Narrowing Data-Structures with Pointers[⋆]

Rachid Echahed and Nicolas Peltier

LEIBNIZ-IMAG, CNRS
46, avenue Félix Viallet
38031 Grenoble Cedex, France
Rachid.Echahed@imag.fr, Nicolas.Peltier@imag.fr

Abstract. We investigate the narrowing relation in a wide class of (cyclic) term-graph rewrite systems. We propose a new sound and complete narrowing-based algorithm able to solve goals in presence of data structures with pointers (e.g., circular lists, doubly linked lists etc.). We first define the class of rewrite systems we consider. Our rules provide features such as pointer (edge) redirections, relabeling of existing nodes, in addition to the creation of new nodes. Moreover, we split the set of nodes of term-graphs in two (possibly empty) subsets: (i) variables and (ii) names. Variable nodes can be mapped against any other node whereas names act as constants and thus they are supposed to match themselves. This distinction between nodes allows us to synthesize, through the narrowing process, data-structures with circular shapes. In a second step, we define the rewriting and narrowing relations. We then show the soundness and completeness of narrowing.

1 Introduction

Narrowing is the heart of the operational semantics of declarative languages which integrate functional and logic programming paradigms [10]. Programs in these languages are term rewrite systems. Their operational semantics consists in solving goals. For example, let us consider the following program which defines the length of a sequence:

$$\mathit{length}(\mathit{nil}) \to 0 \qquad\qquad \mathit{length}(\mathit{cons}(x,u)) \to s(\mathit{length}(u)).$$

To solve the goal $\mathit{length}(\mathit{cons}(x,\mathit{nil})) = z$, one may normalize the term $\mathit{length}(\mathit{cons}(x,\mathit{nil}))$ and gets the unique solution $z = s(0)$. But the following goal $\mathit{length}(u) = s(s(0))$ cannot be solved by simple normalization ; instead narrowing can be used to synthesize the answer $u = \mathit{cons}(x_1, \mathit{cons}(x_2, \mathit{nil}))$ where x_1 and x_2 are *fresh* variables.

Narrowing has been widely investigated in the framework of first order term rewrite systems and optimal strategies have been proposed (e.g. [2]). In this paper, we propose to extend narrowing to a large class of term-graph rewrite systems. There are at least two reasons that motivate our work. First, efficient implementation techniques of declarative languages use dags (directed acyclic graphs) to implement terms. Second, in recent developments of graph transformations [3, 4, 6], it is shown that data-structures with pointers could be handled by using graph rewrite systems, and thus rule-based languages such as declarative

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

ones could benefit from such results in order to fully integrate cyclic term-graphs (with pointers), such as circular lists or doubly linked lists, as first class objects.

In this paper, we consider term-graph rewrite systems composed of rules of the following shape (see Section 4 for details): $L \to A$, where L is a (constrained) term-graph and A is a sequence of actions the aim of which is the construction of the right hand side. Roughly speaking, A could be split into two parts, say R and D, where R is a term-graph and D is a sequence of edge (pointer) redirections. For example, the following rule inserts an element in a circular list (we use the classical linear [5] notation of term-graphs, where non-connected graphs are separated by ","):

$$\gamma{:}insert(a, \alpha{:}cons(b, u)), \beta{:}cons(c, \alpha) \to \gamma{:}cons(a, \alpha); \beta.2 \gg \gamma.$$

Applying this rule to the term-graph $\gamma_1{:}insert(a, \alpha_1{:}cons(b, \beta_1{:}cons(c, \alpha_1)))$ we get the intermediate term-graph $\gamma_1{:}cons(a, \alpha_1{:}cons(b, \beta_1{:}cons(c, \alpha_1)))$ before we perform the action $\beta_1.2 \gg \gamma_1$. The aim of this action is to redirect the second edge outgoing of the node β_1 in order to point the node γ_1. The final result of the application of the rule above is then the term $\gamma_1{:}cons(a, \alpha_1{:}cons(b, \beta_1{:}cons(c, \gamma_1)))$.

The rewrite rules we consider define a large class of term-graph rewrite systems (formally defined in Section 4). It includes several useful features. Left-hand sides could be cyclic with some constraints (disequations) on the nodes. Actions building the right hand side can execute redirections of pointers (edges) either locally as in the example above or globally as it happens when rewriting rooted term graphs. We have no restriction over the cyclic term-graphs to be rewritten.

Solving goals with cyclic term-graphs is certainly not an easy task. Consider for instance the operation # which computes the number of elements of a circular list (the complete definition of this operation is given in Section 5). If we consider the goal $\#(u) = s(s(0))$ then we should get a solution such as $u = \alpha{:}cons(x, \beta{:}cons(y, \alpha))$ with the constraint $\alpha \not\approx \beta$. Note that there is no published algorithm which is able to synthesize such a solution. Nodes α, β, x and y are supposed to be fresh variable nodes. The distinction between (constant) nodes and variable nodes is essential in our setting. Variable nodes behave as classical first order variables in the unification process for example, while the remaining nodes (constants) could be seen as global variables in imperative languages.

Defining narrowing in our setting turns out to be trickier than in the previous works. This is mainly due to the actions we perform on term-graphs such as pointer redirections and also to the fact that graphs are not considered equal up to bisimulation. Consider for instance the following term-graph $f(\delta{:}a, \gamma{:}a)$ where δ and γ are variable nodes, to be narrowed by using the rule $f(\alpha{:}a, \beta) \to \alpha{:}b$ (β denotes a variable). We can get two different narrowing steps

$$f(\delta{:}a, \gamma{:}a) \leadsto_{\{\delta=\gamma\}} f(\delta{:}b, \delta) \qquad \text{or} \qquad f(\delta{:}a, \gamma{:}a) \leadsto_{\{\delta\not\approx\gamma\}} f(\delta{:}b, \gamma{:}a).$$

From this simple example, we can see that instantiation of variable nodes during the narrowing process is not usual. Indeed, in contrast to the usual case, the computed solutions may include disequations, such as $\delta \not\approx \gamma$ in the second derivation above.

There are very few results in the literature on term-graph narrowing. In [12, 9, 11], acyclic term-graph narrowing have been studied and basic narrowing strategies have been proposed in [11, 9]. Cyclic term-graph narrowing was first studied in [7] in the context of weakly-orthogonal term-graph rewrite systems. Its extension with graph collapsing could be found in [8]. Optimal term-graph narrowing strategies have been proposed in [7, 8]. Very recently, [1] extended [7] and proposed efficient term-graph narrowing strategies in the presence of non-deterministic functions (i.e. non-confluent rewrite systems).

In this paper, we go beyond these results and tackle cyclic term graph narrowing in a very large class of term-graph rewrite systems that subsumes by far the weakly-orthogonal graph rewrite systems studied in [7]. We define the narrowing relation induced by the considered term-graph rewrite systems and prove its soundness and completeness.

The paper is organised as follows. Section 2 gives the precise definition of the term-graphs we consider as well as some basic definitions we need in the paper. In Section 3 we give the definitions of different actions we operate on graphs such as node creation and redefinition, pointer redirections etc. Section 4 defines the rewrite rules, rewrite steps and the term-graph rewrite systems we consider. Section 5 is dedicated to the definition of narrowing relation. The soundness and completeness of the narrowing relation are investigated in Section 6. Finally, Section 7 concludes the paper. Due to length restrictions, proofs are omitted.

2 Term-Graph

In this section, we describe the class of data structures (i.e. term-graphs) considered in the paper. The definitions are close to the ones of [5], but some of the notations are slightly adapted in order to better suit our purposes.

We assume given a set of *names* $\mathcal{A}$, a set of *variables* $\mathcal{V}$ and a set of *function symbols* Σ. We denote by $\mathcal{N}$ the set $\mathcal{N} \overset{\text{def}}{=} \mathcal{A} \cup \mathcal{V}$. $\mathcal{N}$ is the set of *nodes*.

