Adjunction for Garbage Collection with Application to Graph Rewriting*

D. Duval[1], R. Echahed[2], and F. Prost[2]

[1] LJK
B. P. 53, F-38041 Grenoble, France
Dominique.Duval@imag.fr
[2] LIG
46, av Félix Viallet, F-38031 Grenoble, France
Rachid.Echahed@imag.fr, Frederic.Prost@imag.fr

Abstract. We investigate garbage collection of unreachable parts of rooted graphs from a *categorical* point of view. First, we define this task as the right adjoint of an inclusion functor. We also show that garbage collection may be stated via a left adjoint, hence preserving colimits, followed by two right adjoints. These three adjoints cope well with the different phases of a traditional garbage collector. Consequently, our results should naturally help to better formulate graph transformation steps in order to get rid of garbage (unwanted nodes). We illustrate this point on a particular class of graph rewriting systems based on a double pushout approach and featuring edge redirection. Our approach gives a neat rewriting step akin to the one on terms, where garbage never appears in the reduced term.

1 Introduction

Garbage collection has been introduced [5,9] in order to improve the management of memory space dedicated to the run of a process. Such memory can be seen as a rooted graph, where the nodes reachable from the roots represent the memory cells whose content can potentially contribute to the execution of the process, while the unreachable nodes represent the garbage, i.e., the cells that may become allocated at will when memory is required. Several algorithms have been proposed in the literature in order to compute the reachable and unreachable nodes (see for instance [4,8]).

In this paper, we investigate garbage collection from a *categorical* point of view. More precisely, our main purpose is a theoretical definition of the process of calculating the reachable nodes of a rooted graph. This corresponds to removing the garbage, in the sense of calculating the memory space made of the reachable cells, starting from a memory space that may include reachable as well as unreachable cells. We show that this process can be defined as the right

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

F. Baader (Ed.): RTA 2007, LNCS 4533, pp. 122–136, 2007.

adjoint of an inclusion functor. We also propose an alternative definition of this process via a left adjoint, followed by two right adjoints. This second definition is close to the actual *tracing* garbage collection algorithms, which proceed by marking the reachable nodes before sweeping the garbage.

Besides the categorical characterisation of the garbage collection process, which can be considered as a motivation per se, the original motivation of this work comes from the definition of graph rewrite steps based on the double pushout approach [7]. In these frameworks a rewrite step is defined as a span $L \leftarrow K \rightarrow R$ where L, K and R are graphs and the arrows represent graph homomorphisms. Let us consider, for instance, the category of graphs defined in [6] (this category is similar to the category **Gr** of section 3.1 except that graphs are not rooted).

The application of such a rule to a graph G consists in finding a homomorphism (a matching) $m : L \rightarrow G$ and computing the reduced graph H so that the following diagram

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

constitutes a double pushout (some conditions are required to ensure the existence of this double pushout [7]).

A main drawback of this approach is that the reduced graph H may contain unreachable nodes. To illustrate this point, let us consider a simple example. Let $f(x) \rightarrow f(b)$ be a classical term rewrite rule. When it is applied on the term $f(a)$, the resulting term is $f(b)$. However, in the double pushout approach (where terms are viewed as graphs), the reduced graph is not just $f(b)$: it includes also the constant a. Indeed, the term rewrite rule $f(x) \rightarrow f(b)$ is turned into a span $f(x) \leftarrow K_0 \rightarrow f(b)$, where the arrows are graph homomorphims (the content of K_0 does not matter here). When this rule is applied to the graph $f(a)$, according to the double pushout approach, we get the diagram

$$\begin{array}{ccccc} f(x) & \longleftarrow & K_0 & \longrightarrow & f(b) \\ \downarrow & & \downarrow & & \downarrow \\ f(a) & \longleftarrow & D_0 & \longrightarrow & H_0 \end{array}$$

where H_0 contains both terms $f(b)$ and a. Actually, the element a occurs in D_0, because the left-hand side is a pushout; thus, a also occurs in H_0, because the right-hand side is a pushout. In order to get rid of a, and more generally to remove all unreachable nodes from the graph H, we propose to use our categorical approach of garbage collection.

When graph rewrite steps are defined following an algorithmic approach such as [2], garbage is easily incorporated within a rewrite step. Unfortunately, in categorical approaches to graph rewriting, garbage is not easily handled. In section 8 of [1], Banach discussed the problem of garbage in an abstract way. He mainly considered what is called "garbage retention", that is to say, garbage is not removed from a graph, as we do, but it should not participate in the rewriting process. In [3], Van den Broek discussed the problem of generated garbage

in the setting of graph transformation based on single pushout approach. He introduced the notion of proper graphs. Informally, a proper graph is a graph where garbage cannot be reachable from non garbage part. The rewrite relation is defined as a binary relation over proper graphs. That is to say, a rewrite rule can be fired only if the resulting graph is proper. In some sense, Van den Broek performs a kind of garbage retention as Banach does.

Outline of the paper

Rooted graphs are introduced in section 2 in order to model reachable and unreachable parts in data-structures. Garbage collection is presented in terms of right and left adjoints in section 3. A double pushout approach for rooted graphs rewriting is defined in section 4, and garbage collection is incorparated within this rewriting setting. We conclude in section 5. Due to space limitations, proofs have been omitted.

2 Rooted Graphs

Rooted graphs are defined as term graphs [2], together with a subset of nodes called roots. These graphs are intended to model usual data-structures implemented with pointers, such as circular lists. The addition of roots models the fact that, due to pointer redirections, some data may become unreachable.

Definition 1 (Signature). A *signature* Ω is a set of operation symbols such that each operation symbol f in Ω is provided with a natural number $\mathrm{ar}(f)$ called its *arity*.

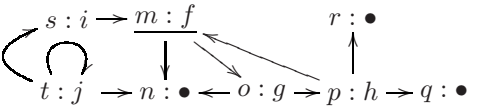
We assume Ω fixed throughout the rest of the paper. Moreover, for each set A, the set of strings over A is denoted A^*, and for each map $f : A \to B$, the map $f^* : A^* \to B^*$ is the extension of f to strings defined by $f^*(a_1 \dots a_n) = f(a_1) \dots f(a_n)$.

