 0) \rightarrow q : false; p \gg q$

$p : eq(n : 0, m : succ(m' : \bullet)) \rightarrow q : false; p \gg q$

The following proposition states that the class of admissible term-graphs is closed under the rewrite relation induced by an admissible inductively sequential tGRS.

Proposition 4. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an admissible inductively sequential tGRS and g^n an admissible rooted term-graph. If g^n rewrites to g^m via a rewrite rule in $\mathcal{R}$, then g^m is also an admissible rooted term-graph.*

Definition 16 (Confluence). *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an admissible inductively sequential tGRS. We say that* the rewriting relation $\rightarrow$ is confluent w.r.t the class of admissible term-graphs *iff for all rooted admissible term-graphs g_1^n, $g_2^{n'}$, g_3^m and $g_4^{m'}$ such that g_1^n and $g_2^{n'}$ are identical up to renaming of nodes ($g_1^n \sim g_2^{n'}$), $g_1^n \stackrel{*}{\rightarrow} g_3^m$ and $g_2^{n'} \stackrel{*}{\rightarrow} g_4^{m'}$, there exist two admissible graphs g_5^o and $g_6^{o'}$ such that $g_3^m \stackrel{*}{\rightarrow} g_5^o$, $g_4^{m'} \stackrel{*}{\rightarrow} g_6^{o'}$ and $g_5^o \sim g_6^{o'}$.*

We state below a new confluence result regarding the class of admissible inductively sequential tGRS. The reader familiar with the confluence property may notice that systems in this class are not always confluent modulo bisimilarity (two term-graphs are said bisimilar iff they represent the same rational term). For instance the application of the operation *length*, as defined in Example 3, to two bisimilar and non isomorphic lists, should yield different values.

Theorem 1. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an admissible inductively sequential tGRS. Then the rewriting relation $\rightarrow$ is confluent w.r.t the class of admissible term-graphs.*

The proof of Theorem 1 is obtained by classical induction on the length of the considered rewrite derivations and leans basically on the following key result.

Lemma 1. *Let $SP = \langle \Sigma, \mathcal{R} \rangle$ be an admissible inductively sequential tGRS and g^n, g_1^m and g_2^o be three admissible term-graphs. If $g^n \rightarrow g_1^m$ and $g^n \rightarrow g_2^o$, then there exist two graphs g_3^p and g_4^q such that $g_1^m \stackrel{\varepsilon}{\rightarrow} g_3^p$, $g_2^o \stackrel{\varepsilon}{\rightarrow} g_4^q$ and g_3^p and g_4^q are equal up to renaming of nodes ($g_3^p \sim g_4^q$). The notation $g \stackrel{\varepsilon}{\rightarrow} g'$ means that g' is either g (zero rewrite step) or it is obtained from g after one rewrite step.*

5 Conclusion

Definitional trees [1] give rise to efficient rewrite and narrowing strategies. In this paper we investigated ways to use Definitional trees with the aim to propose new efficient strategies for term-graph rewriting. We succeeded to show the computation of needed redexes in the particular class of inductively sequential tGRSs. We gave also counter-examples illustrating some negative results. These results give an idea about the limits of the use of Definitional trees in the context of term-graph rewriting. On the other hand, we proposed a new class of admissible term-graph rewrite systems for which the rewrite relation is confluent with respect to admissible term-graphs and for which Definitional trees still behave nicely. The presented results open some directions of work. In

[11], a general narrowing procedure has been proposed. The class of inductively sequential tGRSs seem to be a good candidate to develop an efficient narrowing strategy for term-graphs. Abstraction techniques has been successfully used in the context of term rewrite systems (see, e.g., [7,15]). Extensions of abstraction methods to term-graph rewrite systems worth also to be investigated.

References

1. Antoy, S.: Definitional trees. In: Kirchner, H., Levi, G. (eds.) ALP 1992. LNCS, vol. 632, pp. 143–157. Springer, Heidelberg (1992)
2. Antoy, S.: Evaluation strategies for functional logic programming. Journal of Symbolic Computation 40(1), 875–903 (2005)
3. Antoy, S., Braßel, B.: Computing with subspaces. In: Proc. of the 9th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP 2007), pp. 121–130. ACM Press, New York (2007)
4. Antoy, S., Brown, D.W., Chiang, S.-H.: Lazy context cloning for non-deterministic graph rewriting. Electr. Notes Theor. Comput. Sci. 176(1), 3–23 (2007)
5. Antoy, S., Echahed, R., Hanus, M.: A needed narrowing strategy. J. ACM 47(4), 776–822 (2000)
6. Barendregt, H., van Eekelen, M., Glauert, J., Kenneway, R., Plasmeijer, M.J., Sleep, M.: Term graph rewriting. In: de Bakker, J.W., Nijman, A.J., Treleaven, P.C. (eds.) PARLE 1987. LNCS, vol. 259, pp. 141–158. Springer, Heidelberg (1987)
7. Bert, D., Echahed, R.: Abstraction of Conditional Term Rewriting Systems. In: Lloyd, J. (ed.) Proc. of International Logic Programming Symposiumon (ILPS), pp. 162–176. MIT Press, Cambridge (1995)
8. Duval, D., Echahed, R., Prost, F.: Modeling pointer redirection as cyclic term-graph rewriting. Electr. Notes Theor. Comput. Sci. 176(1), 65–84 (2007)
9. Echahed, R., Janodet, J.C.: Admissible graph rewriting and narrowing. In: Proc. of Joint International Conference and Symposium on Logic Programming (JICSLP 1998), pp. 325–340. MIT Press, Cambridge (1998)
10. Echahed, R., Janodet, J.-C.: Parallel admissible graph rewriting. In: Fiadeiro, J.L. (ed.) WADT 1998. LNCS, vol. 1589, pp. 122–137. Springer, Heidelberg (1999)
11. Echahed, R., Peltier, N.: Narrowing data-structures with pointers. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 92–106. Springer, Heidelberg (2006)
12. Echahed, R., Peltier, N.: Non strict confluent rewrite systems for data-structures with pointers. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 137–152. Springer, Heidelberg (2007)
13. Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.): Handbook of Graph Grammars and Computing by Graph Transformations, Applications, Languages and Tools, vol. 2. World Scientific, Singapore (1999)
14. Ehrig, H., Kreowski, H.-J., Montanari, U., Rozenberg, G. (eds.): Handbook of Graph Grammars and Computing by Graph Transformations, Concurrency, Parallelism and Distribution, vol. 3. World Scientific, Singapore (1999)
15. Hanus, M.: Call pattern analysis for functional logic programs. In: Proc. of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP 2008) (July 2008) (to appear)
16. Hußmann, H.: Nondeterministic algebraic specifications and nonconfluent term rewriting. J. Log. Program 12(3&4), 237–255 (1992)

17. Kennaway, J.R., Klop, J.K., Sleep, M.R., Vries, F.J.D.: On the adequacy of graph rewriting for simulating term rewriting. ACM Transactions on Programming Languages and Systems 16(3), 493–523 (1994); previous version: Technical Report CS-R9204, CWI, Amsterdam (1992)
18. López-Fraguas, F.J., Rodríguez-Hortalá, J., Sánchez-Hernández, J.: A simple rewrite notion for call-time choice semantics. In: PPDP, the 9th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, Wroclaw, Poland, July 14-16, 2007, pp. 197–208. ACM, New York (2007)
19. Plump, D.: Term graph rewriting. In: Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation, vol. 2, pp. 3–61. World Scientific, Singapore (1999)
20. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformations, Foundations, vol. 1. World Scientific, Singapore (1997)