Data-Structure Rewriting

Rachid Echahed, Dominique Duval, Frederic Prost

Data-Structure Rewriting

D. Duval[1], R. Echahed[2], F. Prost[2]
Institut d'Informatique et de Mathématiques Appliquées de Grenoble
{Dominique.Duval | Rachid.Echahed | Frederic.Prost}@imag.fr
[1] Laboratoire LMC, B. P. 53, 38041 Grenoble, France
[2] Laboratoire LEIBNIZ, 46, av. Felix Viallet, 38031 Grenoble, France

March 24, 2005

Abstract

We tackle the problem of data-structure rewriting including pointer redirections. We propose two basic rewrite steps: (i) Local Redirection and Replacement steps the aim of which is redirecting specific pointers determined by means of a pattern, as well as adding new information to an existing data ; and (ii) Global Redirection steps which are aimed to redirect all pointers targeting a node towards another one. We define these two rewriting steps following the double pushout approach. We define first the category of graphs we consider and then define rewrite rules as pairs of graph homomorphisms of the form $L \leftarrow K \rightarrow R$. Unfortunately, inverse pushouts (complement pushouts) are not unique in our setting and pushouts do not always exist. Therefore, we define rewriting steps so that a rewrite rule can always be performed once a matching is found.

1 Introduction

Rewriting techniques have been proven to be very useful to establish formal bases for high level programming laguages as well as theorem provers. These techniques have been widely investigated for strings [7], trees or terms [2] and term graphs [19, 6].

In this paper we tackle the problem of rewriting classical data-structures such as circular lists, double-chained lists, etc. Even if such data-structures can be easily simulated by string or tree processing, they remain very useful in designing algorithms with good complexity. The investigation of data-structure rewrite systems will contribute to define a clean semantics and proof techniques for "pointer" handling. It will also provide a basis for multiparadigm programming languages integrating declarative (functional and logic) and imperative features.

General frameworks of graph transformation are now well established, see e.g. [22, 11, 12]. Unfortunately, rewriting classical data-structures represented as cyclic graphs did not benefit yet of the same effort as for terms or term

graphs. Our aim in this paper is to investigate basic rewrite steps for data-structure transformation. It turns out that pointer redirection is the key issue we had to face, in addition to classical replacement and garbage collection. We distinguish two kinds of redirections: (i) *Global redirection* which consists in redirecting in a row all edges pointing to a given node, to another node ; and (ii) *Local redirection* which consists in redirecting a particular pointer, specified e.g. by a pattern, in order to point to a new target node. Global redirection is very often used in the implementation of functional programming languages, for instance when changing roots of term graphs. As for local redirection, it is useful to express classical imperative algorithms.

We introduce two kind of rewrite steps. The first is one called *local redirection and replacement* and the second kind is dedicated to global redirection. We define these steps following the double pushout approach [8, 16]. We have chosen this approach because it simplifies drastically the presentation of our results. The algorithmic fashion, which we followed first, turns out to be arduous. Thus, basic rewrite rules are given by a pair of graph homomorphisms $L \leftarrow K \rightarrow R$. We precise the rôle that plays K in order to perform local or global redirection of pointers. The considered homomorphisms are not necessarily injective in our setting, unlike classical assumptions as in the recent proposals dedicated to graph programs [20, 17]. This means that inverse pushouts (complement pushouts) are not unique.

The paper is organized as follows: The next section introduces the category of graphs which we consider in the paper. Section 3 states some technical results that help defining rewrite steps. Section 4 introduces data-structure rewriting and defines mainly two rewrite steps, namely LRR-rewriting and GR-rewriting. We compare our proposal to related work in section 5. Concluding remarks are given in section 6. Proofs are found in the appendix. We assume the reader is familiar with basic notions of category theory (see e.g. [1] for an introduction).

2 Graphs

In this section we introduce the category of graphs we consider in the paper. These graphs are supposed to represent data-structures. We define below such graphs in a mono-sorted setting. Lifting our results to the many-sorted case is straightforward.

Definition 2.1 (Signature) A *signature* Ω is a set of operation symbols such that each operation symbol in Ω, say f, is provided by a natural number, n, representing its *arity*. We write $\mathrm{ar}(f) = n$.

In the sequel, we use the following notations. Let A be a set. We note A^* the set of strings made of elements in A. Let $f : A \rightarrow B$ be a function. We note $f^* : A^* \rightarrow B^*$ the unique extension of f over strings defined by $f^*(\epsilon) = \epsilon$ where ϵ is the empty string and $f^*(a_1 \dots a_n) = f(a_1) \dots f(a_n)$.

We assume that Ω is fixed throughout the rest of the paper.

Definition 2.2 (Graph) A *graph* G is made of:

- a set of *nodes* $\mathcal{N}_G$,
- a subset of *labeled nodes* $\mathcal{N}_G^\Omega \subseteq \mathcal{N}_G$,
- a *labeling function* $\mathcal{L}_G : \mathcal{N}_G^\Omega \to \Omega$,
- and a *successor function* $\mathcal{S}_G : \mathcal{N}_G^\Omega \to \mathcal{N}_G^*$,

such that, for each labeled node n, the length of the string $\mathcal{S}_G(n)$ is the arity of the operation $\mathcal{L}_G(n)$.

This definition can be illustrated by the following diagram, where $\lg(u)$ is the length of the string u. :

$$\begin{array}{ccccc} \mathcal{N}_G & \xleftarrow{\supseteq} & \mathcal{N}_G^\Omega & \xrightarrow{\mathcal{S}_G} & \mathcal{N}_G^* \\ & & \downarrow{\scriptstyle \mathcal{L}_G} & = & \downarrow{\scriptstyle \lg} \\ & & \Omega & \xrightarrow[\text{ar}]{} & \mathbb{N} \end{array}$$

Moreover:

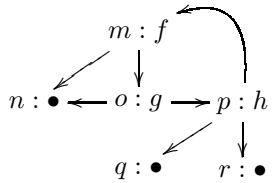
- the *arity* of a node n is defined as the arity of its label,
- the i-th successor of a node n is denoted $succ_G(n, i)$,
- the *edges* of a graph G are the pairs (n, i) where $n \in \mathcal{N}_G^\Omega$ and $i \in \{1, \ldots, \text{ar}(n)\}$, the *source* of an edge (n, i) is the node n, and its *target* is the node $succ_G(n, i)$,
- the fact that $f = \mathcal{L}_G(n)$ can be written as $n\!:\!f$,
- the set of unlabeled nodes of G is denoted $\mathcal{N}_G^\mathcal{X}$, so that: $\mathcal{N}_G = \mathcal{N}_G^\Omega +^1 \mathcal{N}_G^\mathcal{X}$.

Example 2.3 Let G be the graph defined by

- $\mathcal{N}_G = \{m; n; o; p; q; r\}$
- $\mathcal{N}_G^\Omega = \{m; o; p\}$
- $\mathcal{N}_G^\mathcal{X} = \{n; q; r\}$
- $\mathcal{L}_G$ is defined by: $[m \mapsto f; o \mapsto g; p \mapsto h]$
- $\mathcal{S}_G$ is defined by: $[m \mapsto no; o \mapsto np; p \mapsto qrm]$

[1] + stands for disjoint union.

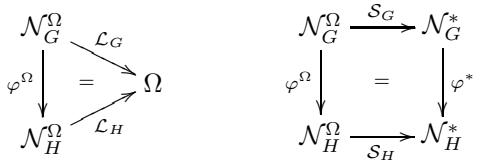
Graphically we represent this graph as:



We use • to denote lack of label. Informally, one may think of • as anonymous variables.