Definition 2 (Graph). A *rooted graph* is a tuple $G = (\mathcal{N}_G, \mathcal{N}_G^R, \mathcal{N}_G^\Omega, \mathcal{L}_G, \mathcal{S}_G)$ where $\mathcal{N}_G$ is the set of *nodes* of G, $\mathcal{N}_G^R \subseteq \mathcal{N}_G$ is the set of *roots*, $\mathcal{N}_G^\Omega \subseteq \mathcal{N}_G$ is the set of *labeled nodes*, $\mathcal{L}_G : \mathcal{N}_G^\Omega \to \Omega$ is the *labeling function*, and $\mathcal{S}_G : \mathcal{N}_G^\Omega \to \mathcal{N}_G^*$ is the *successor function* such that, for each labeled node n, the length of the string $\mathcal{S}_G(n)$ is the arity of the operation $\mathcal{L}_G(n)$.

The *arity* of a node n is the arity of its label and the i-th successor of a node n is denoted $\mathrm{succ}_G(n, i)$. 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 *target* is the node $\mathrm{tgt}(n, i) = \mathrm{succ}_G(n, i)$. The set of edges of G is written $\mathcal{E}_G$ The fact that $f = \mathcal{L}_G(n)$ is written $n : f$, an unlabeled node n of G is written $n : \bullet$. Informally, one may think of $\bullet$ as an anonymous variable. The set of unlabeled nodes of G is denoted $\mathcal{N}_G^\mathcal{X}$, so that $\mathcal{N}_G = \mathcal{N}_G^\Omega + \mathcal{N}_G^\mathcal{X}$, where "+" stands for the disjoint union.

Example 1. 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: $[m \mapsto no, o \mapsto np, p \mapsto qrm, s \mapsto m, t \mapsto tsn]$, and roots of G are $\mathcal{N}_G^R = \{m\}$.

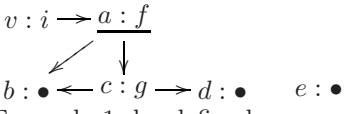
Graphically G is represented as:



Roots of graphs are underlined. Generally in our examples the order of successors is either irrelevant or clear from the context.

Definition 3 (Graph homomorphism). A *rooted graph homomorphism* $\varphi : G \to H$ is a map $\varphi : \mathcal{N}_G \to \mathcal{N}_H$ that preserves the roots, the labeled nodes and the labeling and successor functions, i.e., $\varphi(\mathcal{N}_G^R) \subseteq \mathcal{N}_H^R$, $\varphi(\mathcal{N}_G^{\Omega}) \subseteq \mathcal{N}_H^{\Omega}$, and for each labeled node n, $\mathcal{L}_H(\varphi(n)) = \mathcal{L}_G(n)$ and $\mathcal{S}_H(\varphi(n)) = \varphi^*(\mathcal{S}_G(n))$.
The image $\varphi(n, i)$ of an edge (n, i) of G is defined as the edge $(\varphi(n), i)$ of H.

Example 2. Consider the following graph H:



Let $\varphi : \mathcal{N}_H \to \mathcal{N}_G$, where G is the graph in Example 1, be defined as: $\varphi = [a \mapsto m, b \mapsto n, c \mapsto o, d \mapsto p, e \mapsto p, v \mapsto s]$. Then, φ is a graph homomorphism from H to G.

3 Garbage Collection and Adjunction

A node in a rooted graph is *reachable* if it is a descendant of a root; the unreachable nodes form the *garbage* of the graph. We now address the problem of garbage collection in graphs, in both its aspects: either removing the unreachable nodes or reclaiming them; we also consider the marking of reachable nodes, which constitutes a major step in the so-called *tracing* garbage collection process. We prove in this section that the tracing garbage collection process can be easily expressed in term of adjunctions.

3.1 Garbage Removal Is a Right Adjoint

Rooted graphs and their homomorphisms form the *category of rooted graphs*. From now on, in this paper, the category of rooted graphs is denoted **Gr**, the category of non-rooted graphs is denoted $\mathbf{Gr}_0$, and by "graph" we mean "rooted graph", unless explicitly stated.

Definition 4 (Reachable nodes). The *reachable nodes* of a graph are defined recursively, as follows: a root is reachable, and the successors of a reachable node are reachable nodes.

Example 3. The graph G defined in example 1, has m, n, o, p, q, r as reachable nodes. Nodes s, t are considered as garbage, as well as the edges out of these nodes.

Definition 5 (Reachable graph). A *reachable graph* is a graph where all nodes are reachable.

The reachable graphs and the graph homomorphisms between them form a full subcategory of **Gr**, called the *category of reachable graphs*, **RGr**. Let V denote the inclusion functor: $V : \mathbf{RGr} \to \mathbf{Gr}$.

Definition 6 (Maximal reachable subgraph). The *maximal reachable subgraph of* G is the graph $\Lambda(G)$ such that $\mathcal{N}_{\Lambda(G)}$ is the set of reachable nodes of G, $\mathcal{N}^R_{\Lambda(G)} = \mathcal{N}^R_G$, $\mathcal{N}^\Omega_{\Lambda(G)} = \mathcal{N}_{\Lambda(G)} \cap \mathcal{N}^\Omega_G$, and $\mathcal{L}_{\Lambda(G)}$, $\mathcal{S}_{\Lambda(G)}$ are the restrictions of $\mathcal{L}_G$, $\mathcal{S}_G$ to $\mathcal{N}_{\Lambda(G)}$.