Definition 1. *(Term-Graph) A* reference *on a set of nodes $N \subseteq \mathcal{N}$ is an expression of the form $f(\alpha_1, \ldots, \alpha_n)$ where $f \in \Sigma$, $n \geq 0$ and $\alpha_1, \ldots, \alpha_n \in N$ (if $n = 0$ then $f(\alpha_1, \ldots, \alpha_n)$ should be written f). The set of references on a set of nodes N is denoted by $\mathcal{T}(N)$. A* term-graph *G is defined by a set of nodes $\mathcal{N}(G) \subseteq \mathcal{N}$ and a* partial *function ref_G from $\mathcal{N}(G)$ to $\mathcal{T}(\mathcal{N}(G))$.*

For instance, a term-graph consisting in a variable node α without reference may be seen as a *variable* (in the usual sense), i.e. denotes an arbitrary ground term-graph. If α is a name, then the graph is *partially instantiated*: the name of one of its nodes is known, but its reference and its other nodes remain to be specified.

We denote by $head_G(\alpha)$ the head symbol of $ref_G(\alpha)$ (if it exists, otherwise $head_G(\alpha)$ is undefined). We denote by $dom(G)$ the set of nodes α s.t. $ref_G(\alpha)$ is defined. Note that we may have $dom(G) \neq \mathcal{N}(G)$. A term-graph G is said to be *ground* if $\mathcal{N}(G) \subseteq \mathcal{A}$[1]. We write $G \subseteq H$ iff $\mathcal{N}(G) \subseteq \mathcal{N}(H)$ and if for any $\alpha \in \mathit{dom}(G)$ we have $\alpha \in \mathit{dom}(H)$ and $\mathit{ref}_H(\alpha) = \mathit{ref}_G(\alpha)$. Intuitively, $G \subseteq H$ if G is a subgraph of H. The notion of subgraph is the analogue of the notion of subterm for usual terms. In what follows, we always denote nodes (variables and names) by Greek letters $\alpha, \beta, \ldots$, function symbols by $f, g, \ldots$ and constant symbols by $a, b, \ldots$.

Although Definition 1 is useful from a theoretical point of view, in the forthcoming examples, we adopt a more convenient and readable (commonly used, see for instance [5]) linear notation for term-graphs. We write a term-graph as a standard term, but we prefix some of the subterms (those occurring several times in the considered term-graph) by nodes. Obviously, naming (i.e. prefixing) subterms with nodes allows one to share subterms and to denote infinite (rational) terms. For instance, the expression $\alpha{:}f(a, g(\alpha))$ denotes a (cyclic) term-graph s.t.: $\mathit{dom}(G) = \{\alpha, \beta, \gamma\}$, $\mathit{ref}_G(\alpha) = f(\beta, \gamma)$, $\mathit{ref}_G(\beta) = a$, $\mathit{ref}_G(\gamma) = g(\alpha)$ (β, γ are arbitrarily chosen nodes distinct from α). Depending on the context the unnamed nodes β, γ could be constants or variables. Note that the above term-graph could also be written $\alpha{:}f(\beta{:}a, \gamma{:}g(\alpha))$, but for the sake of clarity, we prefer to skip useless names. Two distinct names necessarily correspond to distinct nodes, whereas two distinct variables can be made identical by instantiation. For instance, let us consider the following term-graph $G = \alpha{:}\mathit{cons}(1, \beta{:}\mathit{cons}(1, \alpha))$. If α, β are variables, then β may be instantiated by α. Thus the term-graph $\delta{:}\mathit{cons}(1, \delta)$ is an *instance* of G. More precisely, G denotes a circular list of length either 1 or 2. In contrast, if α, β are distinct names, G denotes a (specific) circular of length 2. The possibility of handling abstract nodes allows one to handle *partially defined data-structures*, which is absolutely essential for defining narrowing algorithms. It also allows the programmer to define more general rules, which is capital from a practical point of view (for instance we could compare two lists of integers without knowing whether they are *physically* equal or not).

A *substitution* σ is a function mapping each variable x in $\mathcal{V}$ to a node $x\sigma \in \mathcal{N}$. The *domain* of a substitution σ is denoted by $\mathit{dom}(\sigma)$ and defined as the set of variables x s.t. $x\sigma \neq x$. A substitution is said to be *ground* iff $x\sigma \in \mathcal{A}$ for any $x \in \mathit{dom}(\sigma)$. If σ, θ are two substitutions, then $\sigma\theta$ denotes the composition of σ and θ (i.e. $x\sigma\theta = \theta(\sigma(x))$). σ is said to be *more general* than θ if there is a substitution σ' s.t. $\sigma\sigma' = \theta$.

The image of a standard term by a substitution is always a term. However, in our setting, the image of a term-graph by a substitution is not necessarily a term-graph. For instance if $G = f(\alpha{:}a, \beta{:}b)$ is a term-graph where α, β are variables, then the image of G by a substitution $\sigma : \{\alpha \to \gamma, \beta \to \gamma\}$ is not a term-graph. Thus, we can instantiate a term-graph G by a substitution σ only if σ is *compatible* with the term-graph in the sense that if two variables are mapped to the same node then the corresponding references (if they exist) must be the same. Formally, a substitution σ is said to be *compatible* with a graph G iff for any $\alpha, \beta \in \mathit{dom}(G)$ s.t. $\alpha\sigma = \beta\sigma$ we have $\mathit{ref}_G(\alpha)\sigma = \mathit{ref}_G(\beta)\sigma$.

[1] This is not equivalent to the usual notion of "ground term" because the nodes do not need to be associated to a reference.

If σ is compatible with G, then we denote by $G\sigma$ the graph H s.t.: $\mathcal{N}(H) = \{\alpha\sigma \mid \alpha \in \mathcal{N}(G)\}$ and for any $\alpha \in dom(G)$, $ref_H(\alpha\sigma) \overset{\text{def}}{=} ref_G(\alpha)\sigma$. Note that $G\sigma$ is well-defined if σ is compatible with G, since by definition $\alpha\sigma = \beta\sigma \Rightarrow ref_G(\alpha)\sigma = ref_G(\beta)\sigma$. H is called an *instance* of G iff there exists a substitution σ compatible with G s.t. $G\sigma \subseteq H$.

3 Graph Transformation

We introduce some basic operations on term-graphs: creation of a new node, node redefinition (i.e. replacement of the reference associated to an existing node by a new reference) and global redirection (i.e. redirection of all edges pointing to a node α to a node β). Node redefinition subsumes in particular edge redirection (i.e. redirection of an existing edge). For every action a, we shall denote by $G_{[a]}$ the result of the application of the action a on the term-graph G. The actions and their applications are defined in the following sections.

A **node creation** is an expression of the form α^+ where α is a node in $\mathcal{V}$. Applying a node creation to a term-graph simply adds a new node in the term-graph (with no reference).

We assume given an (infinite) subset of $\mathcal{A}$, denoted by $\mathcal{C}$ and a total precedence $\prec$ among elements of $\mathcal{C}$. Every created node is associated to a name in $\mathcal{C}$.

If G be a graph and $\alpha \in \mathcal{V}$ then $G_{[\alpha^+]}$ denotes the term-graph H s.t.:

- $\mathcal{N}(H) \overset{\text{def}}{=} \mathcal{N}(G) \cup \{NewNode(G)\}$, where $NewNode(G)$ denotes the smallest (according to $\prec$) node in $\mathcal{C}$ not occurring in G.
- For every node $\beta \in dom(G)$, $ref_H(\beta) \overset{\text{def}}{=} ref_G(\beta)$.
- $ref_H(NewNode(G))$ is undefined.

Note that H does not depend on α. As we shall see, α will be instantiated by $NewNode(G)$ which is useful only when applying a *sequence* of actions.

A **node redefinition** is a pair $\alpha{:}r$ where α is a node in $\mathcal{V}$ and r a reference. We denote by $G_{[\alpha:r]}$ the term-graph H defined as follows:

- $\mathcal{N}(H) \overset{\text{def}}{=} \mathcal{N}(G)$.
- For every node $\beta \in dom(G)$, if $\beta \neq \alpha$ then $ref_H(\beta) \overset{\text{def}}{=} ref_G(\beta)$.
- $ref_H(\alpha) \overset{\text{def}}{=} r$.