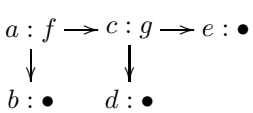
Definition 2.4 (Graph homomorphism) A *graph homomorphism* $\varphi : G \to H$ is a map $\varphi : \mathcal{N}_G \to \mathcal{N}_H$ such that $\varphi(\mathcal{N}_G^\Omega)$ is included in $\mathcal{N}_H^\Omega$ and, for each node $n \in \mathcal{N}_G^\Omega$: $\mathcal{L}_H(\varphi(n)) = \mathcal{L}_G(n)$ and $\mathcal{S}_H(\varphi(n)) = \varphi^*(\mathcal{S}_G(n))$.

Let $\varphi^\Omega : \mathcal{N}_G^\Omega \to \mathcal{N}_H^\Omega$ denote the restriction of φ to the subset $\mathcal{N}_G^\Omega$. Then, the properties in the definition above mean that the following diagrams are commutative:



The image $\varphi(n, i)$ of an edge (n, i) of G is defined as the edge $(\varphi(n), i)$ of H.

Example 2.5 Consider the following graph H:



Let $\varphi : \mathcal{N}_H \to \mathcal{N}_G$, where G is the graph defined in Example 2.3, be defined as: $[a \mapsto m; b \mapsto n; c \mapsto o; d \mapsto n; e \mapsto p]$. Map φ is a graph homomorphism from H to G. Notice that the nodes without labels act as placeholders for any graph.

It is easy to check that the graphs (as objects) together with the graph homomorphisms (as arrows) form a category, which is called the *category of graphs* and noted **Gr** .

3 Disconnected graphs and homomorphisms

This section is dedicated to some technical definitions the aim of which is the simplification of the definition of rewrite rules given in the following section.

Definition 3.1 (Disconnected edge) An edge (n, i) of a graph G is *disconnected* if its target $succ_G(n, i)$ is unlabeled.

The next definition introduces the notion of what we call disconnected graph. Roughly speaking, the disconnected graph associated to a graph G and a set of edges E is obtained by redirecting every edge in E (whether it is yet disconnected or not) towards a *new*, unlabeled, target.

Definition 3.2 (Disconnected graph) The *disconnected graph* associated to a graph G and a set of edges E of G is the following graph $D(G, E)$:

- $\mathcal{N}_{D(G,E)} = \mathcal{N}_G + \mathcal{N}_E$, where $\mathcal{N}_E$ is made of one new node $n[i]$ for each edge $(n, i) \in E$,
- $\mathcal{N}^{\Omega}_{D(G,E)} = \mathcal{N}^{\Omega}_G$,
- for each $n \in \mathcal{N}^{\Omega}_G$: $\mathcal{L}_{D(G,E)}(n) = \mathcal{L}_G(n)$,
- for each $n \in \mathcal{N}^{\Omega}_G$ and $i \in \{1, \ldots, \mathrm{ar}(n)\}$:
  - if $(n, i) \notin E$ then $succ_{D(G,E)}(n, i) = succ_G(n, i)$,
  - if $(n, i) \in E$ then $succ_{D(G,E)}(n, i) = n[i]$.

Definition 3.3 (Connection homomorphism) The *connection homomorphism* associated to a graph G and a set of edges E of G is the homomorphism $\delta_{G,E} : D(G, E) \to G$ such that:

- if $n \in \mathcal{N}_G$ then $\delta_{G,E}(n) = n$,
- if $n[i] \in \mathcal{N}_E$ then $\delta_{G,E}(n[i]) = succ_G(n, i)$.

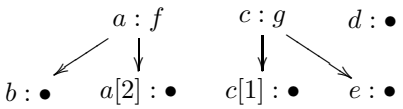
It is easy to check that $\delta_{G,E}$ is a graph homomorphism.

Definition 3.4 (Disconnected homomorphism) The *disconnected graph homomorphism* associated to a graph homomorphism $\varphi : G \to H$ and a set of edges E of G is the homomorphism $D_{\varphi,E} : D(G, E) \to D(H, \varphi(E))$ defined as follows:

- if $n \in \mathcal{N}_G$ then $D_{\varphi,E}(n) = \varphi(n)$,
- if $n[i] \in \mathcal{N}_E$ then $D_{\varphi,E}(n[i]) = \varphi(n)[i]$.

It is easy to check that $D_{\varphi,E}$ is a graph homomorphism.

Example 3.5 Consider the graph H of Example 2.5. Then the disconnected graph associated to H and the set of edges $\{(a, 2); (c, 1)\}$ is the following graph:



Note that even if edge $(c, 1)$ is already disconnected in H it is *redirected* towards a new unlabeled node,$c[1]$, in $D(H, \{(a, 2); (c, 1)\})$.

Now if we consider the graph homomorphism $\varphi : H \to G$ defined in Example 2.5, the disconnected graph homomorphism $D_{\varphi,\{(a,2);(c,1)\}} : D(H, \{(a, 2); (c, 1)\}) \to D(G, \{(m, 2); (o, 1)\})$ is the mapping $[a \mapsto m; b \mapsto n; c \mapsto o; d \mapsto n; e \mapsto p; a[2] \mapsto m[2]; c[1] \mapsto o[1]]$

4 Data-structure rewriting

In this section we define data structure rewriting as a succession of rewrite steps. A rewrite step is defined from a rewrite rule and a matching. A rewrite rule is a *span* of graphs, i.e., a pair of graph homomorphisms with a common source:

$$L \xleftarrow{\delta} K \xrightarrow{\rho} R$$

A matching is a morphism of graphs: $L \xrightarrow{\mu} G$. There are two kinds of rewrite steps.

- The first kind is called *Local Redirection and Replacement Rewriting* (LRR-rewriting, for short). Its rôle is twofold: adding to G a copy of the instance of the right-hand side R, and performing some local redirections of edges specified by means of the rewrite rule.
- The second kind of rewrite steps is called *Global Redirection Rewriting* (GR-Rewriting, for short). Its rôle consists in performing redirections: all incoming edges of some node a in G are redirected to a node b.

We define LRR-rewriting and GR-rewriting in the two following subsections. We use in both cases the double-pushout approach to define rewrite steps.

4.1 LRR-rewriting

Before defining LRR-rewrite rules and steps, we state first a technical result about the existence of inverse pushouts in our setting.

Theorem 4.1 (An inverse pushout) *Let $\mu : L \to U$ be a graph homomorphism, E a set of edges of L, and let $D_{\mu,E} : D(L,E) \to D(U,\mu(E))$ be the disconnected graph homomorphism associated to μ and E. Then the following square is a pushout in the category of graphs (***Gr***):*

$$\begin{array}{ccc} L & \xleftarrow{\delta_{L,E}} & D(L,E) \\ {\scriptstyle\mu}\downarrow & & \downarrow{\scriptstyle D_{\mu,E}} \\ U & \xleftarrow[\delta_{U,\mu(E)}]{} & D(U,\mu(E)) \end{array}$$

Proof. This result is an easy corollary of Theorem A.2. □

Definition 4.2 (Disconnecting pushout) Let $\mu : L \to U$ be a graph homomorphism and E a set of edges of L. The *disconnecting pushout associated to μ and E* is the pushout from Theorem 4.1.

It can be noted that the disconnecting pushout is not unique, in the sense that there are generally several inverse pushouts of:

$$\begin{array}{ccc} L & \xleftarrow{\delta_{L,E}} & D(L,E) \\ {\scriptstyle\mu}\downarrow & & \\ U & & \end{array}$$

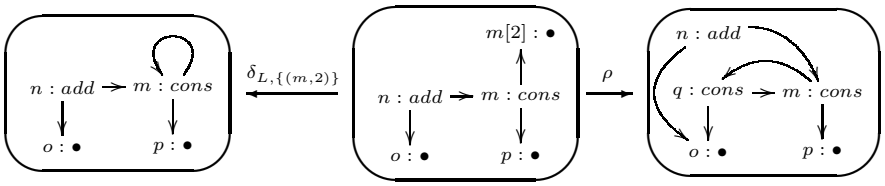
Before stating the next definition, it should be reminded that $\mathcal{N}_{D(L,E)} = \mathcal{N}_L + \mathcal{N}_E = \mathcal{N}_L^\Omega + \mathcal{N}_L^\mathcal{X} + \mathcal{N}_E$.