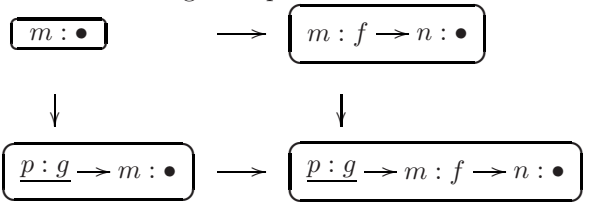
Since a graph homomorphism $\varphi : G \to H$ preserves the roots and the successors, it does also preserve the reachable nodes. Hence, it can be restricted to the maximal reachable subgraphs: $\Lambda(\varphi) : \Lambda(G) \to \Lambda(H)$.

Definition 7 (Garbage removal functor). The *garbage removal* is the functor: $\Lambda : \mathbf{Gr} \to \mathbf{RGr}$ that maps each graph G to its maximal reachable subgraph $\Lambda(G)$ and each graph homomorphism $\varphi : G \to H$ to its restriction $\Lambda(\varphi) : \Lambda(G) \to \Lambda(H)$.

Clearly, the composed functor $\Lambda \circ V$ is the identity of **RGr** : a reachable graph is not modified under garbage removal. Moreover, the next result proves that **RGr** is a *coreflective* full subcategory of **Gr**.

Proposition 1 (Garbage removal is a right adjoint). The garbage removal functor Λ is the right adjoint for the inclusion functor V.

The garbage removal functor $\Lambda : \mathbf{Gr} \to \mathbf{RGr}$ is not a left adjoint. Indeed, a left adjoint does preserve the colimits, whereas the functor Λ does not preserve pushouts, as shown in the following example:



If Λ is applied to this square, the graphs of the upper row will be the empty graphs, whereas the graphs of the lower row will remain unchanged. The obtained diagram is no longer a pushout in category **RGr**, since label f and node $n : \bullet$ appear from nowhere.

3.2 Reachability Marking Is a Left Adjoint

Definition 8 (Marked graph). A *marked graph* is a graph M with a set $\mathcal{N}^\star_M \subseteq \mathcal{N}_M$ of *marked nodes*, such that every root is marked and every successor of a marked node is marked (so that all the reachable nodes of a marked graph are marked but unreachable nodes can be marked). A marked graph homomorphism is a graph homomorphism that preserves the marked nodes.

The marked graphs and their homomorphisms form the *category of marked graphs* $\mathbf{Gr}'$.

Let $\Delta : \mathbf{Gr}' \to \mathbf{Gr}$ denote the underlying functor, that forgets about the marking, and let $\nabla : \mathbf{Gr} \to \mathbf{Gr}'$ denote the functor that generates a marked graph from a graph, by marking all its reachable nodes. The next result is straightforward, since the marking does not modify the underlying graph.

Proposition 2 (Reachability marking is a left adjoint). The reachability marking functor ∇ is the left adjoint for the underlying functor Δ, and this adjunction is such that $\Delta \circ \nabla \cong \mathrm{Id}_{\mathbf{Gr}}$.

Definition 9 (Reachable marked graph). A *reachable marked graph* is a marked graph where all nodes are reachable. So, all the nodes of a reachable marked graph are marked.

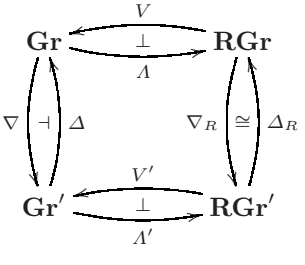
The reachable marked graphs and the marked graph homomorphisms form a full subcategory of $\mathbf{Gr}'$, called the *category of reachable marked graphs*, $\mathbf{RGr}'$. One can note that $\mathbf{RGr}'$ is isomorphic to $\mathbf{RGr}$. We make the distinction between those two cateogries because it enables to give a neat categorical view of how garbage collection is performed. Let V' denote the inclusion functor: $V' : \mathbf{RGr}' \to \mathbf{Gr}'$. As in proposition 1, the inclusion functor V' has a right adjoint: $\Lambda' : \mathbf{Gr}' \to \mathbf{RGr}'$ which is the garbage removal functor for marked graphs.

3.3 Tracing Garbage Collection

A *tracing* garbage collector first determines which nodes are reachable, and then either discards or reclaims all the unreachable nodes. In categorical terms, the fact that reachability marking can be used to perform garbage collection is expressed by theorem 1 and proposition 3 below.

Similarly to the adjunction $\nabla \dashv \Delta$, there is an adjunction $\nabla_R \dashv \Delta_R$ where $\Delta_R : \mathbf{RGr}' \to \mathbf{RGr}$ denotes the underlying functor, that forgets about the marking, and $\nabla_R : \mathbf{RGr} \to \mathbf{RGr}'$ denotes the functor that generates a marked graph from a reachable one, by marking all its nodes. Actually, this adjunction is an isomorphism. $\mathbf{RGr}$ and $\mathbf{RGr}'$ are not identified since they formalize well a phase of a natural garbage collector. Indeed, classical garbage collector begins by suspending the execution of current processes. Then it performs a marking of reachable memory cells (∇). Afterwards, all unmarked cells are deallocated (Λ') and finally the marking is forgotten (Δ_R).

In the following diagram, the four adjunction pairs are represented.



Functors $\nabla \circ V$ and $V' \circ \nabla_R$ are equal, since they both map a reachable graph H to the marked graph made of H with all its nodes marked:

$$\nabla \circ V = V' \circ \nabla_R : \mathbf{RGr} \to \mathbf{Gr}'$$

Theorem 1 below provides a categorical formalization of the tracing garbage removal process, which can be decomposed in three steps: First, the main step freely generates the marked graph $\nabla(G)$ from the given graph G. Then, the reachable marked graph $\Lambda'(\nabla(G))$ is obtained by throwing away the non-marked nodes. Finally, the reachable graph $\Delta_R(\Lambda'(\nabla(G)))$ is obtained by forgetting the marking.
According to theorem 1, $\Delta_R(\Lambda'(\nabla(G)))$ is isomorphic to $\Lambda(G)$.

Theorem 1 (Garbage removal through reachability marking)

$$\Lambda \cong \Delta_R \circ \Lambda' \circ \nabla \ .$$

So, the garbage removal functor Λ can be replaced by $\Delta_R \circ \Lambda' \circ \nabla$. This means that the main step in the tracing garbage removal process is the reachability marking: indeed, the application of Δ_R is "trivial", and the application of Λ' to the image of ∇ simply throws away the unmarked nodes.