For instance, $\beta{:}f(\alpha, \delta{:}a)_{[\alpha:f(\delta,\alpha)]} = \beta{:}f(\alpha{:}f(\delta{:}a, \alpha), \delta)$. Note that we may have $\alpha \in dom(G)$ (in this case α is redirected) or $\alpha \notin dom(G)$ (in this case new edges and label are created). Note that a node redefinition does not introduce new nodes in the term-graph(this has to be done before by the node creation action).

An **edge redirection** may be seen as a particular case of node redefinition in which a unique edge is redirected. It is an expression of the form $\alpha.i \gg \beta$, where $i \in \mathbb{N}$, $\alpha \in \mathcal{N}$ and $\beta \in \mathcal{N}$. Applying an edge redirection to a term-graph consists in redirecting the i-th argument of the node α to point to the node β.

If G is a term-graph and $\alpha, \beta \in \mathcal{N}(G)$, where $\alpha \in \mathit{dom}(G)$, then $G_{[\alpha.i \gg \beta]}$ denotes the graph H defined as follows: $H \overset{\text{def}}{=} G_{[\alpha:f(\beta_1,\ldots,\beta_{i-1},\beta,\beta_{i+1},\ldots,\beta_n)]}$ where $f(\beta_1, \ldots, \beta_n) = \mathit{ref}_G(\alpha)$. Note that if $n < i$ then by convention $f(\beta_1, \ldots, \beta_{i-1}, \beta, \beta_{i+1}, \ldots, \beta_n) = f(\beta_1, \ldots, \beta_n)$ thus $H = G$.

For instance, $\beta{:}f(\alpha{:}f(\delta : a, \delta), \delta)_{[\alpha.1 \gg \beta]} = \beta{:}f(\alpha{:}f(\beta, \delta{:}a), \delta)$.

A **global redirection** is an expression of the form $\alpha \gg \beta$, where $\alpha \in \mathcal{N}$ and $\beta \in \mathcal{N}$. Applying a global redirection to a term-graph consists in redirecting any edge pointing to α to the node β, i.e. in replacing any occurrence of α in a reference in G by β.

If G is a term-graph and $\alpha, \beta \in \mathcal{N}(G)$ then $G_{[\alpha \gg \beta]}$ denotes the graph H defined as follows: $\mathcal{N}(H) \overset{\text{def}}{=} \mathcal{N}(G)$ and for every node $\gamma \in \mathit{dom}(G)$ s.t. $\mathit{ref}_G(\gamma) = f(\beta_1, \ldots, \beta_n)$ then $\mathit{ref}_H(\gamma) \overset{\text{def}}{=} f(\beta'_1, \ldots, \beta'_n)$ where for every $i \in [1..n]$ we have $\beta'_i \overset{\text{def}}{=} \beta_i$ if $\beta_i \neq \alpha$ and $\beta'_i \overset{\text{def}}{=} \beta$ otherwise ($\mathit{ref}_H(\gamma)$ is undefined if $\mathit{ref}_G(\gamma)$ is undefined). This action is said to be "global" because it may affect any node in the term-graph (in the worst case all nodes may be affected). Global redirections are necessary to express easily collapsing rules of the form $f(x) \to x$ (any occurrence of $f(x)$ in the term-graph should be replaced by x).

For instance, $\beta{:}h(\delta{:}g(\alpha{:}a, \delta), \alpha)_{[\alpha \gg \beta]} = \beta{:}h(\delta{:}g(\beta, \delta), \beta), \alpha$.

An **action** is either a node creation, or an edge redirection or a node definition or a global redirection. Substitutions can be extended to sequences of actions using the following definitions (where ϵ denotes the empty sequence and $\tau.\tau'$ denotes the concatenation of τ and τ').

– $\epsilon\sigma \overset{\text{def}}{=} \epsilon$, $(a.\tau)\sigma \overset{\text{def}}{=} a\sigma.\tau\sigma$.
– $(\alpha^+)\sigma \overset{\text{def}}{=} \alpha^+$ (α is not instantiated since α is a variable denoting the new node).
– $(\alpha{:}f(\boldsymbol{\beta}))\sigma \overset{\text{def}}{=} \alpha\sigma{:}f(\boldsymbol{\beta}\sigma)$, $(\alpha \gg \beta)\sigma \overset{\text{def}}{=} \alpha\sigma \gg \beta\sigma$, $(\alpha.i \gg \beta)\sigma \overset{\text{def}}{=} \alpha\sigma.i \gg \beta\sigma$.

If τ is a sequence of actions, and G is a term-graph, then $G_{[\tau]}$ denotes the term-graph defined as follows:

– $G_{[\epsilon]} \overset{\text{def}}{=} G$
– If $a = \alpha^+$, $G_{[a.\tau]} \overset{\text{def}}{=} G_{[a]}{}_{[\tau\{\alpha \to \mathit{NewNode}(G)\}]}$. Note that α is instantiated by the new created node in the rest of the sequence (this allows one to "reuse" this node, hence to create edges starting from or pointing to this node).
– $G_{[a.\tau]} \overset{\text{def}}{=} G_{[a]}{}_{[\tau]}$, if a is not a node creation.

Informally, $G_{[\tau]}$ is obtained from G by applying the actions in τ, in the corresponding order. For instance $\alpha{:}f(\alpha, \alpha)_{[\delta^+, \delta:a, \alpha.1 \gg \delta]} = \alpha{:}f(\alpha, \alpha), \alpha'_{[\alpha':a, \alpha.1 \gg \alpha']} = \alpha{:}f(\alpha, \alpha), \alpha'{:}a_{[\alpha.1 \gg \alpha']} = \alpha{:}f(\alpha'{:}a, \alpha)$, where $\alpha' = \mathit{NewNode}(\alpha{:}f(\alpha, \alpha))$ (commas are used to separate the actions in the sequence).

Note that $G_{[\tau]}$ is not defined if τ is an action $\alpha.i \gg \beta$ s.t. $\alpha \notin \mathit{dom}(G)$. For instance $\alpha{:}a_{[\beta.1 \gg \alpha]}$ is undefined because β is not a node in $\alpha{:}a$. Otherwise, $G_{[\tau]}$ is always defined.

If τ is a sequence of actions then we denote by $r(\tau)$ the set of nodes α s.t. τ contains an action of the form $\alpha \gg \beta$ or $\alpha.i \gg \beta$ or $\alpha{:}f(\boldsymbol{\beta})$. Intuitively, $r(\tau)$ denotes the set of nodes that are affected by the sequence of actions τ.

4 Rewrite Rules

Obviously, rewrite rules operating on term-graphs should be able to check whether two nodes are equal or not. This is useful for instance when traversing a circular list: in order to avoid looping, we need to compare the current node with the initial one before proceeding to the tail of the list. These conditions correspond to *disequality constraints* between nodes, that need to be “attached” to the left-hand side of the rule. More precisely, a *node constraint* is a finite conjunction of (possibly none) disequations of the from $\alpha \not\approx \beta$, where $\alpha, \beta \in \mathcal{N}$. The empty node constraint is denoted by $\top$.

A disequation $\alpha \not\approx \beta$ is *false* if $\alpha = \beta$ and true if α, β are two distinct symbols in $\mathcal{A}$. More formally, a substitution σ is said to be a *solution* of a node constraint ϕ iff for any $(\alpha \not\approx \beta)$ occurring in ϕ we have $\alpha\sigma \neq \beta\sigma$ and $\alpha\sigma, \beta\sigma \in \mathcal{A}$. We denote by $sol(\phi)$ the set of solutions of ϕ. A substitution σ is said to be a *counter-solution* of a node constraint ϕ iff there exists $(\alpha \not\approx \beta)$ in ϕ s.t. we have $\alpha\sigma = \beta\sigma$. We denote by $csol(\phi)$ the set of counter-solutions of ϕ.

