# Modeling Pointer Redirection as Cyclic Term-graph Rewriting [1]

Dominique Duval [2] Rachid Echahed [3] Frédéric Prost [4]

**Abstract**

We tackle the problem of data-structure rewriting including global and local pointer redirections. Each basic rewrite step may perform three kinds of actions: (i) Local redirection, the aim of which is to redirect specific pointers determined by means of a pattern ; (ii) Replacement, that may add new information to data-structures ; (iii) Global redirection, which is aimed at redirecting all pointers targeting a node towards another one. We define a new framework, following the double-pushout approach, where graph rewrite rules may mix these three kinds of actions in a row. We define first the category of graphs we consider and then we define rewrite rules as pairs of graph homomorphisms of the form $L \leftarrow K \rightarrow R$. In our setting, graph $K$ is not arbitrary, it is used to encode pointer redirection. Furthermore, pushouts do not always exist and complement pushouts, when they exist, are not unique. Despite these concerns, our definition of rewriting steps is such that a rewrite rule can always be fired, once a matching is found.

*Key words:* Graph rewriting, category theory, double pushout, pointer redirection.

## 1 Introduction

Pointers allow one to design efficient implementations of algorithms. However, it is well-known that pointer manipulation is error-prone. Due to their flexibility and power, programs which handle pointers are more difficult to write and to maintain. Actually, pointers are a source of different kinds of bugs such as *memory leaks* (i.e. a memory cell can no longer be reached because there is no pointer to it) or *segmentation fault* (i.e. one tries to access a cell which has been deallocated). So, formal techniques to write and validate programs manipulating pointers are crucial to enhance the quality of software. Formally,

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

[2] Dominique.Duval@imag.fr ; Laboratoire LMC ; Grenoble, France

[3] Rachid.Echahed@imag.fr ; Laboratoire Leibniz ; Grenoble, France

[4] Frederic.Prost@imag.fr ; Laboratoire Leibniz ; Grenoble, France

*This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science*
*URL:* www.elsevier.nl/locate/entcs

data-structures are particular graphs and pointers are edges. General frameworks of graph transformation are now well established, see e.g. [28,16,17]. On the other hand, rewriting techniques have been shown to be very useful to establish formal bases of very high level programming languages as well as theorem provers. These techniques have been widely investigated for strings [10], trees or terms [3] and term graphs (or dags) [25,8].

In this paper we follow the double pushout approach [12,22] of graph transformation and propose a new class of graph rewrite systems aiming at rewriting (cyclic) data-structures with pointers such as circular lists, doubly-linked lists etc. Our proposal focuses on pointer rewriting and thus departs from existing ones such as [7,24,15,11](see related work section below for further discussion). A rewrite rule is defined as a pair of graph (data-structure) homomorphisms $L \leftarrow K \rightarrow R$. The graph $K$ plays a key role in pointer rewriting.

Hereafter we illustrate our approach through an example. In Fig. 1, we give an example of a rule introducing our approach. An application of this rule is given in Fig. 2, where graph $G_1$ is rewritten into $G_2$. $G_2$ is obtained from $G_1$ by (i) adding cell $C'$, (ii) redirecting the pointer outgoing cell $C1$ to point $C'$ and finally, (iii) redirecting all pointers targeting cell $C2$, but the one outgoing $C1$, to point to cell $C1$. To achieve such transformation, the left-hand side $L$ of the considered rewrite rule consists of two linked cells $C1$ and $C2$. The role of the considered rewrite rule consists in:

(i) Adding cell $C'$

(ii) Redirecting the pointer outgoing $C1$ to point to a new cell $C'$. Such (local) redirection of pointer is achieved by disconnecting the considered pointer (outgoing $C1$) in graph $K$ and making it point to a new (variable) unlabeled node $p$ such that $l(p) = C2$ and $r(p) = C'$.

(iii) Redirecting all pointers targeting cell $C2$, but the pointer outgoing $C1$, to point to cell $C1$. To perform such (global) redirection of pointers, we add a new (variable) unlabeled node, $\gamma$, in graph $K$ such that $l(\gamma) = C2$ and $r(\gamma) = C1$. $\gamma$ can be isolated in graph $K$, i.e. $\gamma$ is not necessarily target of any pointer in $K$. The role of $\gamma$ can be better explained when one considers the double-pushouts that define a rewrite step as in Fig 3. Indeed, rewriting a graph $G$ by using the considered rule, consists, in particular, to disconnect, in graph $D$ (Fig. 3), all pointers targeting the image of cell $C2$ in $G$ and redirect them to the image of $\gamma$ in $D$. Details of this construction are given in section 4.

In general, one may perform several local and global pointer redirections in one step. For each redirection, new (variable) unlabeled node should be added with the right definitions of morphisms $l$ and $r$.

The considered homomorphisms of a rewrite rule ($l, r, m$ in Fig. 3) are not necessarily injective in our setting, unlike classical assumptions as in the recent proposals dedicated to graph programs [26,23], and complement pushouts (graph $D$ in Fig. 3) are not unique.

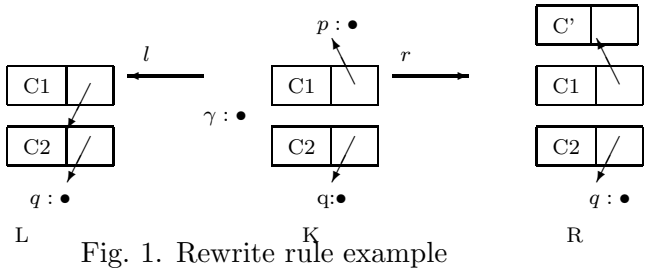



Fig. 1. Rewrite rule example

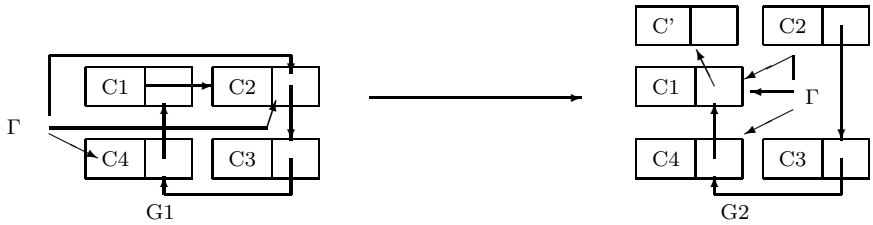



Fig. 2. Rewrite step example

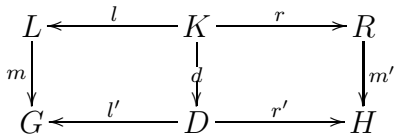



Fig. 3. Double pushout: a rewrite step ($G \rightarrow H$)

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.