In order to express garbage reclaiming, let $\mathcal{N}$ and $\mathcal{N}^\star$ denote the functors from $\mathbf{Gr}'$ to **Set** that map a marked graph to its set of nodes and to its set of marked nodes, respectively. They can be combined into one functor with values in the following category **SubSet**: the objects of *Subset* are the pairs of sets (X, Y) such that $Y \subseteq X$, and a morphism is a pair of maps (f, g) from (X, Y) to (X', Y'), where $f : X \to X'$, $g : Y \to Y'$, and g is the restriction of f.
$(\mathcal{N}, \mathcal{N}^\star) : \mathbf{Gr}' \to \mathbf{SubSet}$
The complement $X \setminus Y$ is the set of unreachable nodes: this is expressed in the next result. Note that the *set complement*, that maps (X, Y) to $X \setminus Y$, cannot reasonably be extended to a functor from **SubSet** to **Set**.

Proposition 3 (Garbage reclaiming through reachability marking)
The composition of ∇ with $(\mathcal{N}, \mathcal{N}^\star)$, followed by the set complement, provides the set of unreachable nodes.

So, the garbage reclaiming can be expressed as $(\mathcal{N}, \mathcal{N}^\star) \circ \nabla$ followed by the set complement. Here also, this means that the main step in the tracing garbage reclaiming process is the reachability marking.

4 Application: Rooted Graph Rewriting with Garbage Removal

In this section we focus on a class of graph rewrite systems dedicated to transform data-structures with pointers [6]. This class has been defined using the double pushout approach [7]. We mainly show how rewrite steps can be enhanced by integrating garbage removal. This integration can be generalized to other rewrite systems based on pushouts, thanks to the fact that left adjoints preserve colimits.

4.1 Disconnections

This section will be used in the left hand side of the double pushout construction. Definitions are adapted from [6], with a more homogeneous presentation.

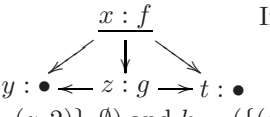
The disconnection of a graph L is made of a graph K and a graph homomorphism $l : K \to L$. Roughly speaking, K is obtained by redirecting some edges of L toward new, unlabeled targets, and the homomorphism l reconnects all the disconnected nodes: $\mathcal{N}_K$ is made of $\mathcal{N}_L$ together with some new, unlabeled nodes, and l is the identity on $\mathcal{N}_L$.

Definition 10 (Disconnection kit). A *disconnection kit* $k = (E_l, N_g, E_g)$ for a graph L, where E_l, E_g are susbsets of $\mathcal{E}_L$ and $N_g \subseteq \mathcal{N}_L$, is made of: (i) a set of edges E_l, called the *locally redirected edges*, (ii) a set of nodes N_g, called the *globally redirected nodes*, and (iii) another set of edges E_g, called the *globally redirected edges*, that is disjoint from E_l and such that the target of every edge in E_g is in N_g. A disconnection kit $(E_l, N_g, \emptyset)$ is simply denoted (E_l, N_g).

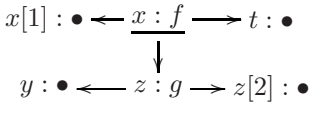
Definition 11 (Disconnection of a graph). Let L be a graph, with a disconnection kit $k = (E_l, N_g, E_g)$. Let K be the graph defined by:

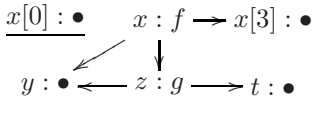
- $\mathcal{N}_K = \mathcal{N}_L + \mathcal{N}_E + \mathcal{N}_N$, where $\mathcal{N}_E$ is made of one new node $n[i]$ for each edge $(n, i) \in E_l$ and $\mathcal{N}_N$ is made of one new node $n[0]$ for each node $n \in N_g$,
- $\mathcal{N}_K^R$ is made of one node for each root n of L: n itself if $n \notin N_g$ and $n[0]$ if $n \in N_g$.
- $\mathcal{N}_K^\Omega = \mathcal{N}_L^\Omega$,
- 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 $(n, i) \notin E_l + E_g$ then $\mathrm{succ}_K(n, i) = \mathrm{succ}_L(n, i)$, if $(n, i) \in E_l$ then $\mathrm{succ}_K(n, i) = n[i]$ and if $(n, i) \in E_g$ then $\mathrm{succ}_K(n, i) = \mathrm{tgt}(n, i)[0]$.

Let $l : \mathcal{N}_K \to \mathcal{N}_L$ be the map defined by: $l(n) = n$ if $n \in \mathcal{N}_L$, $l(n[i]) = \mathrm{succ}_L(n, i)$ if $(n, i) \in E_l$, $l(n[0]) = n$ if $n \in N_g$. Clearly l preserves the roots, the labeled nodes and the labeling and successor functions, so that it is a graph homomorphism. Then $l : K \to L$ is *the disconnection of L with respect to k*.

Example 4. Let L_4 be the following graph: 

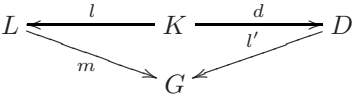
If we respectively consider disconnection kits $k_1 = (\{(x, 1), (z, 2)\}, \emptyset)$ and $k_2 = (\{(x, 3)\}, \{x\})$, we have the following disconnections of L_4, respectively K_4 and K_4':





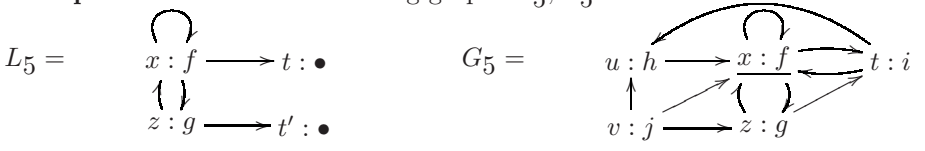
Definition 12 (Matching). Let L be a graph with a disconnection kit $k = (E_l, N_g)$. A *matching of L consistent with k* is a graph homomorphism $m : L \to G$ such that the restriction of m to $(\mathcal{N}_L^\Omega \cup N_g)$ is injective.