Clearly, if $\sigma \in csol(\phi)$ then $\sigma\theta \in csol(\phi)$ for any substitution θ, and $\sigma \notin sol(\phi)$. Similarly, if $\sigma \in sol(\phi)$ then $\sigma\theta \in sol(\phi)$ for any substitution θ, and $\sigma \notin csol(\phi)$. If σ is a ground substitution and $dom(\sigma)$ contains all the variables occurring in ϕ, then we have either $\sigma \in sol(\phi)$ or $\sigma \in csol(\phi)$.

A *constrained term-graph* is a pair $[\![G \mid \phi]\!]$ where G a term-graph and ϕ a node constraint. For the sake of clarity, $[\![G \mid \top]\!]$ is denoted by G.

We are now in position to introduce our notion of term-graph rewrite rule.

Definition 2. *A term-graph* rewrite rule *is an expression of the form $[\![L \mid \phi]\!] \rightarrow_\alpha R$ where:*

1. *$[\![L \mid \phi]\!]$ is a constrained term-graph (the left-hand side of the rule).*
2. *R is a sequence of actions, s.t. if R contains an action of the form $\beta.i \gg \gamma$ then $\beta \in dom(L)$*[2]*.*
3. *$\alpha \in dom(L)$ (α is the root of the rule).*

Example 1. The following rules insert an element α before a cell β in a doubly linked list δ. A doubly linked list cell is denoted by a term-graph $dll(\alpha, \beta, \delta)$ where α denotes the previous cell, β the value of the cell and δ the next cell (tail).

– $\lambda{:}insert(\alpha, \beta, \delta{:}nil) \rightarrow_\lambda \lambda \gg \delta$. If δ is nil then the result is δ[3].

[2] This ensures that the action is always applicable on L.

[3] For the sake of clarity we write the constrained term-graph $[\![\lambda{:}insert(\alpha, \beta, \delta{:}nil) \mid \top]\!]$ without brackets since its constraint part is $\top$

- $[\![\lambda{:}\mathit{insert}(\alpha, \beta, \delta{:}\mathit{dll}(\delta_1, \delta_2, \delta_3)) \mid \beta \not\approx \delta]\!] \to_\lambda \lambda.3 \gg \delta_3$. If β is distinct from δ then α must be inserted into the tail of δ.
- $\lambda{:}\mathit{insert}(\alpha, \beta, \beta{:}\mathit{dll}(\beta_1, \beta_2, \beta_3)) \to_\lambda \lambda{:}\mathit{dll}(\beta_1, \alpha, \beta), \beta_1.3 \gg \lambda, \beta.1 \gg \lambda$. Otherwise, we create a new cell $\lambda{:}\mathit{dll}(\beta_1, \alpha, \beta)$, we redirect the first argument of β to λ and the last argument of the cell before β to λ.

Note we may have $\beta = \beta_1 = \beta_3$ in the last rule (circular list of length 1). Thanks to the flexibility of our language, we do not need to give any specific rule for this particular case (this is essential from a practical point of view).

Definition 3. *(Rewriting Step) Let G be a ground term-graph. Let $\rho = [\![L \mid \phi]\!] \to_\alpha R$ be a rewrite rule. We write $G \to_\rho H$ iff the following holds:*

- *There exists a ground substitution σ of the variables occurring in L s.t. $L\sigma \subseteq G$ and $\sigma \in sol(\phi)$.*
- *$H = G_{[R\sigma]}$.*

If $\mathcal{R}$ is a set of rewrite rules, then we write $G \to_{\mathcal{R}} H$ if $G \to_\rho H$ for some $\rho \in \mathcal{R}$. As usual $\to^*_{\mathcal{R}}$ denotes the reflexive and transitive closure of $\to_{\mathcal{R}}$.

Remark 1. The substitution σ can be easily computed by using standard unification: for any $\alpha \in \mathit{dom}(L)$, one has to find a node $\beta \in \mathit{dom}(G)$ s.t. $\alpha\sigma = \beta$ and $\mathit{ref}_L(\alpha)\sigma = \mathit{ref}_G(\beta)$. Of course, there may be several solutions (as in the usual case: a term may contain several distinct subterms matched by the left-hand side of a given rule). An important difference with the usual case is that even if we fix the value of the root node α in L, there may be still several solutions, except if all the nodes in L are accessible from α.

Example 2. Let ρ be the following rule: $\alpha{:}f(\beta, \delta{:}g(\gamma)) \to_\alpha \alpha{:}h(\beta, \delta), \delta.1 \gg \alpha$. This rule transforms a term-graph $\alpha{:}f(\beta, \delta{:}g(\gamma))$ into $\alpha{:}h(\beta, g(\alpha))$. Let $G = \lambda_1{:}f(\lambda_2{:}g(\lambda_4), \lambda_3{:}g(\lambda_4{:}a))$. We apply the rule ρ on G. We denote by L the left-hand side of ρ. We try to find a substitution σ s.t. $L\sigma \subseteq G$. Since $\mathit{head}_L(\alpha) = f$, we must have $\mathit{head}_G(\alpha\sigma) = f$, thus $\alpha\sigma = \lambda_1$. Since we must have $\mathit{ref}_G(\alpha\sigma) = \mathit{ref}_{L\sigma}(\alpha\sigma)$ we have $\beta\sigma = \lambda_2$ and $\delta\sigma = \lambda_3$. Then since $\mathit{ref}_G(\lambda_3) = L\sigma(\lambda_3)$, we have $g(\lambda_4) = L(\delta)\sigma = g(\gamma)\sigma$, thus $\gamma\sigma = \lambda_4$.

Clearly, the obtained substitution satisfies the desired conditions. We obtain the term-graph:

$$\lambda_1{:}f(\lambda_2{:}g(\lambda_4), \lambda_3{:}g(\lambda_4{:}a))_{[\lambda_1:h(\lambda_2,\lambda_3), \lambda_3.1 \gg \lambda_1]}$$

$$= \lambda_1{:}h(\lambda_2{:}g(\lambda_4), \lambda_3{:}g(\lambda_4{:}a))_{[\lambda_3.1 \gg \lambda_1]} = \lambda_1{:}h(\lambda_2{:}g(\lambda_4), \lambda_3{:}g(\lambda_1)).$$

Example 3. We consider the rule ξ defined as follows: $\alpha{:}f(\beta{:}a, \delta{:}a) \to_\alpha \beta{:}b, \delta{:}c$. Let G be the term-graph $\lambda_1{:}f(\lambda_2{:}a, \lambda_2)$.

We apply ξ on G. The only possible substitution is: $\sigma = \{\alpha \mapsto \lambda_1, \beta \mapsto \lambda_2, \delta \mapsto \lambda_2\}$. We obtain the term-graph: $\lambda_1{:}f(\lambda_2{:}a, \lambda_2)_{[\lambda_2:b, \lambda_2:c]} = \lambda_1{:}f(\lambda_2{:}b, \lambda_2)_{[\lambda_2:c]} = \lambda_1{:}f(\lambda_2{:}c, \lambda_2)$.

5 Narrowing

5.1 Term-Graph Substitutions

We need to introduce some further notations. Two term-graphs G and H are said to be *disjoint* (written $G \parallel H$) if $dom(G) \cap dom(H) = \emptyset$. Two term-graphs G and H are said to be *compatible* (written $G \bowtie H$) iff for any $\alpha \in \mathcal{N}$ s.t. $ref_G(\alpha)$ and $ref_H(\alpha)$ are defined, we have $ref_G(\alpha) = ref_H(\alpha)$ (i.e. G, H coincide on the intersection of their domains). Obviously, if $G \parallel H$ then $G \bowtie H$.

If G, H are compatible then $G \cup H$ denotes the minimal term-graph G' s.t. $G \subseteq G'$ and $H \subseteq G'$ (it is clear that G' always exists).

If $G \bowtie H$ then $G \setminus H$ denotes the (minimal) term-graph I s.t. $I \parallel H$ and $H \cup I = G$.