*Related Work*

Term graph rewriting [7,25,8] has been mainly motivated by implementation issues of functional programming languages. These motivations impact clearly their definition.

In [21,13] jungles, a representation of acyclic term graphs by means of hyper-graphs, 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 [7,24,15] 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 [6], 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 [6] are defined by using the notion of an opfibration of a category while our approach is based on double-pushouts.

[11] is a work combining a categorical approach and cyclic term graphs. It departs from our work in the fact that the explicit manipulation of edges is not

handled. Actually there is an edge manipulation during the redirection phase which corresponds to our notion of global redirection. But it is limited to the root and cannot be handled by the programmer. In [14] cyclic graphs are also studied using addressed term rewriting systems. In this case too the direct manipulation of pointers is not addressed. The same remark can be done for [9] that is an extension of $\rho$-calculus able to deal with cyclic structures. It is not possible in these systems to express the update of a shared data. The work of [1] considers an equational framework for cyclic graph rewriting, it also cannot handle explicitly pointers.

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 role that we tried to take into account by proposing for instance redirections of specific edges within rewrite rules.

In [23], Habel and Plump proposed a kernel language for graph transformation. This language has been improved recently in [26]. 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, complement pushouts are not unique in our setting which is not the case in [23,26].

Recently, in [5,4] the authors are also interested in classical data-structures built by using pointers. Their work is complementary to ours; they proposed *Graph reduction specifications* as a framework to *recognize* data-structure shapes in order to check the type safety of pointer algorithms.

Last, but not least, there are some programming languages which provide graph transformation features (see, e.g. [29,18,20,27]). 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.

The rest of the paper is organised as follows. The next section introduces the category of graphs we consider. Section 3 states some technical results that help defining rewrite steps. Section 4 introduces data-structure rewriting (including global and local redirection of pointers) through a double-pushout approach. Concluding remarks are given in section 5. Proofs may be found in the appendix. We assume some familiarity with basic notions of category theory (see e.g. [2] 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 and are the ones in-

troduced in [7]. 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 $\Omega$ *is a set of operation symbols such that each operation symbol in* $\Omega$*, say* $f$*, is provided with 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 $f : A \to B$ be a function. We note $f^* : A^* \to B^*$ the unique extension of $f$ over strings defined by $f^*(\epsilon) = \epsilon$ where $\epsilon$ is the empty string and $f^*(a_1 \dots a_n) = f(a_1) \dots f(a_n)$.
We assume that $\Omega$ is fixed throughout the rest of the paper.

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

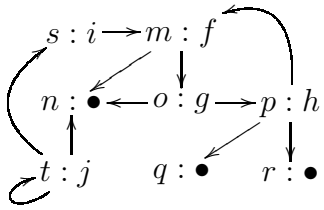
- *a set of* nodes $\mathcal{N}_G$*,*
- *a subset of* labeled nodes $\mathcal{N}_G^\Omega \subseteq \mathcal{N}_G$*,*
- *a* labeling function $\mathcal{L}_G : \mathcal{N}_G^\Omega \to \Omega$*,*
- *a* successor function $\mathcal{S}_G : \mathcal{N}_G^\Omega \to \mathcal{N}_G^*$*,*

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

We can remark the following fact: 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 $i \in \{1, \dots, \mathrm{ar}(n)\}$, the *source* of an edge $(n, i)$ is the node $n$, and its *target* is the node $succ_G(n, i)$, $f = \mathcal{L}_G(n)$ is written as $n : f$, the set of unlabeled nodes of $G$ is denoted $\mathcal{N}_G^\mathcal{X}$, so that: $\mathcal{N}_G = \mathcal{N}_G^\Omega +$ [5] $\mathcal{N}_G^\mathcal{X}$ .

**Example 2.3** Let $G$ be the graph defined by $\mathcal{N}_G = \{m; n; o; p; q; r; s; t\}$, $\mathcal{N}_G^\Omega = \{m; o; p; s; t\}$, $\mathcal{N}_G^\mathcal{X} = \{n; q; r\}$, $\mathcal{L}_G$ is defined by: $[m \mapsto f; o \mapsto g; p \mapsto h; s \mapsto i; t \mapsto j]$, $\mathcal{S}_G$ is defined by: $[m \mapsto no; o \mapsto np; p \mapsto qrm; s \mapsto m; t \mapsto tsn]$.

Graphically we represent this graph as:



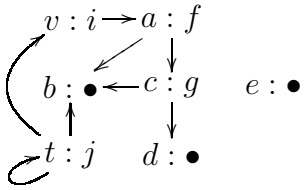

We use • to denote lack of label. Informally, one may think of • as anonymous variables.

**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 for each labeled node* $n$*,* $\mathcal{L}_H(\varphi(n)) = \mathcal{L}_G(n)$ *and* $\mathcal{S}_H(\varphi(n)) = \varphi^*(\mathcal{S}_G(n))$*.*

[5] + stands for disjoint union.

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: $[a \mapsto m; b \mapsto n; c \mapsto o; d \mapsto p; e \mapsto p; v \mapsto s; t \mapsto t]$. Map $\varphi$ 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 Disconnections

This section is dedicated to some technical definitions, in order to simplify the definition of rewrite rules in the next section.

First, we define a disconnection of a graph $L$, it is made of a graph $K$ and a graph homomorphism $l : K \to L$; roughly speaking, the graph $K$ is obtained by redirecting some edges of $L$ towards new, unlabeled targets, and the homomorphism $l$ reconnects all the disconnected nodes.

**Definition 3.1 (Disconnection of a graph)** *A* disconnection *of a graph $L$ is made of a graph $K$ and a graph homomorphism: $L \xleftarrow{\ l\ } K$ such that $l$ is surjective on the nodes and bijective on the labeled nodes.*

The next result builds some kinds of disconnections of graphs.