Definition 13 (Disconnection of a matching). Let L be a graph, with a disconnection kit $k = (E_l, N_g)$, and let $m : L \to G$ be a matching of L consistent with k. Let $E'_l = m(E_l)$ and $N'_g = m(N_g)$ (since m is a matching, the restrictions of m are bijections: $E_l \cong E'_l$ and $N_g \cong N'_g$). Let E'_g be the set of the edges of $G - m(L)$ with their target in N'_g, and let $k' = (E'_l, N'_g, E'_g)$. Let $l : K \to L$ be the disconnection of L with respect to k, and $l' : D \to G$ the disconnection of G with respect to k'. Let $d : \mathcal{N}_K \to \mathcal{N}_D$ be the map defined by: $d(n) = m(n)$ if $n \in \mathcal{N}_L$, $d(n[i]) = m(n)[i]$ if $n[i] \in \mathcal{N}_E$ and $d(n[0]) = m(n)[0]$ if $n[0] \in \mathcal{N}_N$. Clearly, d is a graph homomorphism. Then the following square in **Gr** is called *the disconnection of m with respect to k*:



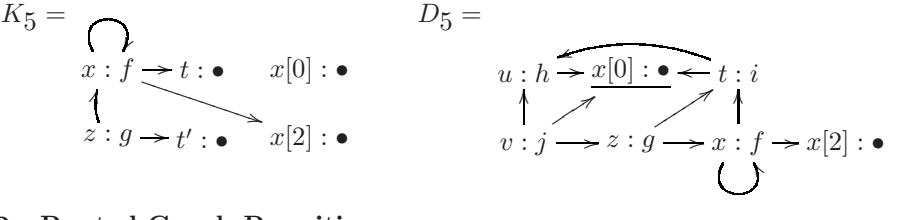
In other words the disconnection of a matching with respect to a disconnection kit consists in the building of, D (and the appropriate morphisms) once K, L, G and m, l are given. Informally D is made of three parts. The first one is the part of G that is not matched. The second part is the image of L in G. Finally there are nodes without labels introduced to perform redirections ($m(n)[i]$ for local redirections, $m(n)[0]$ for global ones).

Example 5. Consider the following graphs L_5, G_5:



and let k be the disconnection kit $(\{(x, 2)\}, \{x\})$, the edge $(x, 2)$ being the edge joining node x to node z.

The graph homomorphism $m_5 = [x \mapsto x, t \mapsto t, t' \mapsto t, z \mapsto z]$, is a morphism from L_5 to G_5. It is also a matching of L_5 consistent with k. Now by disconnection of G_5 with respect to k we have the graphs K_5 which is the disconnection of L_5 with respect to disconnection kit $(\{(x, 2)\}, \{x\})$, and D_5 which is the disconnection of G_5 with respect to disconnection kit $(\{(x, 2)\}, \{x\}, \{(u, 1), (v, 2), (t, 2)\})$:



4.2 Rooted Graph Rewriting

Definition 14 (Rewrite rule). A *rewrite rule*, or *production*, is a span of graphs $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ where l is the disconnection of L with respect to a disconnection kit $k = (E_l, N_g)$, and the restriction of r to $\mathcal{N}_L^{\mathcal{X}}$ is injective and has its values in $\mathcal{N}_R^{\mathcal{X}}$. Then p is a rewrite rule *consistent with k*.

A rewrite step is defined from a rewrite rule $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ and a matching $m : L \rightarrow G$, both with respect to a disconnection kit $k = (E_l, N_g)$ of L. The role of the rewrite step consists in: (i) adding to G an instance of the right-hand side R of p, (ii) performing some local redirections of edges in G: each edge (n, i) in $m(E_l)$ is redirected to the new target $n[i]$, (iii) performing some global redirections of edges in G: all incoming edges of a node n in $m(N_g)$, except the edges in the image of the matching, are redirected to the new target $n[0]$, (iv) modifying the roots of G: if n is a root in G not in $m(N_g)$ then it remains a root, but if n is a root in G and in $m(N_g)$ then $n[0]$ becomes a root instead of n.

As in [6], the basic ingredient in the double pushout approach to graph rewriting is lemma 1 below, about the reflection of pushouts by a faithful functor. The faithful functor in [6] was the node functor $\mathcal{N}$, from the category $\mathbf{Gr}_0$ of non-rooted graphs to the category of sets. Here, it will be the forgetful functor U_0 from the category $\mathbf{Gr}$ of rooted graphs to the category $\mathbf{Gr}_0$ of non-rooted graphs, which clearly is faithful. Since this lemma has not been stated in this form in [6], its proof is given below.

Lemma 1 (Pushout reflection). Let $\Phi : \mathbf{A} \rightarrow \mathbf{A}'$ be a faithful functor. Let $\Sigma = (A_1 \overset{f_1}{\leftarrow} A_0 \overset{f_2}{\rightarrow} A_2)$ be a span and let Γ be a square in $\mathbf{A}$:

$$\begin{array}{ccccc} A_1 & \xleftarrow{f_1} & A_0 & \xrightarrow{f_2} & A_2 \\ & {\scriptstyle g_1}\searrow & & \swarrow{\scriptstyle g_2} & \\ & & A & & \end{array}$$

If $\Phi(\Gamma)$ is a pushout in $\mathbf{A}'$ and if for each cocone Δ on Σ in $\mathbf{A}$, there is a morphism $h : A \rightarrow B$ in $\mathbf{A}$ (where B is the vertex of Δ) such that $\Phi(h)$ is the cofactorisation of $\Phi(\Delta)$ with respect to $\Phi(\Gamma)$, then Γ is a pushout in $\mathbf{A}$.