Definition 4.3 (LRR-rewrite rule) A *Local Redirection and Replacement Rewrite rule* (or a *LRR-rewrite rule*, for short) is a span of graph homomorphisms of the form:

$$L \xleftarrow{\delta_{L,E}} D(L,E) \xrightarrow{\rho} R$$

where E is a set of edges of L, and where $\rho(\mathcal{N}_L^\mathcal{X}) \subseteq \mathcal{N}_R^\mathcal{X}$ and the restriction of ρ to $\mathcal{N}_L^\mathcal{X}$ is injective.

Example 4.4 Consider the function *add* which adds an element to a *circular* list. The span below defines a rewrite rule defining the function *add* in the case where the circular list consists of one element (the case of lists of length greater than one is given in Example 4.10).



L D(L,{(m,2)}) R

In this example we show how (local) edge redirection can be achieved through edge disconnection. Since an element is added to the head of a circular list (of length 1), one has to make the curve pointer $(m, 2)$ to point to the new added cell. For this we disconnect the edge $(m, 2)$ in $D(L, \{(m, 2)\})$ in order to be able to redirect it, thanks to an appropriate homomorphism ρ, to the new cell in R, namely q. Here, $\rho = [n \mapsto n; m[2] \mapsto q; \cdots]$

One may also remark that graph R still has a node labelled by *add*. In this paper we do not tackle the problem of garbage collection which has been treated in a categorical way in e.g. [4].

Definition 4.5 (LRR-matching) A *LRR-matching* with respect to a LRR-rewrite rule $L \xleftarrow{\delta_{L,E}} D(L,E) \xrightarrow{\rho} R$ is a graph homomorphism $\mu : L \to U$ that is Ω-*injective*, which means that the restriction of the map μ to $\mathcal{N}_G^\Omega$ is injective.

Definition 4.6 (LRR-Rewrite step) Let $r = (\ L \xleftarrow{\delta_{L,E}} D(L,E) \xrightarrow{\rho} R\ )$ be a rewrite rule, and $\mu : L \to U$ a matching with respect to r. Then U *rewrites into* V *using rule* r if there are graph homomorphisms $\nu : R \to V$ and $\rho' : D(U, \mu(E)) \to V$ such that the following square is a pushout in the category of graphs (**Gr**):

$$\begin{array}{ccc} D(L,E) & \xrightarrow{\rho} & R \\ {\scriptstyle D_{\mu,E}}\downarrow & & \downarrow{\scriptstyle \nu} \\ D(U,\mu(E)) & \xrightarrow[\rho']{} & V \end{array}$$

Thus, a rewrite step corresponds to a *double pushout* in the category of graphs:

$$\begin{array}{ccccc} L & \xleftarrow{\delta_{L,E}} & D(L,E) & \xrightarrow{\rho} & R \\ {\scriptstyle \mu}\downarrow & & {\scriptstyle D_{\mu,E}}\downarrow & & \downarrow{\scriptstyle \nu} \\ U & \xleftarrow[\delta_{U,\mu(E)}]{} & D(U,\mu(E)) & \xrightarrow[\rho']{} & V \end{array}$$

Theorem 4.7 (Rewrite step is feasible) *Let r be a rewrite rule, and $\mu : L \to U$ a matching with respect to r. Then U can be rewritten using rule r. More precisely, the required pushout can be built as follows (the notations are simplified by dropping E and $\mu(E)$):*

- *the set of nodes of V is $\mathcal{N}_V = (\mathcal{N}_R + \mathcal{N}_{D(U)})/\sim$, where $\sim$ is the equivalence relation generated by $D_\mu(n) \sim \rho(n)$ for each node n of $D(L)$,*
- *the maps ν and ρ', on the sets of nodes, are the inclusions of $\mathcal{N}_R$ and $\mathcal{N}_{D(U)}$ in $\mathcal{N}_R + \mathcal{N}_{D(U)}$, respectively, followed by the quotient map with respect to $\sim$,*
- *$\mathcal{N}_V^\Omega$ is made of the classes modulo $\sim$ which contain at least one labeled node, and a section $\pi : \mathcal{N}_V^\Omega \to \mathcal{N}_R^\Omega + \mathcal{N}_{D(U)}^\Omega$ of the quotient map is chosen, which means that the class of $\pi(n)$ is n, for each $n \in \mathcal{N}_V^\Omega$,*
- *for each $n \in \mathcal{N}_V^\Omega$, the label of n is the label of $\pi(n)$,*
- *for each $n \in \mathcal{N}_V^\Omega$, the successors of n are the classes of the successors of $\pi(n)$.*

Moreover, the resulting pushout does not depend on the choice of the section π.

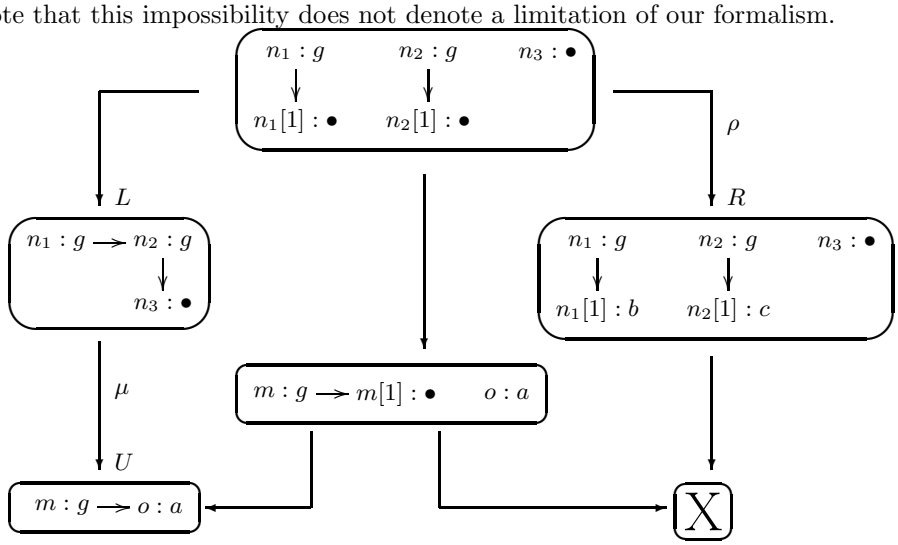
Corollary 4.8 (A description of the labeled nodes) *With the notations and assumptions of Theorem 4.7, the representatives of the equivalence classes of nodes of $\mathcal{N}_R + \mathcal{N}_{D(U)}$ can be chosen in such a way that:*

$$\mathcal{N}_V^\Omega = (\mathcal{N}_U^\Omega - \mu(\mathcal{N}_L^\Omega)) + \mathcal{N}_R^\Omega\ .$$

Proof. Both Theorem 4.7 and Corollary 4.8 are derived from Theorem A.4, their proofs are given at the end of the appendix. □

Example 4.9 Here we consider the case of a non Ω-injective matching in order to show that there may be no double pushout in such cases. Thus justifying our restriction over acceptable matchings (see Definition 4.5).

In this example we identify two nodes of L labelled by g via the homomorphism μ, namely n_1 and n_2, to a single one, m. In the span we disconnect the two edges coming from g's and redirect them to two different nodes labeled by different constants : b and c.This is done by the homomorphism $\rho = id$. Now, as both edges have been merged by the matching in U, the second (right) pushout cannot exist since a single edge cannot point to both b and c in the same time. Note that this impossibility does not denote a limitation of our formalism.



Example 4.10 In this example we complete the definition of the addition of an element to a circular list started in Example 4.4 where we gave a span for the case of list of size 1. In Figure 1 we give the span for lists of size greater than 1, as well as the application of the rule to a list of size 3.

Notice how the disconnection is actually used in order to redirect the pointer $(n_6, 2)$. The homomorphisms of the bottom layer show that the disconnected edge, pointing to the unlabeled node $c_4[2]$ is mapped to c_1 to the left and to n_8 to the right. The mechanism of disconnection allows the categorical manipulation of an edge.