**Proposition 3.2** *Let $L$ be a graph, $E$ a set of edges and $N$ a set of nodes of $L$. Let $K$ be the graph defined by:*

- *$\mathcal{N}_K = \mathcal{N}_L + \mathcal{N}_E + \mathcal{N}_N$, where $\mathcal{N}_E$ is made of one new node $n[i]$ for each edge $(n,i) \in E$ and $\mathcal{N}_N$ is made of one new node $n[0]$ for each node $n \in N$,*
- *$\mathcal{N}_K^\Omega = \mathcal{N}_L^\Omega$,*
- *for each $n \in \mathcal{N}_L^\Omega$: $\mathcal{L}_K(n) = \mathcal{L}_L(n)$,*
- *for each $n \in \mathcal{N}_L^\Omega$ and $i \in \{1, \ldots, \mathrm{ar}(n)\}$:*
  - *if $(n,i) \notin E$ then $succ_K(n,i) = succ_L(n,i)$,*
  - *if $(n,i) \in E$ then $succ_K(n,i) = n[i]$,*

*Let $l : K \to L$ be the graph homomorphism defined by: $l(n) = n$ if $n \in \mathcal{N}_L$ , $l(n[i]) = succ_L(n,i)$ if $n[i] \in \mathcal{N}_E$, $l(n[0]) = n$ if $n[0] \in \mathcal{N}_N$.*

*Then, $l : K \to L$ is a disconnection of $L$.*

**Definition 3.3** *The disconnection defined in proposition 3.2 is called* the disconnection of $L$ with respect to $E$ and $N$.

Now, the notion of disconnection is extended, from a graph $L$ to a graph homomorphism $m : L \to G$.

**Definition 3.4 (Disconnection of a graph homomorphism)** *A* disconnection *of a graph homomorphism $m : L \to G$ is a commutative square:*

$$\begin{array}{ccc} L & \xleftarrow{\ l\ } & K \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle d} \\ G & \xleftarrow{\ l'\ } & D \end{array}$$

*where $l$ and $l'$ are disconnections.*

The next result builds some kinds of disconnections of graph homomorphisms.

**Proposition 3.5** *Let $L$ be a graph, $E$ a set of edges and $N$ a set of nodes of $L$, and $l : K \to L$ the disconnection of $L$ with respect to $E$ and $N$. Let $m : L \to G$ be a graph homomorphism such that the restriction of $m$ to $(\mathcal{N}_L^\Omega \cup N)$ is injective. Let $E' = m(E)$ and $N' = m(N)$. Let $D$ be the graph defined by:*

- *$\mathcal{N}_D = \mathcal{N}_G + \mathcal{N}_{E'} + \mathcal{N}_{N'}$, where $\mathcal{N}_{E'}$ is made of one new node $p[i]$ for each edge $(p, i) \in E'$ and $\mathcal{N}_{N'}$ is made of one new node $p[0]$ for each node $p \in N'$,*
- *$\mathcal{N}_D^\Omega = \mathcal{N}_G^\Omega$,*
- *for each $p \in \mathcal{N}_G^\Omega$: $\mathcal{L}_D(p) = \mathcal{L}_G(p)$,*
- *for each $p \in \mathcal{N}_G^\Omega$ and $i \in \{1, \ldots, \mathrm{ar}(p)\}$:*
  - *if $p \in m(\mathcal{N}_L^\Omega)$ and $(p, i) \notin E'$ then $succ_D(p, i) = succ_G(p, i)$,*
  - *if $p \in m(\mathcal{N}_L^\Omega)$ and $(p, i) \in E'$ then $succ_D(p, i) = p[i]$,*
  - *if $p \notin m(\mathcal{N}_L^\Omega)$ and $succ_G(p, i) \notin N'$ then $succ_D(p, i) = succ_G(p, i)$,*
  - *if $p \notin m(\mathcal{N}_L^\Omega)$ and $succ_G(p, i) \in N'$ then $succ_D(p, i) = succ_G(p, i)[0]$,*

*Let $l' : D \to G$ be the graph homomorphism defined by: $l'(p) = p$ if $p \in \mathcal{N}_G$, $l'(p[i]) = succ_G(p, i)$ if $p[i] \in \mathcal{N}_{E'}$, $l'(p[0]) = p$ if $p[0] \in \mathcal{N}_{N'}$.*
*Let $d : K \to D$ be the graph homomorphism defined by: $d(n) = m(n)$ if $n \in \mathcal{N}_L$, $d(n[i]) = m(n)[i]$ if $n[i] \in \mathcal{N}_E$, $d(n[0]) = m(n)[0]$ if $n[0] \in \mathcal{N}_N$.*
*Then, we get a disconnection of $m : L \to G$.*

*Proof.* The fact that $l'$ is a disconnection is easy to check. For the commutativity, let $n$ be a node of $K$, then:

- if $n \in \mathcal{N}_L$ then $l'(d(n)) = l'(m(n)) = m(n) = m(l(n))$,
- if $n[i] \in \mathcal{N}_E$ then $l'(d(n)) = l'(m(n)[i]) = succ_G(m(n), i) = m(succ_L(n, i)) = m(l(n[i]))$,
- if $n[0] \in \mathcal{N}_N$ then $l'(d(n[0])) = l'(m(n)[0]) = m(n) = m(l(n[0]))$,

so that $l' \circ d = m \circ l$, as required. □

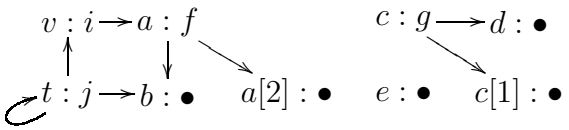
**Definition 3.6** *The disconnection defined in proposition 3.5 is called* the disconnection of $m$ with respect to $E$ and $N$, *or* the disconnection of $m$ extending $l$.

**Remark 3.7** *With above notations, the map $d : \mathcal{N}_K \to \mathcal{N}_D$ can be described as the sum of three maps: $d = d_L + d_E + d_N$ with: $d_L = m : \mathcal{N}_L \to \mathcal{N}_G$, $d_E : \mathcal{N}_E \to \mathcal{N}_{E'}$ such that $n[i] \mapsto m(n)[i]$, $d_N : \mathcal{N}_N \to \mathcal{N}_{N'}$ such that $n[0] \mapsto m(n)[0]$. If, in addition, the restriction of $m$ to $(\mathcal{N}_L^\Omega \cup N)$ is injective, then both $d_E$ and $d_N$ are bijections.*

**Theorem 3.8 (A pushout square)** *Let $m : L \to G$ be a graph homomorphism, $E$ a set of edges and $N$ a set of nodes of $L$, such that the restriction of $m$ to $(\mathcal{N}_L^\Omega \cup N)$ is injective. Then, the disconnection of $m$ with respect to $E$ and $N$ is a pushout in the category of graphs.*

*Proof.* This result is an easy corollary of Theorem A.3 (cf. the appendix). □

**Example 3.9** Consider the graph $H$ of Example 2.5. Then the disconnected graph, $H_d$, associated to $H$ and the set of edges $\{(a, 2); (c, 1)\}$ is the following graph:




Now if we consider the graph homomorphism $\varphi : H \to G$ defined in Example 2.5, the disconnection of a graph homomorphism leads to the following homomorphism : $D_\varphi : H_d \to G_d$, where $G_d$ is the disconnection of $G$ relatively to edges $\{(\varphi(a), 2); (\varphi(c), 1)\}$, is the mapping $[a \mapsto m; b \mapsto n; c \mapsto o; d \mapsto p; e \mapsto p; a[2] \mapsto m[2]; c[1] \mapsto o[1]; v \mapsto s; t \mapsto t]$

## 4 Data-structure rewriting

A rewrite step is defined from a rewrite rule and a matching. A rewrite rule is a kind of span of graphs (a *span* is a pair of homomorphisms with a common source): $L \xleftarrow{l} K \xrightarrow{r} R$. A matching is a kind of morphism of graphs: $L \xrightarrow{m} G$. The role of a rewrite step consists in:

- adding to $G$ an instance of the right-hand side $R$,
- performing some local redirections of edges in $G$: some edges, in the image of the matching, are redirected to other target nodes,
- performing some global redirections of edges in $G$: all incoming edges of some nodes, except those in the image of the matching, are redirected to other target nodes,

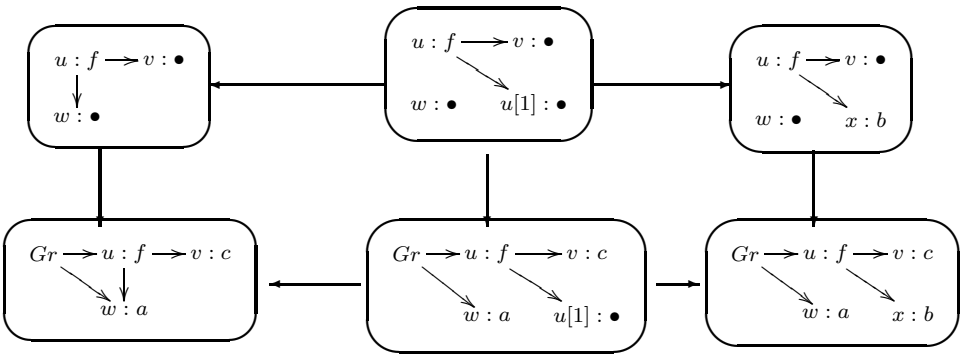



Fig. 4. Local Redirection

We use the double-pushout approach to define a rewrite step. We do not deal with deletion of items in this paper ; this could easily be performed by means of the notion of rooted graphs and the use of garbage collection.

**Definition 4.1 (Rewrite rule)** *A* rewrite rule*, or* production*, is a span of graph homomorphisms of the form:* $L \xleftarrow{l} K \xrightarrow{r} R$ *where* $l : K \to L$ *is the disconnection of* $L$ *with respect to a set of edges* $E$ *and a set of nodes* $N$ *of* $L$ *and where the restriction of* $r$ *to* $\mathcal{N}_L^{\mathcal{X}}$ *is injective and has its values in* $\mathcal{N}_R^{\mathcal{X}}$*. The* locally redirected edges *of* $p$ *are the edges in* $E$*, and its* globally redirected nodes *are the nodes in* $N$*.*

The reader may notice that the rewrite rules we consider are different from *disconnected productions* of [19]. Actually, we do not use the same notion of disconnected graphs. In general, disconnected graphs according to our definition do not fulfill the disconnection conditions given in [19].

We now give two toy examples to illustrate this definition. We start by the local redirection mechanism.

**Example 4.2** Let us observe on the following double pushout how local redirection works. Consider the double pushout given in Fig. 4. We would like to redirect the first argument (pointer) of the function $f$ to a new one, say $x : b$. This is done by the introduction in $K$ of an unlabeled node $u[1]$. This node is mapped to the actual argument in $L$ ($w$) and to the new target $x : b$ (with $b$ some 0-ary operator) in $R$ by morphisms. Notice also that edges coming from other parts of the graph (symbolized by $Gr$) are not modified by this (local) redirection.

**Example 4.3** In this example we show how global redirection works. A Global redirection is intended to redirect in a row, all pointers in the environment, but those in the left-hand side, which point a particular node, to point to a new node. In the example given in figure 5, we want to redirect all edges with target $n$, but $(n, 1)$, towards $o$. For this purpose, we define a rewrite rule (i.e. a span) . We introduce a node $n[0]$ in $K$. $n[0]$ is associated by morphisms to $n$ in $L$ and to $o$ in $R$.

**Definition 4.4 (Matching)** *Let* $p$ *be a rewrite rule* $L \xleftarrow{l} K \xrightarrow{r} R$*. A*

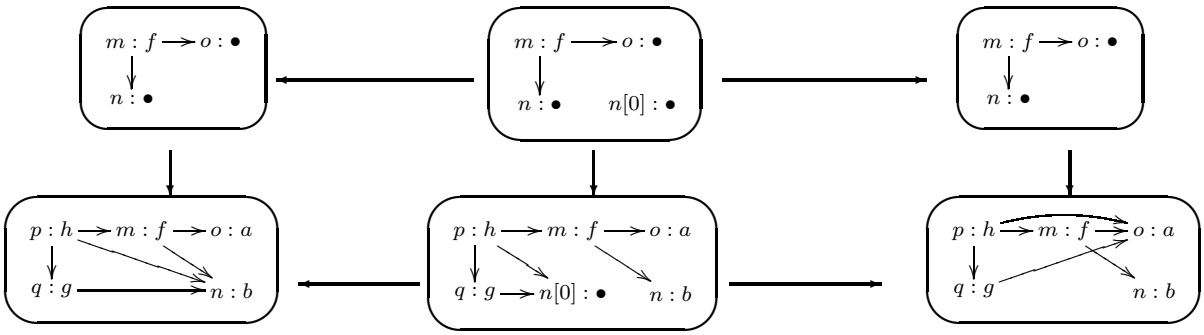



Fig. 5. Global Redirection

matching *with respect to* $p$ *is a graph homomorphism* $m : L \to G$ *such that the restriction of* $m$ *to* $(\mathcal{N}_L^\Omega \cup N)$ *is injective, where* $N$ *is the set of globally redirected nodes of* $p$.