The notion of g-substitution is the analogue of the notion of substitution for terms. When instantiating a term-graph, one has not only to specify the value of the variables occurring in it, but also to define the references corresponding to the nodes that are introduced by the substitution. Clearly, these nodes should be distinct from the ones already occurring in the considered term-graph. This is formalized by the following:

Definition 4. *(g-substitution) A* g-substitution *is a pair $\varsigma = (\sigma, G)$ where σ is a substitution and G a term-graph s.t. if $x \in dom(\sigma)$ then $x\sigma \in \mathcal{N}(G)$. A* g-substitution *of a term-graph H is a g-substitution $\varsigma = (\sigma, G)$ s.t. σ is compatible with H and $G \bowtie H\sigma$. In this case, $H\varsigma$ denotes the term-graph: $H\sigma \cup G$.*

For instance, if $H = \alpha{:}f(\beta, \delta)$ and $\varsigma = (\{\beta \to \delta\}, \delta{:}g(\delta, \lambda{:}a))$, then $H\varsigma = \alpha{:}f(\delta{:}g(\delta, \lambda{:}a), \delta)$. Note that δ cannot have a reference in H distinct from the one in ς, according to the previous definition, since it is defined in $\delta{:}g(\delta, \lambda{:}a)$.

(σ, G) is said to be *ground* if σ, G are ground. If $\varsigma = (\sigma, G)$ then σ_ς denotes the substitution σ and Gr_ς denotes the term-graph G. If ϑ is a g-substitution of Gr_ς, then $\varsigma\vartheta$ denotes the g-substitution $(\sigma_\varsigma\sigma_\vartheta, \mathrm{Gr}_\varsigma\theta \cup \mathrm{Gr}_\vartheta)$ (composition of ς and ϑ). ς is said to be *more general* than ϑ if there is a substitution ϱ s.t. $\varsigma\varrho = \vartheta$.

By a slight abuse of notation, we write $\varsigma \in sol(\phi)$ (resp. $\varsigma \in csol(\phi)$) iff $\sigma_\varsigma \in sol(\phi)$ (resp. $\sigma_\varsigma \in csol(\phi)$). Similarly, if ϕ is a node constraint (resp. a node or a sequence of actions) then $\phi\varsigma$ denotes the expression $\phi\sigma_\varsigma$ (note that if G is a term-graph, then $G\sigma_\varsigma$ is not equal to $G\varsigma$ if Gr_ς is not empty).

5.2 Symbolic Handling of Actions

We now introduce additional definitions allowing one to encode actions at the object level.

An *s-graph* is either a term-graph or an expression of the form $apply(G, \tau)$ where G is a term-graph and τ is a sequence of actions. A *g-term* is a triple $[\![G \mid \phi]\!]^\tau$ where G is an s-graph, ϕ a node constraint and τ a sequence of actions. τ denotes in some sense the "history" of G, i.e. the set of actions applied for getting G. ϕ imposes additional constraint on the variables occurring in G.

The use of g-terms allows us to handle actions in a symbolic way (i.e. without performing them explicitly). $G_{[\tau]}$ and $apply(G, \tau)$ have very different meanings: $G_{[\tau]}$ denotes the term-graph obtained by applying τ on G, whereas $apply(G, \tau)$ is merely a syntactic object. We relate these two notions by associating semantics to g-terms. If $\mathcal{G}$ is a g-term, then $value(\mathcal{G})$ is a term-graph defined as follows: $value([\![G \mid \phi]\!]^\tau) \stackrel{\text{def}}{=} G$ if G is term-graph, and $value([\![apply(G, \chi) \mid \phi]\!]^\tau) \stackrel{\text{def}}{=} G_{[\chi]}$ otherwise (note that $value([\![G \mid \phi]\!]^\tau)$ does not depend on ϕ and τ).

5.3 Narrowing Steps

Obviously, an action can be applied on a node α only if every other node β occurring in the term-graph is known to be distinct from α. Otherwise, we do not know whether β is to be redirected or not, hence we cannot apply the action. The next definition formalizes this notion.

Let a be an action s.t. $r(a) = \{\alpha\}$. A node β is said to be *a-isolated* in a constrained term-graph $[\![G \mid \phi]\!]$ iff either β is syntactically equal to α or if the application of the action a cannot affect the occurrences of β in $[\![G \mid \phi]\!]$, i.e. iff one of the following conditions holds: either $\beta = \alpha$, or $\beta \notin dom(G)$ and a is not a global redirection, or $\beta \not\approx \alpha$ occurs in ϕ, or $\alpha, \beta \in \mathcal{A}$.

A constrained term-graph $[\![G \mid \phi]\!]$ is said to be *ready* for an action a if any node in $[\![G \mid \phi]\!]$ is a-isolated. Roughly speaking a node β is a-isolated if one has enough information to decide whether β is affected by a or not. This ensures that a behaves in a similar way *for all possible values of* β.

Our narrowing algorithm uses several rules. The first one corresponds to the usual narrowing step and is defined as follows.

$$[\![G \mid \psi]\!]^\tau \leadsto_{\rho,\varsigma} [\![H \mid \psi']\!]^{\tau\sigma}$$

If:

- G is a term-graph, $\rho = [\![L \mid \phi]\!] \to_\alpha R$ is a rewrite rule.
- σ is a most general substitution compatible with L and G s.t.:
  - $L\sigma \bowtie G\sigma$ ($L\sigma$ and $G\sigma$ must be compatible),
  - $\alpha\sigma \in dom(G\sigma)$ (the root of the rule occurs in the considered term-graph).
- $H = apply(G\sigma \cup L\sigma, R\sigma)$, $G' \stackrel{\text{def}}{=} L\sigma \setminus G\sigma$, $\varsigma \stackrel{\text{def}}{=} (\sigma, G')$.
- $\psi' = \psi\sigma \wedge \phi\sigma \wedge \bigwedge_{\beta \in r(\tau\sigma), \delta \in dom(G')} \beta \not\approx \delta$.

ψ' inherits from the constraints in ψ and in ϕ. The additional disequations $\beta \not\approx \delta$ express the fact that every synthesized node δ (i.e. every node occurring in $dom(L\sigma)$ but not in $dom(G\sigma)$) should be distinct from the nodes β that have been previously redirected. This property is essential for soundness. The g-substitution (σ, G') plays a role similar to the one of unifiers in term narrowing. In contrast to the rewrite step, this first narrowing rule does not explicitly apply the sequence of actions R on the term-graph $G\sigma \cup L\sigma$ but only *encodes* them into the g-term (they will be done by forthcoming rules). The reason is that the actions are not necessarily applicable at this point, since the term-graph may not be ready for them.

Example 4. Let ρ be the following rule: $\alpha{:}f(\beta,\delta{:}g(\gamma{:}a,\zeta)) \rightarrow_\alpha \alpha{:}h(\beta,\delta), \delta.1 \gg \alpha$. Let $G = [\![\lambda_1{:}f(\lambda_2,\lambda_3) \mid \top]\!]^\tau$ (where $\lambda_1,\lambda_2,\lambda_3$ are variables). We assume that $\tau = \lambda_1{:}f(\lambda_2,\lambda_3)$ (hence λ_1 has been redirected). We apply the narrowing rule on G, using the rule ρ. We denote by L the left-hand side of ρ. We try to find a substitution σ, compatible with G and L, s.t. $\alpha\sigma$ occurs in $dom(G)$ and $G\sigma \bowtie L\sigma$. Since $head_L(\alpha) = f$, we must have $head_G(\alpha\sigma) = f$, thus $\alpha\sigma = \lambda_1$. Since we must have $ref_{G\sigma}(\alpha\sigma) = ref_{L\sigma}(\alpha\sigma)$ we have $\beta\sigma = \lambda_2$ and $\delta\sigma = \lambda_3$.

Obviously, the obtained substitution satisfies the desired conditions. The term-graph G' in the definition of the narrowing rule is $\lambda_3{:}g(\gamma{:}a,\zeta)$ (γ,ζ are variables). We obtain:

$[\![apply(\ (\lambda_1{:}f(\lambda_2,\lambda_3), \lambda_3{:}g(\gamma{:}a,\zeta))\ ,\ \lambda_1{:}h(\lambda_2,\lambda_3), \lambda_3.1 \gg \lambda_1) \mid \lambda_1 \not\approx \lambda_3 \wedge \lambda_1 \not\approx \gamma]\!]^\tau$.