The Ω-injectivity hypothesis is also useful in this rule since edges $(n_6, 2)$ and $(n_3, 2)$ must be different, thus a list of size less than or equal to one cannot be matched by this rule.

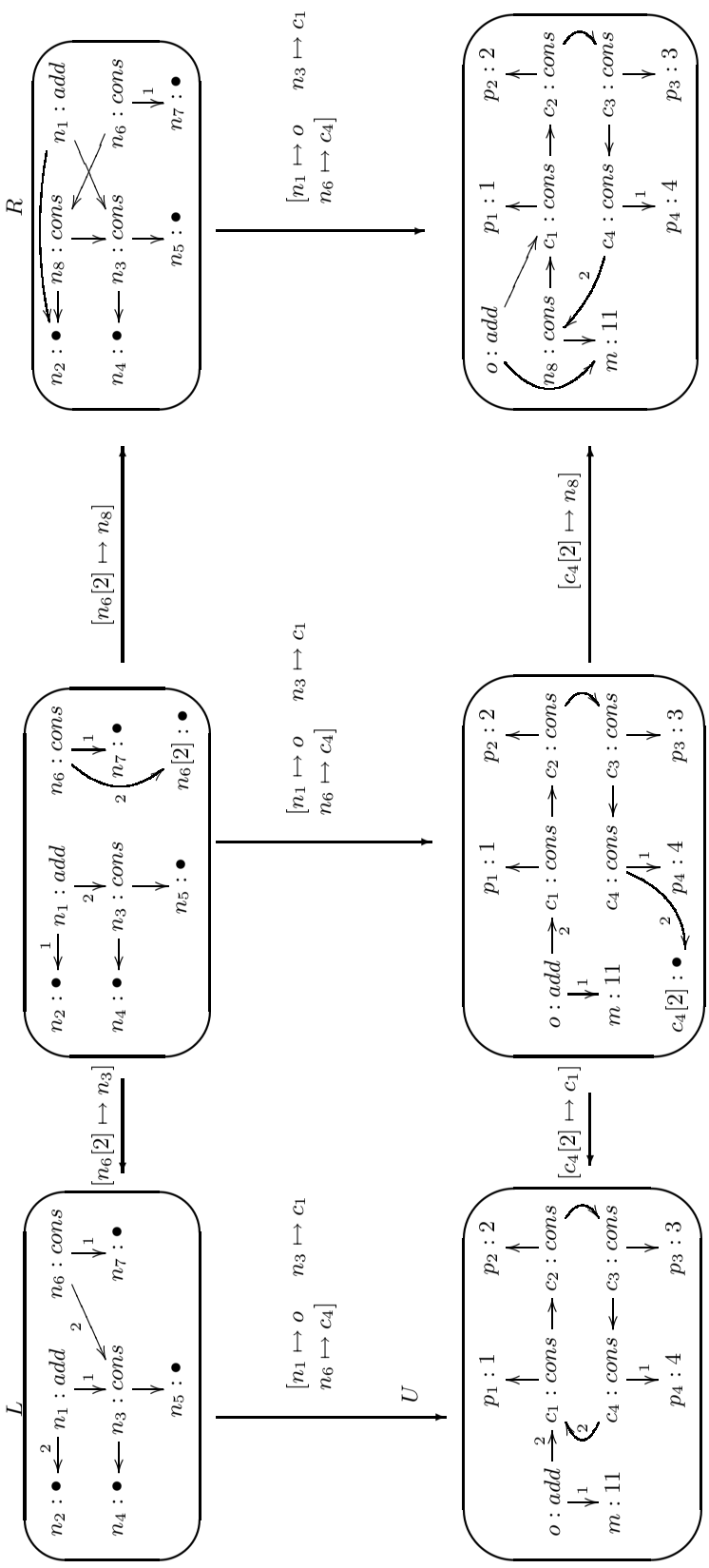


Figure 1: LRR-rewrite step defining "add" function on circular lists of size greater than one

4.2 GR-Rewriting

Let U be graph and let $a, b \in \mathcal{N}_U$. we say that U rewrites into V using the global redirection from a to b and write $U \xrightarrow{a \to b} V$ iff V is obtained from U by redirecting all edges targeting node a to point towards node b. This kind of rewriting is very useful when dealing with rooted term graphs (see, e.g. [4]). We define below one GR-rewriting step following the double pushout approach.

Definition 4.11 (GR-rewrite rule) A *Global Redirection rewrite rule* (or a *GR-rewrite rule*, for short) is a span of graph homomorphisms of the form:

$$P \xleftarrow{\lambda} SW \xrightarrow{\rho} P$$

where

- P is made of two unlabeled nodes ar and pr,
- SW (switch graph) is made of three unlabeled nodes ar, pr and mr,
- $\lambda(ar) = \lambda(mr) = ar$ and $\lambda(pr) = pr$,
- $\rho(ar) = ar$ and $\rho(pr) = \rho(mr) = pr$.

Definition 4.12 (GR-matching) A *GR-matching* with respect to a GR-rewrite rule $P \xleftarrow{\lambda} SW \xrightarrow{\rho} P$ is a graph homomorphism $\mu : P \to U$.

In order to define one GR-rewrite step, $U \xrightarrow{a \to b} V$, we need first somme technical definitions and properties we give below.

Definition 4.13 (Disconnected graph w.r.t. a node) Let G be a graph and o a node of G. Let mr denote a node which is not in $\mathcal{N}_G$. The *disconnected graph* associated to G and o is the following graph $\bar{D}(G, o)$:

- $\mathcal{N}_{\bar{D}(G,o)} = \mathcal{N}_G + \{mr\}$,
- $\mathcal{N}^{\Omega}_{\bar{D}(G,o)} = \mathcal{N}^{\Omega}_G$,
- $\forall n \in \mathcal{N}^{\Omega}_G$, $\mathcal{L}_{\bar{D}(G,o)}(n) = \mathcal{L}_G(n)$,
- $\forall n \in \mathcal{N}^{\Omega}_G$, $\forall i \in \{1, \ldots, \mathrm{ar}(n)\}$, $succ_G(n, i) = o \Rightarrow succ_{\bar{D}(G,o)}(n, i) = mr$
- $\forall n \in \mathcal{N}^{\Omega}_G$, $\forall i \in \{1, \ldots, \mathrm{ar}(n)\}$, $succ_G(n, i) \neq o \Rightarrow succ_{\bar{D}(G,o)}(n, i) = succ_G(n, i)$

Informally, $\bar{D}(G, o)$ is obtained from the graph G after redirecting all incoming edges of node o to point to the new unlabeled node mr.

Proposition 4.14 (Inverse pushout) *Let U be a graph, $P \xleftarrow{\lambda} SW \xrightarrow{\rho} P$ be a GR-rewrite rule, and $\mu : P \to U$ a GR-matching. Let $\bar{D}_\mu : SW \to \bar{D}(U, \mu(ar))$ be the homomorphism defined by $\bar{D}_\mu(ar) = \mu(ar)$, $\bar{D}_\mu(pr) = \mu(pr)$ and $\bar{D}_\mu(mr) = mr$. Let $\delta_\mu : \bar{D}(U, \mu(ar)) \to U$ be the homomorphism defined by $\delta_\mu(n) = n$ if $n \neq mr$ and $\delta_\mu(mr) = \mu(ar)$. Then the following square is a pushout in the category of graphs (**Gr**):*

$$\begin{array}{ccc} P & \xleftarrow{\lambda} & SW \\ {\scriptstyle\mu}\downarrow & & \downarrow{\scriptstyle\bar{D}_\mu} \\ U & \xleftarrow[\delta_\mu]{} & \bar{D}(U, \mu(ar)) \end{array}$$