**Definition 4.5 (Rewrite step)** *Let* $p$ *be the rewrite rule* $L \xleftarrow{l} K \xrightarrow{r} R$ *and* $m : L \to G$ *a matching with respect to* $p$. *Let:*

$$\begin{array}{ccc} L & \xleftarrow{l} & K \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle d} \\ G & \xleftarrow{l'} & D \end{array}$$

*be the disconnection of* $m$ *extending* $l$. *Then* $G$ rewrites to $H$ using rule $p$ *if there are graph homomorphisms* $m' : R \to H$ *and* $r' : D \to H$ *such that the following square is a pushout in the category of graphs:*

$$\begin{array}{ccc} K & \xrightarrow{r} & R \\ {\scriptstyle d}\downarrow & & \downarrow{\scriptstyle m'} \\ D & \xrightarrow{r'} & H \end{array}$$

According to theorem 3.8 and to the definition of a matching, the disconnection of $m$ extending $l$ is a pushout. So, a rewrite step corresponds to a *double pushout* in the category of graphs. However, the reader can easily verify that, in general, double pushouts do not always exist whenever the matching $m$ is non injective on $(\mathcal{N}_L^\Omega \cup N)$ (see Definition 4.4) and the complement pushout $D$ is not unique :

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle d} & & \downarrow{\scriptstyle m'} \\ G & \xleftarrow{l'} & D & \xrightarrow{r'} & H \end{array}$$

**Theorem 4.6 (Rewrite step is feasible)** *Let* $p$ *be a rewrite rule and* $m : L \to G$ *a matching with respect to* $p$. *Then* $G$ *can be rewritten using rule* $p$.

*More precisely, the required pushout can be built as follows, with the notations from definition 4.5:*

- *the set of nodes of* $H$ *is* $\mathcal{N}_H = (\mathcal{N}_R + \mathcal{N}_D)/\sim$*, where* $\sim$ *is the equivalence relation generated by* $d(n) \sim r(n)$ *for each node* $n$ *of* $K$*,*
- *the maps* $m'$ *and* $r'$*, on the sets of nodes, are the inclusions of* $\mathcal{N}_R$ *and* $\mathcal{N}_D$ *in* $\mathcal{N}_R + \mathcal{N}_D$*, respectively, followed by the quotient map with respect to* $\sim$*,*
- $\mathcal{N}_H^\Omega$ *is made of the classes modulo* $\sim$ *which contain at least one labeled node (let* $\rho : \mathcal{N}_H^\Omega \to \mathcal{N}_R^\Omega + \mathcal{N}_D^\Omega$ *be a section of the quotient map, which means that the class of* $\rho(n)$ *is* $n$*, for each* $n \in \mathcal{N}_H^\Omega$*),*
- *for each* $n \in \mathcal{N}_H^\Omega$*, the label of* $n$ *is the label of* $\rho(n)$*,*
- *for each* $n \in \mathcal{N}_H^\Omega$*, the successors of* $n$ *are the classes of the successors of* $\rho(n)$*,*

*Moreover, the resulting pushout does not depend on the choice of the section* $\rho$*.*

**Corollary 4.7 (A description of the nodes)** *With the notations and assumptions of Theorem 4.6, the representatives of the equivalence classes of nodes of* $\mathcal{N}_R + \mathcal{N}_D$ *can be chosen in such a way that:*

$$\mathcal{N}_H^\Omega = (\mathcal{N}_G^\Omega - m(\mathcal{N}_L^\Omega)) + \mathcal{N}_R^\Omega \quad \text{and} \quad \mathcal{N}_H^\mathcal{X} = \mathcal{N}_G^\mathcal{X} + (\mathcal{N}_R^\mathcal{X} - r(\mathcal{N}_L^\mathcal{X}))\ .$$

*Proof.* Both Theorem 4.6 and Corollary 4.7 are derived from Theorem A.5, their proofs are given at the end of the appendix. □

## 5 Conclusion

We proposed a new framework for cyclic data-structure rewriting, including pointer redirections. The rewrite relationships induced by our rewrite rules 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 guaranteed even for non-overlapping rewrite systems, and thus user-definable strategies are necessary when using all the power of data-structure rewriting.

On the other hand, data-structures are better represented by means of graphics (e.g. [27]). Our purpose in this paper was rather the definition of the basic rewrite steps for data-structures. We intend to consider syntactic issues in a future work.

## References

[1] Z. M. Ariola and J. W. Klop. Equational term graph rewriting. *Fundamentae Informaticae*, 26(3/4):207–240, 1996.

[2] 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.

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

[4] A. Bakewell, D. Plump, and C. Runciman. Checking the shape safety of pointer manipulations. In *RelMiCS*, pages 48–61, 2003.

[5] A. Bakewell, D. Plump, and C. Runciman. Specifying pointer structures by graph reduction. In *AGTIVE*, pages 30–44, 2003.

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

[7] 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.

[8] 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.

[9] C. Bertolissi, P. Baldan, H. Cirstea, and C. Kirchner. A rewriting calculus for cyclic higher-order term graphs. *Electr. Notes Theor. Comput. Sci.*, 127(5):21–41, 2005.

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

[11] A. Corradini and F. Gadducci. Rewriting on cyclic structures: Equivalence between the operational and the categorical description. *ITA*, 33(4/5):467–493, 1999.

[12] 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.

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

[14] D. J. Dougherty, P. Lescanne, L. Liquori, and F. Lang. Addressed term rewriting systems: Syntax, semantics, and pragmatics: Extended abstract. *Electr. Notes Theor. Comput. Sci.*, 127(5):57–82, 2005.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] F. Gadducci, R. Heckel, and M. Llabrés. A bi-categorical axiomatisation of concurent graph rewriting. *Electronic Notes in Theoretical Computer Science*, 29, 1999.

[20] 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.

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

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

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

[24] 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.

[25] 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.

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

[27] 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.

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

[29] 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.

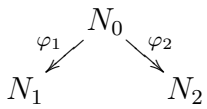
# A Pushouts of graphs

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 $\varphi$, 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 [2], 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.

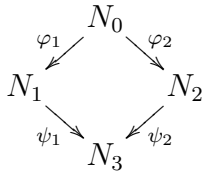
It is well-known that the category **Set** has pushouts, which can be built as follows. For each span of sets:




let $\sim$ denote the equivalence relation on the disjoint union $N_1 + N_2$ generated by:

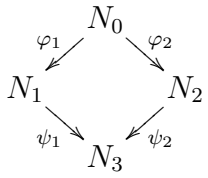
$$\varphi_1(n_0) \sim \varphi_2(n_0) \quad \text{for all} \quad n_0 \in N_0 \ ,$$

let $N_3$ be the quotient $N_3 = (N_1 + N_2)/\sim$, and $\psi : N_1 + N_2 \to N_3$ the quotient map. Two nodes $n, n'$ in $N_1 + N_2$ are called *equivalent* if $n \sim n'$. For $i \in \{1, 2\}$, let $\psi_i : N_i \to N_3$ be made of the inclusion of $N_i$ in $N_1 + N_2$ followed by $\psi$. Then, it is well-known that the following square of sets is a pushout, which will be called *canonical*:

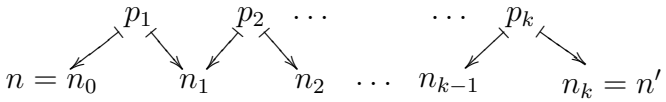



The next lemma will be used in the proofs of Theorem 4.6 and Corollary 4.7.