For each span Σ of sets $(N_1 \overset{\varphi_1}{\leftarrow} N_0 \overset{\varphi_2}{\rightarrow} N_2)$, let $\sim$ denote the equivalence relation induced by Σ on $N_1 + N_2$, which means that it is generated by $\varphi_1(n_0) \sim \varphi_2(n_0)$ for all $n_0 \in N_0$, and let N be the quotient $N = (N_1 + N_2)/\sim$. For $i \in \{1, 2\}$, let $\psi_i : N_i \rightarrow N$ map every node n_i of G_i to its class modulo $\sim$. Then, it is well-known that the following square is a pushout in **Set**, which will be called *canonical*:

$$\begin{array}{ccccc} & & N_0 & & \\ & {\scriptstyle \varphi_1}\swarrow & & \searrow{\scriptstyle \varphi_2} & \\ N_1 & & & & N_2 \\ & {\scriptstyle \psi_1}\searrow & & \swarrow{\scriptstyle \psi_2} & \\ & & N & & \end{array}$$

The notion of "strongly labeled span of graphs" comes from [6], where it was defined for non-rooted graphs, but actually roots are not involved in this notion.

Definition 15 (Strongly labeled span of graphs). A span $\Sigma = (G_1 \overset{\varphi_1}{\leftarrow} G_0 \overset{\varphi_2}{\rightarrow} G_2)$ in $\mathbf{Gr}$ is *strongly labeled* if, as soon as two labeled nodes in $\mathcal{N}(G_1) + \mathcal{N}(G_2)$ are equivalent with respect to Σ, they have the same label and equivalent successors.

Definition 16 (Canonical square of graphs). Let $\Sigma = (G_1 \overset{f_1}{\leftarrow} G_0 \overset{f_2}{\rightarrow} G_2)$ be a strongly labeled span in $\mathbf{Gr}$. The *canonical square on* Σ is the square in $\mathbf{Gr}$:

$$\begin{array}{ccccc} G_1 & \xleftarrow{\varphi_1} & G_0 & \xrightarrow{\varphi_2} & G_2 \\ & {\scriptstyle\psi_1}\searrow & & \swarrow{\scriptstyle\psi_2} & \\ & & G & & \end{array}$$

where the underlying square of nodes is the canonical pushout in **Set**, a node n in G is a root if and only if $n = \psi_i(n_i)$ for a root n_i in G_1 or G_2, a node n in G is labeled if and only if $n = \psi_i(n_i)$ for a labeled node n_i in G_1 or G_2, and moreover the label of n is the label of n_i and the successors of n are the equivalence classes of the successors of n_i.

Clearly, $\Gamma(\Sigma)$ is a commutative square in **Gr**. It is actually a pushout in **Gr** as stated in Theorem 2. The next result is proved in [6].

Lemma 2 (Pushout of non-rooted graphs). Let Σ be a strongly labeled span in **Gr**, and let Γ be the canonical square on Σ. Then $U_0(\Gamma)$ is a pushout in $\mathbf{Gr}_0$.

Theorem 2 (Pushout of rooted graphs). Let Σ be a strongly labeled span in **Gr**, and let Γ be the canonical square on Σ. Then Γ is a pushout in **Gr**.

It is easy to see that a disconnection square is the canonical square on a strongly labeled span, so that the next result follows from theorem 2.

Theorem 3 (A pushout complement). Let L be a graph with a disconnection kit k and let m be a matching of L consistent with k. The disconnection square of m with respect to k is a pushout in the category of graphs.

Theorem 3 means that d and l' form a *complement pushout* to l and m. Other complement pushouts to l and m can be obtained by replacing E'_g, in definition 13, by any of its subsets.

The next result is not so easy, its proof can be found in [6] (except for the property of the roots, which is clear).

Theorem 4 (A direct pushout). Let p be a rewrite rule $(L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ and $m : L \rightarrow G$ a matching, both consistent with a disconnection kit k of L. Then the span $D \overset{d}{\leftarrow} K \overset{r}{\rightarrow} R$ is strongly labeled, so that the canonical square on it is a pushout.

Definition 17 (Rewrite step). Let p be a rewrite rule $(L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ and $m : L \rightarrow G$ a matching, both consistent with a disconnection kit k of L. Then G *rewrites to* H *using rule* p if there is a diagram:

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

where the left hand side of the diagram is the disconnection of m with respect to k and the right hand side is a canonical square.

So, according to theorems 3 and 4, a rewrite step corresponds to a *double pushout* in the category of graphs.

Proposition 1 (A description of the nodes). *With the notations and assumptions of definition 17, the representatives of the equivalence classes of nodes of* $\mathcal{N}_R + \mathcal{N}_D$ *can be chosen in such a way that:* $\mathcal{N}_H^\Omega = (\mathcal{N}_G^\Omega - m(\mathcal{N}_L^\Omega)) + \mathcal{N}_R^\Omega$ *and* $\mathcal{N}_H^\mathcal{X} = \mathcal{N}_G^\mathcal{X} + (\mathcal{N}_R^\mathcal{X} - r(\mathcal{N}_L^\mathcal{X}))$ *and* $\mathcal{N}_H^R = r'(\mathcal{N}_D^R) \cup m'(\mathcal{N}_R^R)$.

4.3 Graph Rewriting with Garbage removal

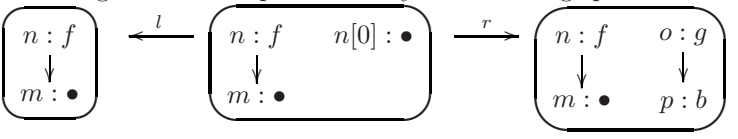
In this section we give a direct application of garbage removal via a left adjoint in rooted graph rewriting. Indeed, left adjoints preserve pushouts. Therefore it is possible to apply functor ∇ on the double push out. Then the composition of $\Delta_R \circ \Lambda'$ gives us the garbage free reduced graph. This schema applies to every double pushout settings, we illustrate this on a particular one.