Proof. This result is a direct consequence of Theorem A.2. □

Definition 4.15 (GR-rewrite step) Let U be a graph, $r = P \xleftarrow{\lambda} SW \xrightarrow{\rho} P$ be a GR-rewrite rule, and $\mu : P \to U$ be a GR-matching. Let $\bar{D}_\mu : SW \to \bar{D}(U, \mu(ar))$ be the homomorphism defined by $\bar{D}_\mu(ar) = \mu(ar)$, $\bar{D}_\mu(pr) = \mu(pr)$ and $\bar{D}_\mu(mr) = mr$. Then U *rewrites into* V *using rule* r if there are graph homomorphisms $\nu : P \to V$ and $\rho' : \bar{D}(U, \mu(ar)) \to V$ such that the following square is a pushout in the category of graphs (**Gr**):

$$\begin{array}{ccc} SW & \xrightarrow{\rho} & P \\ {\scriptstyle\bar{D}_\mu}\downarrow & & \downarrow{\scriptstyle\nu} \\ \bar{D}(U, \mu(ar)) & \xrightarrow[\rho']{} & V \end{array}$$

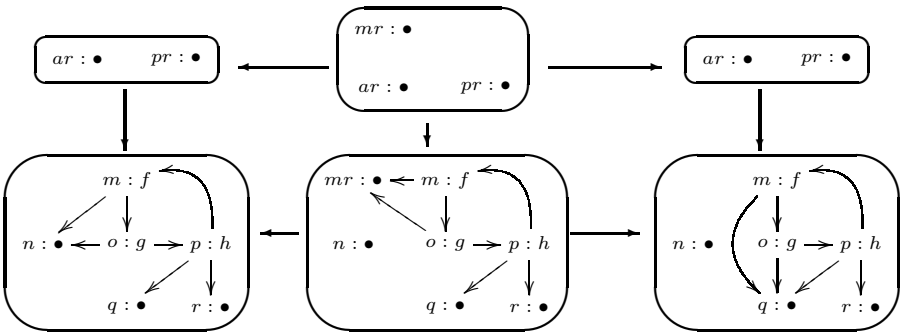
Thus, a GR-rewrite step, $U \overset{\mu(ar)\to\mu(pr)}{\longrightarrow} V$, corresponds to a *double pushout* in the category of graphs:

$$\begin{array}{ccccc} P & \xleftarrow{\delta_\mu} & SW & \xrightarrow{\rho} & P \\ {\scriptstyle\mu}\downarrow & & \downarrow{\scriptstyle\bar{D}_\mu} & & \downarrow{\scriptstyle\nu} \\ U & \xleftarrow[\delta_\mu]{} & D(U, \mu(ar)) & \xrightarrow[\rho']{} & V \end{array}$$

The construction of graph V is straightforward. It may be deduced from Theorem A.4 given in the appendix.

Example 4.16 In this example we show how global redirection works. In the graph G, given in Example 2.3, we want redirect all edges with target n towards q. For this pupose, we define the homomorphism μ from P to G by mapping appropriately the nodes ar (*ante-rewriting*), and pr (*post-rewriting*). I.e. in our

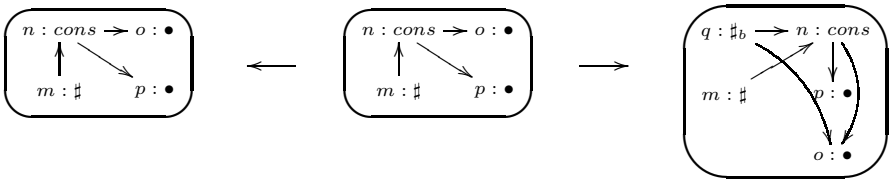
case $\mu = [ar \mapsto n; pr \mapsto q]$. Applying this on G, we get the following double push-out:



Notice how node mr (*midrewriting*) is used. It is mapped to n on the left and to q on the right. Thus in the middle graph, mr allows to disconnect edges targeting n in order to redirect them towards q.

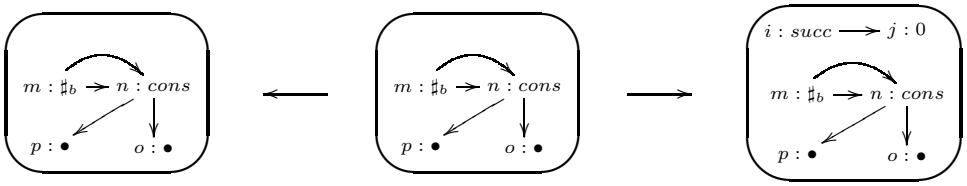
Example 4.17 In this additional example, we give rewriting rules defining the function length (written $\sharp$) which computes the size of non-empty circular lists. In this example every LRR-rewriting is followed by a GR-rewriting. That is why we precise the global rewriting that should be performed after each LRR-rewrite step.

The first rule simply introduces an auxiliary function, $\sharp_b$, which has two arguments. The first one indicates the head of the list while the second one will move along the list in order to measure it. We have the following span for $\sharp$:



together with the pair (m, q) for the global redirection.

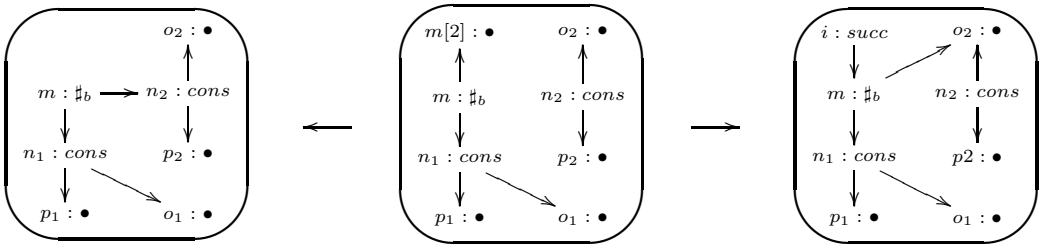
Now we have two rules for $\sharp_b$. The first one considers the case where the two arguments of $\sharp_b$ are the same ; and thus the length of the list equals one ($succ(0)$). Thus we have the following span:



together with the pair (m, i) for the global redirection. Notice that in this particular case we simply drop the input and replace it by a new graph as in

classical term rewrite systems, before performing the global redirection induced by the pair (m, i).

The next rule defines $\sharp_b$ when its arguments are different. Once again we use the hypothesis of Ω-injectivity to ensure that both *cons* nodes cannot be identified via matching.



together with the pair (m, i) for the global redirection. We let the reader check that circular lists of size n actually reduce to $\overbrace{succ(succ \ldots}^{n}(0))$ by successive application of rewriting rules (LRR and GR rewrite steps).

5 Related Work

Term graph rewriting [5, 19, 6] have been mainly motivated by implementation issues of functional programming languages. These motivations impact clearly their definition.

In [15, 9] jungles, a representation of acyclic term graphs by means of hypergraphs, have been investigated. We share with these proposals the use of the double-pushout approach of rewriting. However, we are rather interested in cyclic graphs.

In [5, 18, 10] cyclic term graph rewriting is considered using the algorithmic way. Pointer redirection is limited to global redirection of all edges pointing to the root of a redex by redirecting them to point to the root of the instance of the right-hand side. In [4], Banach, inspired by features found in implementations of declarative languages, proposed rewrite systems close to ours. We share the same graphs and global redirection of pointers. However, Banach did not discuss local redirections of pointers. We differ also in the way to express rewriting. Rewriting steps in [4] are defined by using the notion of opfibration of a category while our approach is based on double-pushouts.

The difference between our proposal to generalize term graph rewriting and previous works comes from the motivation. Our aim is not the implementation of declarative programming languages. It is rather the investigation of the elementary transformation rules of data-structures as occur in classical algorithms. In such structures pointers play a key rôle that we tried to take into account by proposing for instance redirections of specific edges within rewrite rules.