**Lemma A.2** *Let us consider a canonical pushout of sets:*




*Let $n, n' \in N_1 + N_2$ be distinct equivalent nodes. From the definition of the equivalence relation $\sim$, there is a chain of relations:*



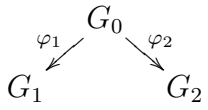

*with length $2k$ for some $k \geq 1$, where each $p_i$ is in $N_0$, each $n_i$ in $N_1 + N_2$, and the mappings are either $\phi_1$ or $\phi_2$. Let us assume that this chain has minimal length, among similar chains from $n$ to $n'$. Then:*

- *all the $p_i$'s are distinct;*
- *two consecutive $n_i$'s cannot be both in $N_1$, nor both in $N_2$, so that $n_i = \phi_{j(i)}(p_i) = \phi_{j(i)}(p_{i+1})$ for each $i$, where $j(i)$ is alternatively 1 and 2;*
- *if $\widetilde{N}$ is a subset of $N_0$ such that the restriction of $\phi_{j(i)}$ to $\widetilde{N}$ is injective, then $p_i$ and $p_{i+1}$ cannot be both in $\widetilde{N}$.*

*Proof.*

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 the same $N_j$ (for $j = 1$ or 2), then $n_{i-1} = \phi_j(p_i) = n_i$, 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 $N_j$. If $n_i = \phi_{j(i)}(p_i) = \phi_{j(i)}(p_{i+1})$ with both $p_i$ and $p_{i+1}$ in $\widetilde{N}$ and the restriction of $\psi_{j(i)}$ to $\widetilde{N}$ is injective, then $p_i = p_{i+1}$, in contradiction with the first point. □

In contrast with **Set**, the category **Gr** does not have pushouts. For instance, let us consider a span of graphs:

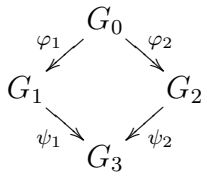



where $G_0$, $G_1$ and $G_2$ are made of only one node: $n_0$ in $G_0$ is unlabeled, $n_1 : a_1$ in $G_1$ and $n_2 : a_2$ in $G_2$, where $a_1$ and $a_2$ are distinct constants. This span has no pushout, because there cannot be any commutative square of graphs based on it.

Theorem A.3 below states a sufficient condition for a commutative square of graphs to be a pushout, and Theorem A.5 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.3 (Pushout of graphs from pushout of sets)** *If a square* $\Gamma$ *of the following form in the category of graphs:*




*is such that:*

(i) $\Gamma$ *is a commutative square in* **Gr**,

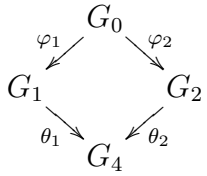
(ii) $\mathcal{N}(\Gamma)$ *is a pushout in* **Set**,

(iii) *and each* $n \in \mathcal{N}_3^\Omega$ *is in* $\psi_i(\mathcal{N}_i^\Omega)$ *for* $i = 1$ *or* $i = 2$,

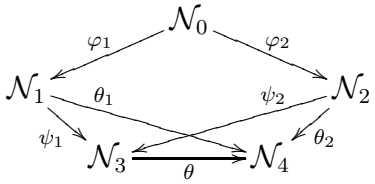
*then* $\Gamma$ *is a pushout in* **Gr**.

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 $\Gamma'$ 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 $\theta$ 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 $\theta_1$ is a graph homomorphism, the node $n'$ is labeled.

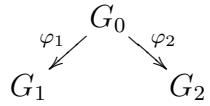
Since $\psi_1$ and $\theta_1$ are graph homomorphisms, $\mathcal{L}_3(n) = \mathcal{L}_1(n_1)$ and $\mathcal{L}_4(n') = \mathcal{L}_1(n_1)$, thus $\mathcal{L}_3(n) = \mathcal{L}_4(n')$, as required for labels.

Since $\psi_1$ and $\theta_1$ 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.

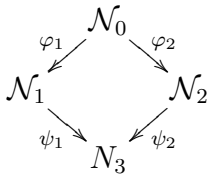
This proves that $\theta : G_3 \to G_4$ is a graph homomorphism. Then, from the faithfulness of the functor $\mathcal{N}$ (Proposition A.1), for $i \in \{1, 2\}$, the equality of the underlying maps $\theta \circ \psi_i = \theta_i : \mathcal{N}_i \to \mathcal{N}_4$ is an equality of graph homomorphisms: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

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: $\theta = \theta' : G_3 \to G_4$. □

**Definition A.4 (Strongly labeled span of graphs)** *Let us consider a span of graphs $\Sigma$:*



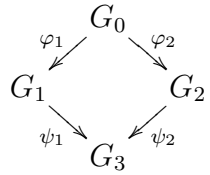

*and the canonical pushout of sets:*




*Then $\Sigma$ is* strongly labeled *if for each $n_3 \in N_3$, i.e., 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.5 (Pushout of a strongly labeled span of graphs)** *A strongly labeled span of graphs has a pushout:*

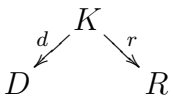



*which can be built as follows:*

- *the underlying pushout of sets is the canonical pushout, 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* $n_3$*.*

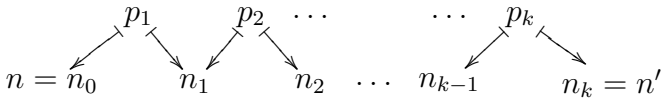
*Proof.* It follows easily from Theorem A.3 that this square is a pushout of graphs. □

*Proof of Theorem 4.6.* Let us prove that the following span of graphs is strongly labeled:




Then, Theorem 4.6 derives easily from Theorem A.5.