Definition 18 (Rewrite step with garbage removal). Let p be a rewrite rule $(L \xleftarrow{l} K \xrightarrow{r} R)$ and $m : L \to G$ a matching, both consistent with a disconnection kit k of L. Then G *rewrites with garbage removal to* P *using rule* p if there is a diagram:

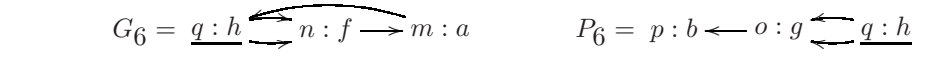
$$\begin{array}{ccccc} L & \xleftarrow{\ l\ } & K & \xrightarrow{\ r\ } & R \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle d} & & \downarrow{\scriptstyle m'} \\ G & \xleftarrow{\ l'\ } & D & \xrightarrow{\ r'\ } & H \end{array}$$

where the left hand side is the disconnection of m with respect to k and the right hand side is a canonical square, and $P = V(\Delta_R(\Lambda'(\nabla(H)))) = V(\Lambda(H))$.

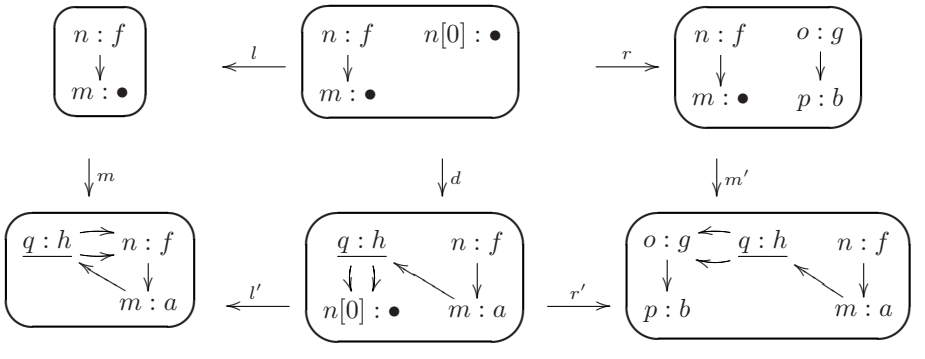
Example 6. First, let us simulate a term rewrite rule. Consider the rule $f(x) \to g(b)$. In our setting it can be implemented by the following span s:



Where l, r are the expected graph homomorphisms with $l(n[0]) = n$ and $r(n[0]) = o$. Then G_6 rewrites to P_6 by using the rule s



Indeed one has the following double pushout in **Gr**:

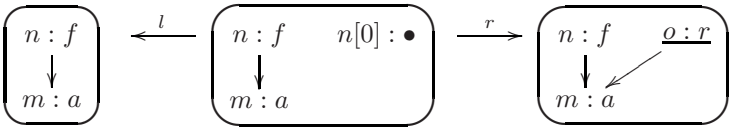


In this example $f(a)$ becomes unreachable because of edge redirection. Let H_6 be the graph at the bottom right of this double pushout. Then, $\Lambda(H_6)$ is the marked graph P_6, as above.

The following graph illustrates well the role played by roots in garbage removal. Consider G'_6 where the root is now n, it rewrites to P'_6 with:

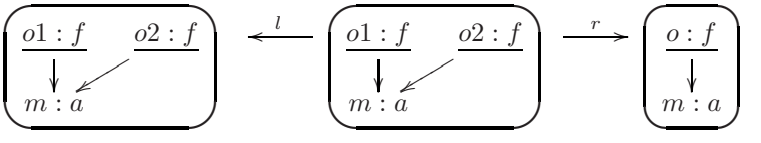
This simple example is not possible to simulate in [6] where garbage cannot be removed.

Another interesting property of graph rewriting with garbage removal is the management of roots. New roots can be introduced by simple rules like:



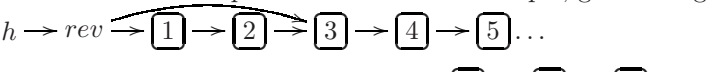
Where $r(n[0]) = o$ and $l(n[0]) = n$. This rule adds a new root $o : r$.

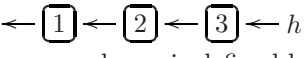
Dually, the number of roots can be reduced, in special circumstances: this can be done by associating two roots equally labeled. For instance consider the span:



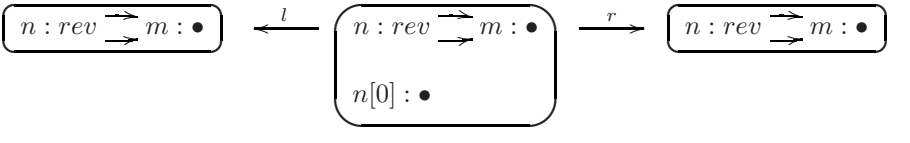
where $r(o1) = r(o2) = o$. Note that by injectivity hypothesis of matching on labeled nodes, the left hand side of the span must match two different roots. Note also that injectivity hypothesis on morphism r only applies to unlabeled nodes, thus $o1, o2$ can be collapsed to a single node o.

Example 7. Let us now consider a more complicated example: the in-place reversal of a list between two particular cells. For example, given the graph:



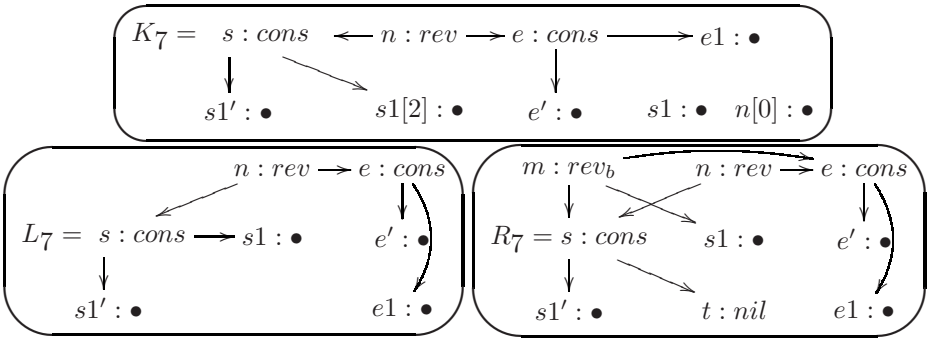
we want to produce the following graph: 

