The Contractum in Algebraic Graph Rewriting

R. Banach[1]

Computer Science Department, Manchester University,

Manchester, M13 9PL, U.K.

Abstract

Algebraic graph rewriting, which works by first removing the part of the graph to be regarded as garbage, and then gluing in the new part of the graph, is contrasted with term graph rewriting, which works by first gluing in the new part of the graph (the contractum) and performing redirections, and then removing garbage. It is shown that in the algebraic framework these two strategies can be reconciled. This is done by finding a natural analogue of the contractum in the algebraic framework, which requires the reformulation of the customary "double pushout" construction. The new and old algebraic constructions coexist within a pushout cube. In this, the usual "outward" form of the double pushout appears as the two rear squares, and the alternative "inward" formulation as the two front squares. The two formulations are entirely equivalent in the world of algebraic graph rewriting. An application illustrating the efficacy of the new approach to the preservation of acyclicity in graph rewriting is given.

1 Introduction

Algebraic graph rewriting has a relatively long history and forms a mature body of knowledge with applications in many areas of computer science. Both the applications and the theory continue to expand in many directions. From the large literature on the subject we might mention Ehrig (1979, 1986), and Ehrig et al. (1990). See also T.C.S. (1993).

Term graph rewriting arose rather more recently, (Barendregt et al. (1987)), and its application is typically rather more focussed, principally at intermediate and lower level descriptions of implementations of functional languages and similar systems; though the amount of work in related areas is expanding. See eg. Banach (1994), and Sleep et al. (1993).

Algebraic graph rewriting works by the well known "double pushout" construction. In this construction, the first step of a rewrite, once a redex has been located, is to remove the part of the graph that is to be garbaged by the rewrite, leaving a suitable hole. Then the new part of the graph is glued into the hole, yielding the result. In term graph rewriting by contrast, the first step of a rewrite, once the redex has been located, is to glue into the graph some new structure, called the contractum; then to change the shape of the graph by redirecting arcs. Finally, the garbage is removed.

Thus one goes about things in the opposite order in the two models of rewriting; and so one question of interest, is whether there is any relationship between the two approaches. Now the algebraic approach has been used to address some of the problems of direct interest to the term graph rewriting community, (Habel et al. (1988), Hoffman and Plump (1988), Plump (1993)), so one might speculate that the two approaches are not so far apart.

1. Email: rbanach@cs.man.ac.uk

The aim of this paper is to show that the strategy of the term graph rewriting approach can be used to reformulate the algebraic approach into a construction entirely equivalent to the original double pushout construction, but having much of the superficial appearance of the term graph rewriting construction. In particular, the new construction allows a precise notion of contractum and of contractum building to be formulated within the algebraic graph rewriting world.

The rest of this paper is as follows. Section 2 reviews the details of the conventional double pushout construction for a suitable class of graphs. Section 3 describes term graph rewriting and highlights the contrast between it and the algebraic approach. Section 4 gives the new construction in the algebraic world, shows that it is entirely equivalent to the original construction, and argues that it displays the features required for it to be regarded as incorporating a convincing analogue of the term graph contractum concept. Section 5 presents a simple example of the new approach, while section 6 presents an application of the approach by proving a theorem on the preservation of acyclicity in the rewriting of directed graphs, which would have been somewhat more inconvenient to establish in the conventional approach. Section 7 concludes.

2 Algebraic Graph Rewriting

Algebraic graph rewriting originated as a way of manipulating the objects in a specific category of graphs; one whose objects have coloured nodes and coloured edges, with source and target functions mapping each edge to its source and target. However the underlying algebraic construction is very general and can be adapted to many other categories of graph-like systems (see Ehrig et al. (1991a,b, 1993)). Since the main point that this paper makes is algebraic in nature, it too can be adapted to many such categories. However, rather than seek the greatest possible generality in the presentation, by heavy use of universal algebra, we will pick a fairly simple category of graphs to work with, and the reader will be quickly able to construct the appropriate generalisations as required.