Let $n, n' \in \mathcal{N}_R^\Omega + \mathcal{N}_D^\Omega$ be distinct equivalent nodes. We have to prove that $n$ and $n'$ have the same label and that their successors are pairwise equivalent. Let us consider a chain of relations:

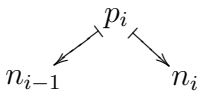



of minimal length $2k$, with each $p_i$ in $\mathcal{N}_K$, each $n_i$ in $\mathcal{N}_D$ or in $\mathcal{N}_R$, and mappings either $d$ or $r$, so that lemma A.2 can be applied to this chain. In particular, since $d = m + d_E + d_N$ with $m : \mathcal{N}_L \to \mathcal{N}_G$, $d_E : \mathcal{N}_E \to \mathcal{N}_{E'}$, $d_N : \mathcal{N}_N \to \mathcal{N}_{N'}$, and $d_E$, $d_N$ are bijections, and since $p_i \neq p_{i+1}$, we get:

(A.1) If $i < k$, it cannot happen that $p_i \in \mathcal{N}_E + \mathcal{N}_N$ and $n_i = d(p_i)$.

If all the nodes in this chain are labeled, then, since $d$ and $r$ are graph homomorphisms, all nodes in the chain have the same label and have pairwise equivalent successors, so that the result follows: $n$ and $n'$ have the same label.

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 $r$ and $d$ are graph homomorphisms, the first unlabeled node (starting from $n$) is some $p_i$. Let us focus on such a situation, where $n_{i-1}$ is labeled and $p_i$ is unlabeled:




So, $p_i \in \mathcal{N}_K^\mathcal{X} = \mathcal{N}_L^\mathcal{X} + \mathcal{N}_E + \mathcal{N}_N$.

*(R.)* Let $n_{i-1}$ be a node of $R$, i.e., $n_{i-1} \in \mathcal{N}_R^\Omega$. Then $n_{i-1} = r(p_i)$ and $n_i = d(p_i)$. If $p_i \in \mathcal{N}_L^\mathcal{X}$ then $n_{i-1} \in \mathcal{N}_R^\mathcal{X}$, a contradiction to $n_{i-1} \in \mathcal{N}_R^\Omega$. If

$p_i \in \mathcal{N}_E + \mathcal{N}_N$ then $n_i = d(p_i)$ is unlabeled, so that $i < k$; from remark (A.1), this situation cannot occur.

*(D.)* Let $n_{i-1}$ be a node of $D$, i.e., $n_{i-1} \in \mathcal{N}_D^\Omega$, or equivalently $n_{i-1} \in \mathcal{N}_G^\Omega$. Then $n_{i-1} = d(p_i)$ and $n_i = r(p_i)$. If $p_i \in \mathcal{N}_E + \mathcal{N}_N$ then $n_{i-1} = d(p_i)$ is unlabeled, a contradiction to our assumption. If $p_i \in \mathcal{N}_L^\mathcal{X}$ then $n_i = r(p_i)$ is unlabeled, so that $i < k$; then $p_{i+1}$ is an unlabeled node of $D$, which means that $p_{i+1} \in \mathcal{N}_L^\mathcal{X} + \mathcal{N}_E + \mathcal{N}_N$. If $p_{i+1} \in \mathcal{N}_L^\mathcal{X}$, since the restriction of $r$ to $\mathcal{N}_L^\mathcal{X}$ is injective, a contradiction follows from lemma A.2. If $p_{i+1} \in \mathcal{N}_E + \mathcal{N}_N$ then $n_{i+1} = d(p_{i+1})$ is unlabeled, so that $i+1 < k$; from remark (A.1), this situation cannot occur.

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

*Proof of Corollary 4.7.* Let $n \in \mathcal{N}_H$, we have to choose a representative $\rho(n)$ of $n$. We know that there is either a node $n_D \in \mathcal{N}_D$ such that $n = r'(n_D)$, or a node $n_R \in \mathcal{N}_R$ such that $n = m'(n_R)$, or both. We use the notations from theorem 4.6 and its proof.

*(Ω.)* First, in order to prove that $\mathcal{N}_H^\Omega = (\mathcal{N}_G^\Omega - m(\mathcal{N}_L^\Omega)) + \mathcal{N}_R^\Omega$, let $n \in \mathcal{N}_H^\Omega$.

*(Ω, R.)* If there is a node $n_R \in \mathcal{N}_R^\Omega$ such that $n = m'(n_R)$, let us prove that it is unique. Let $n'_R \in \mathcal{N}_R^\Omega$ be another node such that $n = m'(n'_R)$, then $n_R \sim n'_R$. Let us consider a chain with minimal length $2k$ from $n_R$ to $n'_R$; we know from the proof of theorem 4.6 that all the nodes in this chain are labeled. ¿From lemma A.2, $n_0$ and $n_1$ cannot be both in $\mathcal{N}_R$, so that $n_1 \in \mathcal{N}_G^\Omega$, subsequently $k \geq 2$ and $p_1, p_2 \in \mathcal{N}_L^\Omega$ with $n_1 = m(p_1) = m(p_2)$. Since $m$ is injective on $\mathcal{N}_L^\Omega$, from lemma A.2 this cannot occur. So, we have proved that $m'^\Omega : \mathcal{N}_R^\Omega \to \mathcal{N}_H^\Omega$ is injective, and we define $\rho(n) = n_R$.

*(Ω, G.)* If there is no node $n_R \in \mathcal{N}_R^\Omega$ such that $n = m'(n_R)$, then there is a node $n_D \in \mathcal{N}_D^\Omega$ (i.e., $n_D \in \mathcal{N}_G^\Omega$) such that $n = r'(n_D)$. Let us prove that it is unique. Let $n'_D \in \mathcal{N}_G^\Omega$ be another node such that $n = r'(n'_D)$, then $n_D \sim n'_D$. Let us consider a chain with minimal length from $n_D$ to $n'_D$; we know from the proof of theorem 4.6 that all the nodes in this chain are labeled. ¿From lemma A.2, $n_0$ and $n_1$ cannot be both in $\mathcal{N}_D$, so 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 $n = m'(n_R)$. Let $\widetilde{\mathcal{N}_G^\Omega}$ denote the subset of $\mathcal{N}_G^\Omega$ made of the nodes which are not equivalent to any node in $\mathcal{N}_R^\Omega$. We have proved that the restriction of $r'^\Omega : \mathcal{N}_D^\Omega \to \mathcal{N}_H^\Omega$ to $\widetilde{\mathcal{N}_G^\Omega}$ is injective, and we define $\rho(n) = n_D$.