In [17], Habel and Plump proposed a kernel language for graph transformation. This language has been improved recently in [20]. Basic rules in this

framework are of the form $L \leftarrow K \rightarrow R$ satisfying some conditions such as the inclusion $K \subseteq L$. Unfortunately, our rewrite rules do not fulfill such condition ; particularly when performing local edge redirections. Furthermore, inverse pushouts (or pushout complements) are not unique in our setting which is not the case in [17, 20].

Recently, in [3] the authors are also interested in classical data-structures built by using pointers. Their work is complementary to ours in the sense that they are rather concerned by *recognizing* data-structure shapes by means of so called Graph reduction specifications.

Last, but not least, there are yet some programming languages which provide graph transformation features (see, e.g. [23, 13, 14, 21]). Our purpose in this paper is to focus on formal definition of basic data-structure transformation steps rather than building an entire programming language with suitable visual syntax and appropriate evaluation strategies.

6 Conclusion

We defined two basic rewrite steps dedicated to data-structure rewriting. The rewrite relationships induced by LRR-rewrite rules as well as GR-rewrite rules over graphs are trickier than the classical ones over terms (trees). There was no room in the present paper to discuss classical properties of the rewrite relationship induced by the above definitions such as confluence and termination or its extension to narrowing. However, our preliminary investigation shows that confluence is not guaranted even for nonoverlapping rewrite systems, and thus user-definable strategies are necessary when using all the power of data-structure rewriting. In addition, integration of LRR and GR rewriting in one step is also possible and can be helpful in describing some algorithms.

On the other hand, data-structures are better represented by means of graphics (e.g. [21]). Our purpose in this paper was rather the definition of the basic rewrite steps for data-structures. We intend to consider syntactical issue in a future work.

References

[1] A. Asperti and G. Longo. *Categories, Types and Structures. An introduction to Category Theory for the working computer scientist.* M.I.T. Press, 1991. http://www.di.ens.fr/users/longo/download.html.

[2] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[3] A. Bakewell, D. Plump, and C. Runciman. Checking the shape safety of pointer manipulations. In *International Seminar on Relational Methods in Computer Science (RelMiCS 7), Revised Selected Papers, Lecture Notes in Computer Science 3051,Springer-Verlag*, pages 48–61, 2004.

[4] R. Banach. Term graph rewriting and garbage collection using opfibrations. *Theoretical Computer Science*, 131:29–94, 1994.

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

[6] E. Barendsen and S. Smetsers. Graph rewriting aspects of functional programming. In H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 63–102. World Scientific, 1999.

[7] R. V. Book and F. Otto. *String-rewriting systems*. Springer-Verlag, 1993.

[8] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation - part I: Basic concepts and double pushout approach. In *Handbook of Graph Grammars*, pages 163–246, 1997.

[9] A. Corradini and F. Rossi. Hyperedge replacement jungle rewriting for term-rewriting systems and programming. *Theor. Comput. Sci.*, 109(1&2):7–48, 1993.

[10] R. Echahed and J. C. Janodet. Admissible graph rewriting and narrowing. In *Proc. of Joint International Conference and Symposium on Logic Programming (JICSLP'98)*, pages 325–340. MIT Press, June 1998.

[11] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2: Applications, Languages and Tools*. World Scientific, 1999.

[12] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism and Distribution*. World Scientific, 1999.

[13] C. Ermel, M. Rudolf, and G. Taentzer. The AGG approach: language and environment. In *Handbook of graph grammars and computing by graph transformation: vol. 2: applications, languages, and tools*, pages 551–603. World Scientific Publishing Co., Inc., 1999.

[14] J. R. W. Glauert, R. Kennaway, and M. R. Sleep. Dactl: An experimental graph rewriting language. In *Graph-Grammars and Their Application to Computer Science, LNCS 532*, pages 378–395, 1990.

[15] A. Habel, H. J. Kreowski, and D. Plump. Jungle evaluation. *Fundamenta Informaticae*, 15(1):37–60, 1991.

[16] A. Habel, J. Muller, and D. Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 11, 2001.

[17] A. Habel and D. Plump. Computational completeness of programming languages based on graph transformation. In *FoSSaCS LNCS 2030*, pages 230–245, 2001.

[18] J. R. Kennaway, J. K. Klop, M. R. Sleep, and F. J. D. Vries. On the adequacy of graph rewriting for simulating term rewriting. *ACM Transactions on Programming Languages and Systems*, 16(3):493–523, 1994.

[19] D. Plump. Term graph rewriting. In H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 3–61. World Scientific, 1999.

[20] D. Plump and S. Steinert. Towards graph programs for graph algorithms. In *ICGT, LNCS 3256*, pages 128–143, 2004.

[21] P. Rodgers. A Graph Rewriting Programming Language for Graph Drawing. In *Proceedings of the 14th IEEE Symposium on Visual Languages*. IEEE, IEEE Computer Society Press, September 1998.

[22] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[23] A. Schürr, A. J. Winter, and A. Zündorf. The PROGRES approach: language and environment. In *Handbook of graph grammars and computing by graph transformation: vol. 2: applications, languages, and tools*, pages 487–550. World Scientific Publishing Co., Inc., 1999.

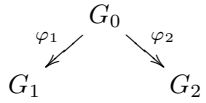
A Pushouts of graphs

Let **Gr** denote the category of graphs and **Set** the category of sets. The *node functor* $\mathcal{N} : \mathbf{Gr} \to \mathbf{Set}$ maps each graph G to its set of nodes $\mathcal{N}_G$, and each graph homomorphism $\varphi : G \to H$ to its underlying map on nodes $\varphi : \mathcal{N}_G \to \mathcal{N}_H$. As in the rest of the paper, this map is simply denoted φ, and this is not ambiguous: indeed, if two graph homomorphisms $\varphi, \psi : G \to H$ are such that their underlying maps are equal $\varphi = \psi : \mathcal{N}_G \to \mathcal{N}_H$, then it follows directly from the definition of graph homomorphisms that $\varphi = \psi : G \to H$. In categorical terms [1], this is expressed by the following result.

Proposition A.1 (Faithfulness) *The functor $\mathcal{N} : \mathbf{Gr} \to \mathbf{Set}$ is faithful.*

It is worth noting that this property does not hold for the "usual" directed multigraphs, where the set of successors of a node is unordered.

It is well-known that the category **Set** has pushouts. On the contrary, the category **Gr** does not have pushouts. For instance, let us consider a span of graphs:

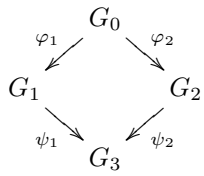


where G_0, G_1 and G_2 are made of only one node: n_0 in G_0 is unlabeled, $n_1 : a_1$ in G_1 and $n_2 : a_2$ in G_2, where a_1 and a_2 are distinct constants. This span has no pushout, because there cannot be any commutative square of graphs based on it.

Theorem A.2 below states a sufficient condition for a commutative square of graphs to be a pushout, and Theorem A.4 states a sufficient condition for a span of graphs to have a pushout, together with a construction of this pushout.

In the following, when G_i occurs as an index, it is replaced by i.

Theorem A.2 (Pushout of graphs from pushout of sets) *If a square* Γ *of the following form in the category of graphs:*



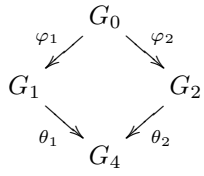
is such that:

1. Γ *is a commutative square in* **Gr**,
2. $\mathcal{N}(\Gamma)$ *is a pushout in* **Set**,
3. *and each* $n \in \mathcal{N}_3^\Omega$ *is in* $\psi_i(\mathcal{N}_i^\Omega)$ *for* $i = 1$ *or* $i = 2$,

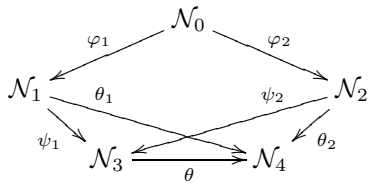
then Γ *is a pushout in* **Gr**.

Point (2) implies that each $n \in \mathcal{N}_3$ is the image of at least a node in G_1 or in G_2, and point (3) adds that, if n is labeled, then it is the image of at least a labeled node in G_1 or in G_2.