Note that the disequations $\lambda_1 \not\approx \lambda_3 \wedge \lambda_1 \not\approx \gamma$ have been added in the constraint part of the g-term. This is due to the fact that since λ_1 has already been redirected, the nodes synthesized during the application of the narrowing rule should be distinct from λ_1. The actions $\lambda_1{:}h(\lambda_2,\lambda_3)$ and $\lambda_3.1 \gg \lambda_1$ are not performed at this point but only stored into the g-term.

Additional narrowing rules are required to handle g-terms of the form $[\![apply(G,\tau) \mid \phi]\!]^\xi$, i.e. to explicitly apply the actions introduced by the previous rule. The first one – denoted by T – is trivial: it simply transforms a g-term into a term-graph in case of empty sequences of actions.

$$\boxed{[\![apply(G,\epsilon) \mid \phi]\!]^\tau \leadsto_{T,\emptyset} [\![G \mid \phi]\!]^\tau.}$$

The second and third rules, respectively denoted by A and A^+, apply the first action in the sequence and then proceed to the next ones (assuming that the considered term-graph is ready for this action). A^+ handles node creations, whereas A handles all other actions.

$$[\![apply(G,a.\tau) \mid \phi]\!]^\xi \leadsto_{A,\emptyset} [\![apply(G_{[a]},\tau) \mid \phi]\!]^{\xi.a}.$$

If a is not a node creation, $[\![G \mid \phi]\!]$ is ready for a.

$$[\![apply(G,\alpha^+.\tau) \mid \phi]\!]^\xi \leadsto_{A^+,\sigma} [\![apply(G_{[\alpha^+]},\tau\sigma) \mid \phi \wedge \bigwedge_{\beta \in \mathcal{V} \cap \mathcal{N}(G)} \beta \not\approx \gamma]\!]^{\xi.\gamma^+}$$

If $\sigma = \{\alpha \rightarrow \gamma\}$ where $\gamma = NewNode(G)$.

The added disequations ensure that the variables already present in the term-graph are distinct from the newly created node γ (this is essential to prevent these variables to be unified with γ afterwards).

The above rules are clearly not sufficient to ensure completeness. Consider for instance the following rule: $\lambda{:}f(\alpha{:}a,\beta) \rightarrow_\lambda \alpha{:}b$. Assume we want to apply the narrowing rule on the g-term $f(\delta{:}a,\delta'{:}a)$, where δ,δ' denote variables. Then the narrowing rule cannot apply, because we do not know at this point whether $\delta = \delta'$ or not. If $\delta = \delta'$ then the term-graph should be reduced to: $f(\delta{:}b,\delta)$. Otherwise, we should get: $f(\delta{:}b, \delta'{:}a)$. In other words, the considered term-graph is not ready for the action $\delta{:}b$ because δ' is not isolated. Since both options are possible (namely $\delta = \delta'$ or $\delta \neq \delta'$) we need to consider the two possibilities separately, i.e. in two distinct branches. This is done by the two following branching rules, denoted by $B^{=}$ and $B^{\neq}$ respectively.

$$\boxed{\begin{array}{c} [\![\mathit{apply}(G, a.\tau) \mid \phi]\!]^{\xi} \rightsquigarrow_{B^{=},\sigma} [\![\mathit{apply}(G\sigma, (a.\tau)\sigma) \mid \phi\sigma]\!]^{\xi\sigma} \\ \textit{If } \alpha \in r(a),\ \beta \textit{ is a non } a\textit{-isolated node in } G \\ \textit{and } \sigma \textit{ is a most general substitution compatible with } G \textit{ s.t. } \alpha\sigma = \beta\sigma. \\ \\ [\![\mathit{apply}(G, a.\tau) \mid \phi]\!]^{\xi} \rightsquigarrow_{B^{\neq},\emptyset} [\![\mathit{apply}(G, a.\tau) \mid \phi \wedge \alpha \not\approx \beta]\!]^{\xi} \\ \textit{If } \alpha \in r(a),\ \beta \textit{ is a non } a\textit{-isolated node in } G. \end{array}}$$

The reader should note that in both cases, β becomes a-isolated after application of the rule (either because it is instantiated by α or because the disequation $\alpha \not\approx \beta$ is added in the constraints). Applying these rules on the term-graph $[\![\mathit{apply}(f(\delta{:}a, \delta'{:}a), \delta{:}b) \mid \top]\!]^{\emptyset}$ yields the two following g-terms: $[\![\mathit{apply}(f(\delta{:}a, \delta), \delta{:}b) \mid \top]\!]^{\emptyset} \rightsquigarrow [\![f(\delta{:}b, \delta) \mid \top]\!]^{\delta:b}$ and $[\![\mathit{apply}(f(\delta{:}a, \delta'{:}a), \delta{:}b) \mid \delta \not\approx \delta']\!]^{\emptyset} \rightsquigarrow [\![f(\delta{:}b, \delta'{:}a) \mid \delta \not\approx \delta']\!]^{\delta:b}$

Definition 5. *If $\mathcal{R}$ is a set of rewrite rules, then we write $G \rightsquigarrow_{\mathcal{R},\varsigma} H$ if $G \rightsquigarrow_{\rho,\varsigma} H$ for some $\rho \in \mathcal{R} \cup \{T, A, A^{+}, B^{\neq}, B^{=}\}$. $\rightsquigarrow^{*}_{\mathcal{R},\varsigma}$ is inductively defined as follows: $G \rightsquigarrow^{*}_{\mathcal{R},\varsigma} H$ iff either $\varsigma = \emptyset$ and $G = H$ or $G \rightsquigarrow_{\mathcal{R},\varrho} G', G' \rightsquigarrow^{*}_{\mathcal{R},\vartheta} H$, ϱ is a g-substitution of Gr_{ϑ} and $\varsigma = \varrho\vartheta$.*

We provide a detailed example of application. We define the following functions # and *equal* computing respectively the length # of a circular list (rules ρ_1, ρ_2, ρ_3) and the equality on natural numbers (ξ_1, ξ_2):

$$\begin{array}{c} \alpha{:}\#(\beta) \rightarrow_{\alpha} \alpha{:}\#'(\beta, \beta)\ \ (\rho_1) \qquad \alpha{:}\#'(\beta_1{:}\mathit{cons}(\beta_2, \beta_3), \beta_3) \rightarrow_{\alpha} \alpha'^{+}, \alpha{:}s(\alpha'), \alpha'{:}0\ \ (\rho_2) \\ [\![\beta_1{:}\#'(\beta_2{:}\mathit{cons}(\beta_3, \beta_4), \beta_5) \mid \beta_4 \not\approx \beta_5]\!] \rightarrow_{\beta_1} \alpha'^{+}, \beta_1{:}s(\alpha'), \alpha'{:}\#'(\beta_4, \beta_5)\ \ (\rho_3) \\ \alpha{:}\mathit{equal}(0, 0) \rightarrow_{\alpha} \alpha{:}\mathit{true}\ \ (\xi_1) \qquad \alpha{:}\mathit{equal}(s(\beta_1), s(\beta_2)) \rightarrow_{\alpha} \alpha{:}\mathit{equal}(\beta_1, \beta_2)\ \ (\xi_2) \end{array}$$

Assume we want to solve the goal[4]: $\gamma{:}\mathit{equal}(\gamma'{:}\#(\gamma''), s(s(0))) \rightarrow^{*} \mathit{true}$ (i.e. to find the circular lists γ'' of length 2). We assume that γ'' is a variable and γ, γ' are names. The corresponding narrowing derivation is depicted below. We get the (unique) solution: $\gamma''{:}\mathit{cons}(\beta_3, \beta_4{:}\mathit{cons}(\beta_2', \gamma''))$ where $\gamma'' \neq \beta_4$ (the other disequations are irrelevant since α' and α'' do not occur in the term-graph). In order to improve the readability we do not specify the sequence of actions occurring in the g-terms, since they can be easily recovered from the previous steps. Moreover, we only give the derivation yielding *true* (the reader can check that all the other derivations fail).