*(Ω, L.)* We still have to prove that $\widetilde{\mathcal{N}_G^\Omega} = \mathcal{N}_G^\Omega - m(\mathcal{N}_L^\Omega)$, i.e., that a node $n_G \in \mathcal{N}_G^\Omega$ is equivalent to a node $n_R \in \mathcal{N}_R^\Omega$ if and only if there is node $p \in \mathcal{N}_L^\Omega$ such that $n_G = m(p)$. Clearly, if $p \in \mathcal{N}_L^\Omega$ and $n_G = m(p)$, then $n_G \sim r(p)$ with $r(p) \in \mathcal{N}_R^\Omega$. Now, let $n_G \sim n_R$ for some $n_G \in \mathcal{N}_G^\Omega$ and $n_R \in \mathcal{N}_R^\Omega$. Let us consider a chain with minimal length $2k$ from $n_R$ to $n_G$; we know that all the nodes in this chain are labeled. If $k > 1$ then $n_1 = d(p_1) = d(p_2)$ with $p_1, p_2 \in \mathcal{N}_L^\Omega$; since the restriction of $d$ to $\mathcal{N}_L^\Omega$ is injective, a contradiction

follows from lemma A.2. Hence $k = 1$, which means the node $p_1 \in \mathcal{N}_L^\Omega$ is such that $n_R = r(p_1)$ and $n_G = m(p_1)$.

This concludes the proof that $\mathcal{N}_H^\Omega = (\mathcal{N}_G^\Omega - m(\mathcal{N}_L^\Omega)) + \mathcal{N}_R^\Omega$.

*($\mathcal{X}$.)* Now, in order to prove that $\mathcal{N}_H^\mathcal{X} = \mathcal{N}_G^\mathcal{X} + (\mathcal{N}_R^\mathcal{X} - r(\mathcal{N}_L^\mathcal{X}))$, let $n \in \mathcal{N}_H^\mathcal{X}$.

*($\mathcal{X}$, G.)* If there is a node $n_D \in \mathcal{N}_D$ such that $n = r'(n_D)$, then $n_D$ is unlabeled, i.e., $n_D \in \mathcal{N}_G^\mathcal{X} + \mathcal{N}_{E'} + \mathcal{N}_{N'}$.

– If $n_D \in \mathcal{N}_G^\mathcal{X}$, let us prove that it is unique. Let $n'_D \in \mathcal{N}_G^\mathcal{X}$ be another node such that $n = r'(n'_D)$, then $n_D \sim n'_D$. Let us consider a chain with minimal length $2k$ from $n_D$ to $n'_D$, with $k \geq 2$ since both $n_D$ and $n'_D$ are in $D$. Then $p_1 \in \mathcal{N}_L^\mathcal{X}$, and $n_1 = r(p_1) \in \mathcal{N}_R^\mathcal{X}$ because $r$ maps $\mathcal{N}_L^\mathcal{X}$ to $\mathcal{N}_R^\mathcal{X}$, and $n_1 = r(p_2)$ with $p_2 \in \mathcal{N}_K^\mathcal{X} = \mathcal{N}_L^\mathcal{X} + \mathcal{N}_E + \mathcal{N}_N$. If $p_2 \in \mathcal{N}_L^\mathcal{X}$, since $r$ is injective on $\mathcal{N}_L^\mathcal{X}$, a contradiction follows from lemma A.2. If $p_2 \in \mathcal{N}_E + \mathcal{N}_N$ then $n_2 = d(p_2) \in \mathcal{N}_{E'} + \mathcal{N}_{N'}$, it is different from $n'_D$ since $\mathcal{N}_G^\mathcal{X}$ is disjoint from $\mathcal{N}_{E'} + \mathcal{N}_{N'}$. So, $2 < k$, and from remark (A.1) this is impossible. So, we have proved that the restriction of $r' : \mathcal{N}_D \to \mathcal{N}_H$ to $\mathcal{N}_G^\mathcal{X}$ is injective, and we define $\rho(n) = n_D$.

– If $n_D \in \mathcal{N}_{E'} + \mathcal{N}_{N'}$, let us prove that there is a node $n_R \in \mathcal{N}_R^\mathcal{X}$ such that $n = m'(n_R)$. Let $p \in \mathcal{N}_E + \mathcal{N}_N$ be such that $n_D = d(p)$, and $n_R \in \mathcal{N}_R$ such that $n_R = r(p)$. Then $n = r'(d(p)) = m'(r(p))$. Since $n$ is unlabeled, $n_R \in \mathcal{N}_R^\mathcal{X}$, as required. This case is considered below.

*($\mathcal{X}$, R.)* It has been proved above that, for every $n \in \mathcal{N}_H^\mathcal{X}$, if there is no node $n_D \in \mathcal{N}_G^\mathcal{X}$ such that $n = r'(n_G)$, then there is a node $n_R \in \mathcal{N}_R^\mathcal{X}$ such that $n = m'(n_R)$.

– Such a node $n_R$ cannot be in $r(\mathcal{N}_L)$: otherwise, let $p \in \mathcal{N}_L$ be such that $n_R = r(p)$, then $n = m'(r(p)) = r'(d(p)) = r'(n_G)$ where $n_G = d(p) \in \mathcal{N}_G^\mathcal{X}$. So, $n_R \in (\mathcal{N}_R^\mathcal{X} - r(n_L^\mathcal{X}))$.

– Let us prove that such a node $n_R$ is unique. Let $n'_R \in (\mathcal{N}_R^\mathcal{X} - r(\mathcal{N}_L^\mathcal{X}))$ be another node such that $n = m'(n'_R)$, then $n_R \sim n'_R$. Let us consider a chain with minimal length $2k$ from $n_R$ to $n'_R$, with $k \geq 2$ since both $n_R$ and $n'_R$ are in $R$. Then $n_R = r(p_1)$ with $p_1 \in \mathcal{N}_K^\mathcal{X} = \mathcal{N}_L^\mathcal{X} + \mathcal{N}_E + \mathcal{N}_N$. If $p_1 \in \mathcal{N}_L^\mathcal{X}$ then $n_R \in r(\mathcal{N}_L^\mathcal{X})$, in contradiction with our assumption. If $p_1 \in \mathcal{N}_E + \mathcal{N}_N$ then $n_1 = d(p_1)$ and $1 < k$, which is impossible from remark (A.1).

So, we have proved that the restriction of $m' : \mathcal{N}_R \to \mathcal{N}_H$ to $\mathcal{N}_R^\mathcal{X} - r(\mathcal{N}_L^\mathcal{X})$ is injective, and we define $\rho(n) = n_R$.

This concludes the proof that $\mathcal{N}_H^\mathcal{X} = \mathcal{N}_G^\mathcal{X} + (\mathcal{N}_R^\mathcal{X} - r(\mathcal{N}_L^\mathcal{X}))$.

□