Proof. Let us consider a commutative square Γ' in **Gr** of the form:



Then $\mathcal{N}(\Gamma')$ is a commutative square in **Set**, and since $\mathcal{N}(\Gamma)$ is a pushout in **Set**, there is a unique map $\theta : \mathcal{N}_3 \to \mathcal{N}_4$ such that $\theta \circ \psi_i = \theta_i$, for $i = 1, 2$.



Let us now prove that θ actually is a graph homomorphism. According to Definition 2.4, we have to prove that, for each labeled node n of G_3, its image $n' = \theta(n)$ is a labeled node of G_4, and that $\mathcal{L}_4(n') = \mathcal{L}_3(n)$ and $\mathcal{S}_4(n') = \theta^*(\mathcal{S}_3(n))$.

So, let $n \in \mathcal{N}_3^\Omega$, and let $n' = \theta(n) \in \mathcal{N}_4$. ¿From our third assumption, without loss of generality, $n = \psi_1(n_1)$ for some $n_1 \in \mathcal{N}_1^\Omega$. It follows that $\theta_1(n_1) = \theta(\psi_1(n_1)) = \theta(n) = n'$:

$$n = \psi_1(n_1) \text{ and } n' = \theta_1(n_1) \, .$$

Since n_1 is labeled and θ_1 is a graph homomorphism, the node n' is labeled.

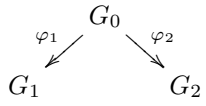
Since ψ_1 and θ_1 are graph homomorphisms, $\mathcal{L}_3(n) = \mathcal{L}_1(n_1)$ and $\mathcal{L}_4(n') = \mathcal{L}_1(n_1)$, thus $\mathcal{L}_3(n) = \mathcal{L}_4(n')$, as required for labels.

Since ψ_1 and θ_1 are graph homomorphisms, $\mathcal{S}_3(n) = \psi_1^*(\mathcal{S}_1(n_1))$ and $\mathcal{S}_4(n') = {\theta_1}^*(\mathcal{S}_1(n_1))$. So, $\theta^*(\mathcal{S}_3(n)) = \theta^*(\psi_1^*(\mathcal{S}_1(n_1))) = {\theta_1}^*(\mathcal{S}_1(n_1) = \mathcal{S}_4(n')$, as required for successors.

This proves that $\theta : G_3 \to G_4$ is a graph homomorphism. Then, from the faithfulness of the functor $\mathcal{N}$ (Proposition A.1), for $i \in \{1, 2\}$, the equality of the underlying maps $\theta \circ \psi_i = \theta_i : \mathcal{N}_i \to \mathcal{N}_4$ is an equality of graph homomorphisms: $\theta \circ \psi_i = \theta_i : G_i \to G_4$.

Now, let $\theta' : G_3 \to G_4$ be a graph homomorphism such that $\theta' \circ \psi_i = \theta_i$ for $i \in \{1, 2\}$. Since $\mathcal{N}(\Gamma)$ is a pushout in **Set**, the underlying maps are equal: $\theta = \theta' : \mathcal{N}_3 \to \mathcal{N}_4$. Then, it follows from the faithfulness of the functor $\mathcal{N}$ that the graph homomorphisms are equal: $\theta = \theta' : G_3 \to G_4$. □

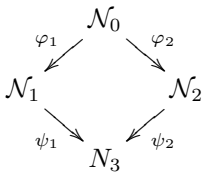
For each span of graphs Σ:



let $\sim$ denote the equivalence relation on the disjoint union $\mathcal{N}_1 + \mathcal{N}_2$ generated by:

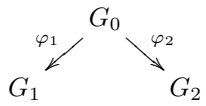
$$\varphi_1(n_0) \sim \varphi_2(n_0) \text{ for all } n_0 \in \mathcal{N}_0 \, ,$$

let N_3 be the quotient set $N_3 = (\mathcal{N}_1 + \mathcal{N}_2)/\sim$, and $\psi : \mathcal{N}_1 + \mathcal{N}_2 \to N_3$ the quotient map. Two nodes n, n' in $\mathcal{N}_1 + \mathcal{N}_2$ are called *equivalent* if $n \sim n'$. For $i \in \{1, 2\}$, let $\psi_i : \mathcal{N}_i \to N_3$ be made of the inclusion of $\mathcal{N}_i$ in $\mathcal{N}_1 + \mathcal{N}_2$ followed by ψ. Then, it is well-known that the square of sets:



is a pushout, which can be called *canonical*.

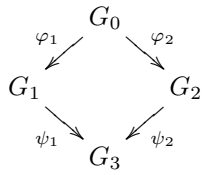
Definition A.3 (Strongly labeled span of graphs) A span of graphs:



is *strongly labeled* if for each $n_3 \in (\mathcal{N}_1 + \mathcal{N}_2)/\sim$:

- all the labeled nodes in the class n_3 have the same label,
- and all the labeled nodes in the class n_3 have equivalent successors.

Theorem A.4 (Pushout of a strongly labeled span of graphs) *A strongly labeled span of graphs has a pushout:*

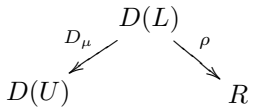


which can be built as follows:

- *the underlying square of sets is the canonical pushout square, so that* $\mathcal{N}_3 = (\mathcal{N}_1 + \mathcal{N}_2)/\sim$,
- $\mathcal{N}_3^\Omega$ *is made of the classes of* $\mathcal{N}_1 + \mathcal{N}_2$ *(modulo* $\sim$*) which contain at least one labeled node,*
- *for each* $n_3 \in \mathcal{N}_3^\Omega$*, the label of* n_3 *is the label of any labeled node in the class* n_3*,*
- *for each* $n_3 \in \mathcal{N}_3^\Omega$*, the successors of* n_3 *are the classes of the successors of any labeled node in the class* n_3*.*

Proof. It follows easily from Theorem A.2 that this square is a pushout of graphs. □

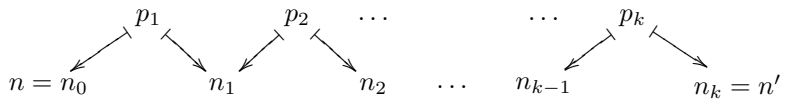
Proof of Theorem 4.7. (the notations are simplified by dropping E and $\mu(E)$). Let us prove that the following span of graphs is strongly labeled:



Then, Theorem 4.7 derives easily from Theorem A.4.

Let $n, n' \in \mathcal{N}_R^\Omega + \mathcal{N}_{D(U)}^\Omega$ be distinct equivalent nodes. We have to prove that n and n' have the same label and that their successors are pairwise equivalent.

¿From the definition of the equivalence relation $\sim$, there is a chain of relations:

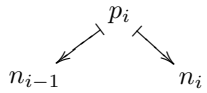


for some $k \geq 1$, where each p_i is in $\mathcal{N}_{D(L)}$, each n_i in $\mathcal{N}_{D(U)} + \mathcal{N}_R$, and the mappings are either D_μ or ρ. Let us assume that this chain has minimal length, among similar chains from n to n'. Then:

- if $p_i = p_j$ for some $i < j$, the part of the chain between p_i and p_j can be dropped, giving rise to a shorter chain from n to n': hence all the p_i's are distinct;
- if n_{i-1} and n_i are both in $\mathcal{N}_R$, then $n_{i-1} = \rho(p_i) = n_i$, and the part of the chain between n_{i-1} and n_i can be dropped, giving rise to a shorter chain from n to n': hence n_{i-1} and n_i cannot be both in $\mathcal{N}_R$;
- similarly, n_{i-1} and n_i cannot be both in $\mathcal{N}_{D(U)}$.

If all the nodes in this chain are labeled, then, since D_μ and ρ are graph homomorphisms, all nodes in the chain have the same label and have pairwise equivalent successors, so that the result follows.