[4] There are many ways to define goals in the literature (equations, booleans, expressions,...). In this paper we rather focus on the basic narrowing steps. Additional rules may be added when needed.

$$
\begin{array}{ll}
 & \gamma{:}\mathit{equal}(\gamma'{:}\#(\gamma''), s(s(0))) \\
\rightsquigarrow_{\rho_1} & \mathit{apply}(\gamma{:}\mathit{equal}(\gamma'{:}\#(\gamma''), s(s(0))), \gamma'{:}\#'(\gamma'', \gamma'')) \\
\rightsquigarrow_{A} & \mathit{apply}(\gamma{:}\mathit{equal}(\gamma'{:}\#'(\gamma'', \gamma''), s(s(0))), \epsilon) \\
\rightsquigarrow_{T} & \gamma{:}\mathit{equal}(\gamma'{:}\#'(\gamma'', \gamma''), s(s(0))) \\
\rightsquigarrow_{\rho_3} & \llbracket \mathit{apply}(\gamma{:}\mathit{equal}(\gamma'{:}\#'(\gamma'', \gamma''{:}\mathit{cons}(\beta_3, \beta_4)), s(s(0))), \\
 & \alpha'^{+}, \gamma'{:}s(\alpha'), \alpha'{:}\#'(\beta_4, \gamma'')) \\
 & \mid \beta_4 \neq \gamma'' \rrbracket \\
\rightsquigarrow_{A^{+}} & \llbracket \mathit{apply}((\gamma{:}\mathit{equal}(\gamma'{:}\#'(\gamma''{:}\mathit{cons}(\beta_3, \beta_4), \gamma''), s(s(0))), \alpha'), \\
 & \gamma'{:}s(\alpha'), \alpha'{:}\#'(\beta_4, \gamma'')) \\
 & \mid \beta_4 \not\approx \gamma'', \beta_3 \not\approx \alpha', \beta_4 \not\approx \alpha', \gamma'' \not\approx \alpha' \rrbracket \\
\rightsquigarrow_{A} & \llbracket \mathit{apply}((\gamma{:}\mathit{equal}(\gamma'{:}s(\alpha'), s(s(0))), \gamma''{:}\mathit{cons}(\beta_3, \beta_4), \alpha'), \\
 & \alpha'{:}\#'(\beta_4, \gamma'')) \\
 & \mid \beta_4 \not\approx \gamma'', \beta_3 \not\approx \alpha', \beta_4 \not\approx \alpha', \gamma'' \not\approx \alpha' \rrbracket \\
\rightsquigarrow_{A} & \llbracket \mathit{apply}(\gamma{:}\mathit{equal}(\gamma'{:}s(\alpha'{:}\#'(\beta_4, \gamma''{:}\mathit{cons}(\beta_3, \beta_4))), s(s(0))), \\
 & \epsilon) \\
 & \mid \beta_4 \not\approx \gamma'', \beta_3 \not\approx \alpha', \beta_4 \not\approx \alpha', \gamma'' \not\approx \alpha' \rrbracket \\
\rightsquigarrow_{T} & \llbracket \gamma{:}\mathit{equal}(\gamma'{:}s(\alpha'{:}\#'(\beta_4, \gamma''{:}\mathit{cons}(\beta_3, \beta_4))), s(s(0))) \\
 & \mid \beta_4 \not\approx \gamma'', \beta_3 \not\approx \alpha', \beta_4 \not\approx \alpha', \gamma'' \not\approx \alpha' \rrbracket \\
\rightsquigarrow_{\xi_1, A} & \llbracket \gamma{:}\mathit{equal}(\alpha'{:}\#'(\beta_4, \gamma''{:}\mathit{cons}(\beta_3, \beta_4)), s(0)) \\
 & \mid \beta_4 \not\approx \gamma'', \beta_3 \not\approx \alpha', \beta_4 \not\approx \alpha', \gamma'' \not\approx \alpha' \rrbracket \\
\rightsquigarrow_{\rho_2} & \llbracket \mathit{apply}(\gamma{:}\mathit{equal}(\alpha'{:}\#'(\beta_4, \gamma''{:}\mathit{cons}(\beta_3, \beta_4{:}\mathit{cons}(\beta'_2, \gamma''))), s(0)), \\
 & \alpha''^{+}, \alpha'{:}s(\alpha''), \alpha''{:}0)) \\
 & \mid \beta_4 \not\approx \gamma'', \beta_3 \not\approx \alpha', \beta_4 \not\approx \alpha', \gamma'' \not\approx \alpha' \rrbracket \\
\rightsquigarrow_{A^{+}} & \llbracket \mathit{apply}((\gamma{:}\mathit{equal}(\alpha'{:}\#'(\beta_4, \gamma''{:}\mathit{cons}(\beta_3, \beta_4{:}\mathit{cons}(\beta'_2, \gamma''))), s(0)), \alpha''), \\
 & \alpha'{:}s(\alpha''), \alpha''{:}0) \\
 & \mid \beta_4 \not\approx \gamma'', \beta_3 \not\approx \alpha', \beta_4 \not\approx \alpha', \gamma'' \not\approx \alpha', \gamma'' \not\approx \alpha'', \beta_3 \not\approx \alpha'', \beta_4 \not\approx \alpha'', \beta'_2 \not\approx \alpha'' \rrbracket \\
\rightsquigarrow_{A, A, T} & \llbracket \gamma{:}\mathit{equal}(\alpha'{:}s(\alpha''{:}0), s(0)), \gamma''{:}\mathit{cons}(\beta_3, \beta_4{:}\mathit{cons}(\beta'_2, \gamma'')) \\
 & \mid \beta_4 \not\approx \gamma'', \beta_3 \not\approx \alpha', \beta_4 \not\approx \alpha', \gamma'' \not\approx \alpha', \gamma'' \not\approx \alpha'', \beta_3 \not\approx \alpha'', \beta_4 \not\approx \alpha'', \beta'_2 \not\approx \alpha'' \rrbracket \\
\rightsquigarrow_{\xi_2, A, T} & \llbracket \gamma{:}\mathit{equal}(\alpha''{:}0, 0), \gamma''{:}\mathit{cons}(\beta_3, \beta_4{:}\mathit{cons}(\beta'_2, \gamma'')) \\
 & \mid \beta_4 \not\approx \gamma'', \beta_3 \not\approx \alpha', \beta_4 \not\approx \alpha', \gamma'' \not\approx \alpha', \gamma'' \not\approx \alpha'', \beta_3 \not\approx \alpha'', \beta_4 \not\approx \alpha'', \beta'_2 \not\approx \alpha'' \rrbracket \\
\rightsquigarrow_{\xi_1, A, T} & \llbracket \gamma{:}\mathit{true}, \gamma''{:}\mathit{cons}(\beta_3, \beta_4), \beta_4{:}\mathit{cons}(\beta'_2, \gamma'') \\
 & \mid \beta_4 \not\approx \gamma'', \beta_3 \not\approx \alpha', \beta_4 \not\approx \alpha', \gamma'' \not\approx \alpha', \gamma'' \not\approx \alpha'', \beta_3 \not\approx \alpha'', \beta_4 \not\approx \alpha'', \beta'_2 \not\approx \alpha'' \rrbracket
\end{array}
$$

6 The Properties of the Narrowing Relation

Soundness and Completeness are defined w.r.t. the ground rewriting rule introduced in Definition 3.

Soundness ensures that every narrowing derivation can be related to a sequence of rewriting steps, operating at the ground level. More precisely, if we have $\llbracket G \mid \top \rrbracket^{\emptyset} \rightsquigarrow_{\mathcal{R},\varsigma} \llbracket H \mid \phi \rrbracket^{\tau}$ then for any ground instance ϑ of H solution of ϕ, we should have $G\varsigma\vartheta \rightarrow^{*}_{\mathcal{R}} H\vartheta$. Unfortunately, this property does not hold for all substitutions ϑ. Indeed, if ϑ contains a node α on which a global redirection is performed during the narrowing derivation, then the term-graph obtained by rewriting from $G\varsigma\vartheta$ is different from $H\vartheta$, since any instance of α in ϑ should be redirected during the rewriting process. Thus we will assume that ϑ contains no such node (this property is always satisfied if we restrict ourself to irreducible derivations, see Definition 6). Similarly, ϑ should not contain any created node.