Potentially, the rest of the graph should be removed. rev is defined by means of four rules. Moreover, one can notice that the programmer does not need to take a particular care of garbage management: it is automatically managed. The first rule is for trivial cases (when the first and last items are equal) and is implemented as follows:



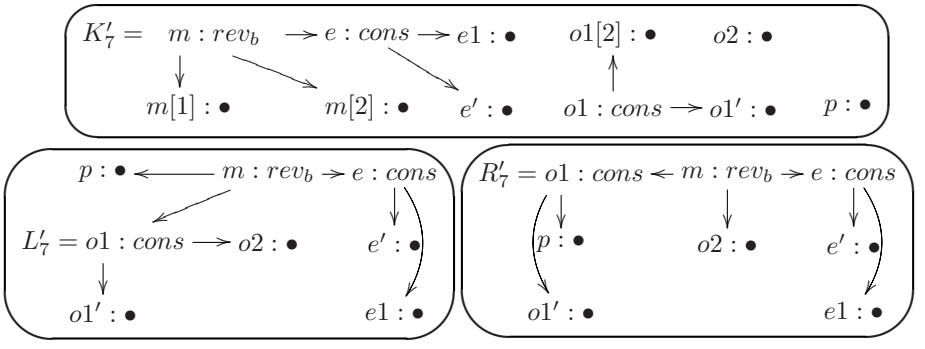
This rule only performs a global redirection from n to m ($r(n[0]) = m$ and $l(n[0]) = n$), thus node $n : rev$ will be garbage removed after the rewrite step (nothing can no longer points to it because of the global redirection).

The second rule classically introduces an auxiliary function rev_b of three parameters which performs the actual rewriting of the list. The first two parameters of rev_b record the pair of list cells to be inverted (current and preceding cells) and the last parameter stores the halting cell. It is done by the span $L_7 \leftarrow K_7 \rightarrow R_7$ where:



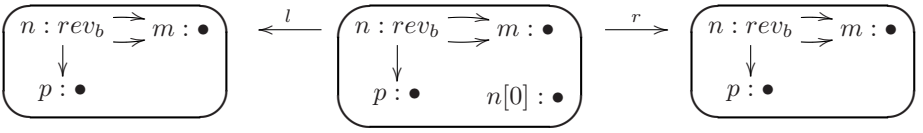
where $n[0] : \bullet$ mapped to n on L_7 and to m on R_7, $s1[2]$ is mapped to t in R_7.

The general step of rev_b is given by the span $L'_7 \leftarrow K'_7 \rightarrow R'_7$, where:



where $m[1], m[2]$ are respectively mapped to $o1, o2$ and $o1[2]$ is mapped to p in R'_7. This rule disconnects the first two parameters of rev_b ($m[1], m[2]$) to make them progress along the list (p is replaced by $o1$ and $o1$ by $o2$). It also redirects the local edge of the list to be reversed (second edge from $o1$) to the previous cell of the list (node p). One has to remember the injectivity hypothesis of the matching homomorphism on labeled nodes. It ensures that nodes $o1, e$ are actually different nodes. Thus there can be no confusion with the halt case.

The halt case for rev_b is similar to the one on rev and just amounts to a global redirection, namely:



where $n[0]$ is mapped to m on the graph on the right side.

We let the reader check on examples that these rules implement the in-situ list reversal between two given nodes.

5 Conclusion

We have presented a categorical approach to garbage collection and garbage removal which can be applied to various graph rewriting frameworks, especially the ones based on pushouts. Garbage removal may be seen either as a right adjoint or as a left adjoint. The right adjoint is the mathematical translation of the description of what is garbage collection: the removal of unreachable parts. On the other hand the left adjoint gives an operational point of view. It illustrates well the three basic steps of any real garbage collector in programming languages: when a garbage collector starts there is first a propagation phase to compute the live parts, then follows the removal of non live nodes and finally the halt of the garbage collector (the return to a normal evaluation mode). Those three steps are represented by three associated functors.

As a future work, we plan to use graph transformation frameworks in order to model memory, as well as mutable objects, transformation.

References

1. Banach, R.: Term graph rewriting and garbage collection using opfibrations. Theoretical Computer Science 131, 29–94 (1994)
2. Barendregt, H., van Eekelen, M., Glauert, J., Kenneway, R., Plasmeijer, M.J., Sleep, M.: Term graph rewriting. In: de Bakker, J.W., Nijman, A.J., Treleaven, P.C. (eds.) PARLE'87. LNCS, vol. 259, pp. 141–158. Springer, Heidelberg (1987)
3. Broek, P.V.D.: Algebraic graph rewriting using a single pushout. In: Abramsky, S. (ed.) TAPSOFT'91: Proceedings of the International Joint Conference on Theory and Practice of Software Development. LNCS, vol. 493, pp. 90–102. Springer, Heidelberg (1991)
4. Cohen, J.: Garbage collection of linked data structures. Computing Surveys 13(3), 341–367 (1981)
5. Collins, G.: A method for overlapping and erasure of lists. Communication of the ACM 3(12), 655–657 (1960)
6. Duval, D., Echahed, R., Prost, F.: Modeling pointer redirection as cyclic term graph rewriting. In: TERMGRAPH 06 (2006) (Extended version to appear in ENTCS)
7. Ehrig, H., Pfender, M., Schneider, H.J.: Graph-grammars: An algebraic approach. In: FOCS 1973, pp. 167–180 (1973)
8. Jones, R.E., Lins, R.: Garbage Collection: Algorithms for Automatic Dynamic Memory Management. J. Wiley & Son, New York (1996)
9. McCarthy, J.: Recursive functions of symbolic expressions and their computation by machine-i. Communication of the ACM 3(1), 184–195 (1960)