We now prove that all the nodes in the chain are labeled, by contradiction. Let us assume that at least one node in the chain is unlabeled. Since ρ and D_μ are graph homomorphisms, the first unlabeled node (starting from n) is some p_i. Let us focus on:

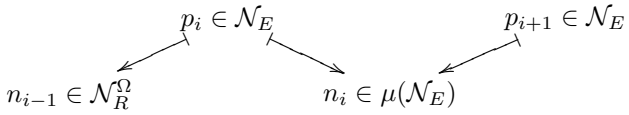


where n_{i-1} is labeled and p_i is unlabeled.

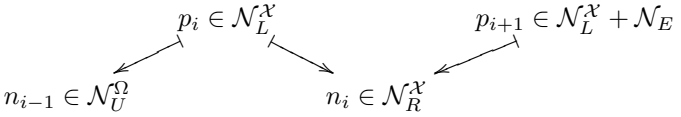
It should be reminded that:

- $\mathcal{N}_{D(L)} = \mathcal{N}_L + \mathcal{N}_E$ and $\mathcal{N}_{D(U)} = \mathcal{N}_U + \mu(\mathcal{N}_E)$, with $D_\mu(\mathcal{N}_L) \subseteq \mathcal{N}_U$ and D_μ injective on $\mathcal{N}_E$ (the last point comes from the fact that μ is Ω-injective);
- $\rho(\mathcal{N}_L^{\mathcal{X}}) \subseteq \mathcal{N}_R^{\mathcal{X}}$ and the restriction of ρ to $\mathcal{N}_L^{\mathcal{X}}$ is injective, since $L \xleftarrow{\delta_L} D(L) \xrightarrow{\rho} R$ is a rewrite rule.

Case 1: n_{i-1} is a node of R. Then $n_{i-1} \in \mathcal{N}_R^\Omega$. Since $\rho(\mathcal{N}_L^{\mathcal{X}}) \subseteq \mathcal{N}_R^{\mathcal{X}}$ and p_i is unlabeled, it follows that $p_i \in \mathcal{N}_E$. Then, since D_μ maps $\mathcal{N}_E$ to $\mu(\mathcal{N}_E)$, $n_i \in \mu(\mathcal{N}_E)$. Then $k > i$, since the last node in the chain is labeled. Since D_μ is injective on $\mathcal{N}_E$, and maps $\mathcal{N}_L$ to $\mathcal{N}_U$, it follows that $p_{i+1} = n_i$. So, $p_i = p_{i+1}$, which is impossible since the chain is minimal.



Case 2: n_{i-1} is a node of $D(U)$. Then $n_{i-1} \in \mathcal{N}_U^\Omega$. Since D_μ maps $\mathcal{N}_E$ to $\mu(\mathcal{N}_E)$ and $D_\mu(\mathcal{N}_L)$ on $\mathcal{N}_U$, it follows that $p_i \in \mathcal{N}_L^\mathcal{X}$. Since ρ maps $\mathcal{N}_L^\mathcal{X}$ to $\mathcal{N}_R^\mathcal{X}$, it follows that $n_i \in \mathcal{N}_R^\mathcal{X}$. Then $k > i$, since the last node in the chain is labeled. Then $p_{i+1} \in \mathcal{N}_L^\mathcal{X} + \mathcal{N}_E$. If $p_{i+1} \in \mathcal{N}_E$, a contradiction follows as in case 1. Hence, $p_{i+1} \in \mathcal{N}_L^\mathcal{X}$. Since the restriction of ρ to $\mathcal{N}_L^\mathcal{X}$ is injective, $p_{i+1} = p_i$, which is also impossible since the chain is minimal.



Finally, it has been proved that all the nodes in this chain are labeled, which concludes the proof. □

Proof of Corollary 4.8. We use the proof of theorem 4.7, as well as the notations in this proof. Let $n \in \mathcal{N}_V^\Omega$, we have to choose a representative $r(n)$ of n. It should be reminded that $\mathcal{N}_{D(U)}^\Omega = \mathcal{N}_U^\Omega$.

(R.) If there is a node $n_R \in \mathcal{N}_R^\Omega$ such that $n = \nu(n_R)$, let us prove that it is unique. Let $n'_R \in \mathcal{N}_R^\Omega$ be another node such that $n = \nu(n'_R)$, i.e., such that $n_R \sim n'_R$. Let us consider a chain with minimal length $k \geq 1$ from $n_R (= n_0)$ to $n'_R (= n_k)$; we know that all the nodes in this chain are labeled. Since n_0 and n_1 cannot be both in $\mathcal{N}_R$, it follows that $n_1 \in \mathcal{N}_U^\Omega$, so that $p_0, p_1 \in \mathcal{N}_L^\Omega$ and $n_1 = \mu(p_0) = \mu(p_1)$. The Ω-injectivity of μ implies that $p_0 = p_1$, but this is impossible. So, we have proved that $\nu^\Omega : \mathcal{N}_R^\Omega \to \mathcal{N}_V^\Omega$ is injective, and we define $r(n) = n_R$.

(U.) If there is no node $n_R \in \mathcal{N}_R^\Omega$ such that $n = \nu(n_R)$, then there is a node $n_U \in \mathcal{N}_U^\Omega$ such that $n = \rho'(n_U)$. Let us prove that it is unique. Let $n'_U \in \mathcal{N}_U^\Omega$ be another node such that $n = \rho'(n'_U)$, i.e., such that $n_U \sim n'_U$. Let us consider a chain with minimal length $k \geq 1$ from $n_U (= n_0)$ to $n'_U (= n_k)$; we know that all the nodes in this chain are labeled. Since n_0 and n_1 cannot be both in $\mathcal{N}_U$, it follows that $n_1 \in \mathcal{N}_R^\Omega$, which contradicts our assumption: there is no node $n_R \in \mathcal{N}_R^\Omega$ such that $n = \nu(n_R)$. Let $\widetilde{\mathcal{N}_U^\Omega}$ denote the subset of $\mathcal{N}_U^\Omega$ made of the nodes which are not equivalent to any node in $\mathcal{N}_R^\Omega$. So, we have proved that the restriction of $\rho'^\Omega : \mathcal{N}_{D(U)}^\Omega \to \mathcal{N}_V^\Omega$ to $\widetilde{\mathcal{N}_U^\Omega}$ is injective, and we define $r(n) = n_U$.

(L.) We still have to prove that $\widetilde{\mathcal{N}_U^\Omega} = \mathcal{N}_U^\Omega - \mu(\mathcal{N}_L^\Omega)$, i.e., that a node $n_U \in \mathcal{N}_U^\Omega$ is equivalent to a node $n_R \in \mathcal{N}_R^\Omega$ if and only if there is node $n_L \in \mathcal{N}_L^\Omega$ such that $n_U = \mu(n_L)$.
Clearly, if $n_L \in \mathcal{N}_L^\Omega$ and $n_U = \mu(n_L)$, let $n_R = \rho(n_L)$, then $n_R \in \mathcal{N}_R^\Omega$ and $n_U \sim n_R$.
Now, let $n_U \sim n_R$ for some $n_U \in \mathcal{N}_U^\Omega$ and $n_R \in \mathcal{N}_R^\Omega$. Let us consider a chain with minimal length $k \geq 1$ from $n_R (= n_0)$ to $n_U (= n_k)$; we know that all the nodes in this chain are labeled. If $k > 1$, then the Ω-injectivity of μ leads to a contradiction, as in part **(R)** of the proof. Hence $k = 1$, which means that $p_1 \in \mathcal{N}_L^\Omega$ is such that $n_R = \rho(p_1)$ and $n_U = \mu(p_1)$, so that there is node $n_L = p_1 \in \mathcal{N}_L^\Omega$ such that $n_U = \mu(n_L)$.

This concludes the proof that:

$$\mathcal{N}_V^\Omega = (\mathcal{N}_U^\Omega - \mu(\mathcal{N}_L^\Omega)) + \mathcal{N}_R^\Omega \ .$$

□