Theorem 1. *(Soundness) Let $\mathcal{R}$ be a set of rewrite rules. Let $\llbracket G \mid \psi \rrbracket^{\tau}$ and $\llbracket H \mid \psi' \rrbracket^{\tau'}$ be two g-terms s.t. $\llbracket G \mid \psi \rrbracket^{\tau} \rightsquigarrow^{*}_{\mathcal{R},\varsigma} \llbracket H \mid \psi' \rrbracket^{\tau'}$. Let ϑ be a ground g-substitution of H s.t. $\vartheta \in sol(\psi')$ and Gr_{ϑ} contains no node α s.t. $\alpha \in \mathcal{C}$ or $\alpha \gg \beta \in \tau'$, for some $\beta \in \mathcal{N}$. $\varsigma\vartheta$ is a g-substitution of G. Moreover value$(\llbracket G \mid \psi \rrbracket^{\tau}\varsigma\vartheta) \to^{*}_{\mathcal{R}}$ value$(\llbracket H \mid \psi \rrbracket^{\tau'}\vartheta)$.*

In particular, this implies that if G, H are two term-graphs s.t. $\llbracket G \mid \top \rrbracket^{\epsilon} \rightsquigarrow_{\mathcal{R},\varsigma} \llbracket H \mid \phi \rrbracket^{\tau}$ and ϑ is a g-substitution of H satisfying the above conditions, then $G\varsigma\vartheta \to^{*}_{\mathcal{R}} H\vartheta$.

Completeness expresses the fact that every rewriting derivation from a ground instance of a considered term-graph G can be subsumed by narrowing from $\llbracket G \mid \top \rrbracket^{\emptyset}$. More precisely, if ς is a g-substitution of G s.t. $G\varsigma \to^{*} H$ then there should exist a narrowing derivation $\llbracket G \mid \top \rrbracket^{\emptyset} \rightsquigarrow_{\mathcal{R},\varrho} \llbracket G' \mid \phi \rrbracket^{\tau}$ and a g-substitution $\vartheta \in sol(\phi)$ s.t. $G'\vartheta = H$ and $\varsigma = \varrho\vartheta$.

However, as for usual narrowing algorithms, this property does not hold for every substitution ς, but only for those that are irreducible w.r.t. the considered set of rewrite rules. The definition of an irreducible substitution is more complicated than in the usual case (i.e. for standard terms), since we have to take into account global redirections. Roughly speaking a term-graph G will be considered as reducible iff a rule can be applied on a node in $dom(G)$ (1) *or* if a rule can globally redirect a node in G (2) *or* if it contains a created node (3). More formally:

Definition 6. *A g-substitution $\varsigma = (\sigma, G)$ is said to be $\mathcal{R}$-*irreducible *iff the following conditions hold:*

1. *There is no rule $L \to_{\lambda} R \in \mathcal{R}$ s.t. there exists a substitution θ of the variables in L s.t. $L\theta \bowtie G$ and $\lambda\theta \in dom(G)$.*
2. *There is no rule $L \to_{\lambda} R \in \mathcal{R}$ s.t. there exists a substitution θ of the variables in L and a node $\alpha \in \mathcal{N}(L)$ s.t. $L\theta \bowtie G$, $\alpha\sigma \in \mathcal{N}(G)$, R contains an action of the form $\alpha \gg \beta$ for some $\beta \in \mathcal{N}$.*
3. *$\mathcal{N}(G) \cap \mathcal{C} = \emptyset$.*

If a constructor based signature is used, then Condition 1 simply states that ς contains no defined function. Similarly, Condition 2 can be easily guaranteed if only nodes labeled by defined functions can be globally redirected (this is a rather natural restriction). Condition 3 simply expresses the fact that ς should not contain any created node.

Theorem 2. *(Completeness) Let $\mathcal{R}$ be a set of rewrite rules. Let G, H be two term-graphs and let ς be an $\mathcal{R}$-irreducible ground substitution of the variables in G s.t. $G\varsigma \to^{*}_{\mathcal{R}} H$. For any ϕ s.t. $\varsigma \in sol(\phi)$ and for any sequence of actions τ s.t. $r(\tau) \subseteq dom(G)$, there exists ϱ s.t. $\llbracket G \mid \phi \rrbracket^{\tau} \rightsquigarrow^{*}_{\mathcal{R},\varrho} \llbracket G' \mid \psi \rrbracket \tau'$ and $\vartheta' \in sol(\psi)$ s.t. $\varsigma = \varrho\vartheta'$, G' is a term-graph, and value$(\llbracket G' \mid \psi \rrbracket \vartheta') = H$.*

In particular, if $G\varsigma \to^{*}_{\mathcal{R}} H$ and ς is $\mathcal{R}$-irreducible, then there exist ϱ s.t. $\llbracket G \mid \top \rrbracket^{\epsilon} \rightsquigarrow^{*}_{\mathcal{R},\varrho} \llbracket G' \mid \psi \rrbracket \tau'$ and $\vartheta \in sol(\psi)$ s.t. $\varsigma = \varrho\vartheta$, G' is a term-graph, and $G'\vartheta = H$.

7 Conclusion

We have shown that narrowing could be extended to a large class of term-graph rewrite systems. The considered rewrite rules allow one to fully handle data-structures with pointers thanks to the actions like pointer redirections, node redefinition and creation. These results are the first ones concerning the narrowing relation in such a wide class of term-graph rewrite systems. It is also the first narrowing-based algorithm able to synthesize cyclic data-structures as answers in a context where bisimilar graphs are considered as equal only if they are identical. In this paper we were rather interested in the basic definition of narrowing, its soundness and completeness. Optimal term-graph narrowing strategies as studied in [7, 8, 1] are out of the scope of this paper, but a matter of future work. The considered term-graph rewrite systems are not always confluent. We proposed in [6] the use of term-graphs with priority in order to recover the confluence property within orthogonal systems. The future narrowing strategies should certainly integrate the priority over the nodes of a term-graph in addition to neededness properties.

References

1. S. Antoy, D. W. Brown, and S.-H. Chiang. Lazy context cloning for non-deterministic graph rewriting. In *Third International Workshop on Term Graph Rewriting, TERMGRAPH*, pages 61–70, 2006.
2. S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. *Journal of the ACM*, 47(4):776–822, July 2000.
3. A. Bakewell, D. Plump, and C. Runciman. Checking the shape safety of pointer manipulations. In *RelMiCS*, pages 48–61, 2003.
4. A. Bakewell, D. Plump, and C. Runciman. Specifying pointer structures by graph reduction. In *AGTIVE*, pages 30–44, 2003.
5. H. Barendregt, M. van Eekelen, J. Glauert, R. Kenneway, M. J. Plasmeijer, and M. Sleep. Term Graph Rewriting. In *PARLE'87*, pages 141–158. Springer, LNCS 259, 1987.
6. R. Caferra, R. Echahed, and N. Peltier. Rewriting term-graphs with priority. In *Proceedings of PPDP (Principle and Practice of Declarative Programming)*. ACM, 2006.
7. R. Echahed and J.-C. Janodet. Admissible graph rewriting and narrowing. In *IJCSLP*, pages 325–342, 1998.
8. R. Echahed and J. C. Janodet. Completeness of admissible graph collapsing narrowing. In *Proc. of Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (GRATRA 2000)*, March 2000.
9. A. Habel and D. Plump. Term graph narrowing. *Mathematical Structures in Computer Science*, 6:649–676, 1996.
10. M. Hanus. The integration of functions into logic programming : from theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994.
11. M. K. Rao. Completeness results for basic narrowing in non-copying implementations. In *Proc. of Joint Int. Conference and Symposium on Logic Programming*, pages 393–407. MIT press, 1996.
12. H. Yamanaka. Graph narrowing and its simulation by graph reduction. Research report IIAS-RR-93-10E, Institute for Social Information Science, Fujitsu Laboratories LDT, June 1993.