Let DG be the category of directed graphs and graph morphisms. An object G of DG is a pair $\langle N_G, A_G \rangle$ where N_G is a set of nodes and $A_G \subseteq N_G \times N_G$ is a set of arcs built from N_G, i.e. a set of ordered pairs of N_G. An arrow $g : G \to H$ of DG is a map $g : N_G \to N_H$ such that

(x, y) is an arc of $G \Rightarrow (g(x), g(y))$ is an arc of H.

Like many categories of graph-like systems, DG has all pushouts. Thus if $f : K \to X$ and $g : K \to Y$ are two arrows, their pushout is the graph $P = \langle N_P, A_P \rangle$ given by:

$N_P = N_X \uplus N_Y / \approx$ where $\uplus$ is disjoint union, and $\approx$ is the smallest equivalence relation such that $x \approx y$ if there is a $k \in N_K$ such that $x = f(k)$ and $y = g(k)$.

$A_P = \{([x]_P, [y]_P) \mid \exists\, u \in [x]_P, v \in [y]_P$ such that $[(u, v) \in A_X$ or $(u, v) \in A_Y\}$ where we have not distinguished between $u \in N_X$ and the tagged version of $u \in N_P$.

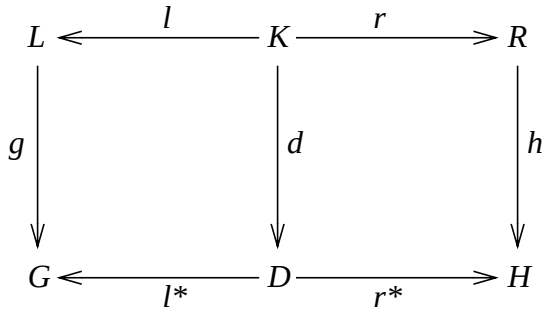
And the arrows $f^* : Y \to P$ and $g^* : X \to P$ are obvious.

Algebraic graph rewriting is given by the double pushout construction. Rules are given by a pair of arrows in DG

$$L \xleftarrow{l} K \xrightarrow{r} R$$

with $l : K \to L$ injective. (For categories of graph-like systems, there is normally a natural notion of injectivity that is used; in our case it is ordinary set-theoretic injectivity). A redex for a rule

$L \leftarrow K \rightarrow R$ is an arrow $g : L \rightarrow G$, and the rewrite proceeds by constructing the diagram below where both squares are pushouts.



The construction is a two stage process.

Intuitively, the first stage of the construction removes the g image of L from G, except for the $g \circ l$ image of K, which provdes the interface for the second stage. In the second stage, a copy of R is glued into the "hole" left behind by the first stage; the edge of the hole being the aforementioned $g \circ l(K)$.

The first stage attempts to construct the object D and the arrows $d : K \rightarrow D$, $l^* : D \rightarrow G$, such that the left square is a pushout. D is known as the pushout complement and is not guaranteed to exist even if (as is the case here) DG has all pushouts. It is standard lore in algebraic graph rewriting theory that a unique smallest pushout complement exists if

(INJ-O) $l : K \rightarrow L$ is injective.

(IDENT-O) $\{x, y\} \subseteq N_L$ and $g(x) = g(y) \Rightarrow [\ x = y$, or $\{x, y\} \subseteq l(N_K)\ ]$.

(DANGL-O) $(x, y) \in A_G - g(A_L)$, and $\{x, y\} \cap g(N_L) \neq \emptyset \Rightarrow \{x, y\} \cap g(N_L) \subseteq g(l(N_K))$.

(INJ-O), which we have assumed already, ensures that a pushout complement with unique $d(K)$ exists if one exists at all. (IDENT-O) ensures that the pushout of l and d is in fact G, by ensuring that the pushout is never forced to try to map distinct nodes of L into the same node of G, other than as instructed by d — something the pushout definition above can never accomplish. (DANGL-O) ensures that D is actually an object of DG, so that when the g image of $(L - l(K))$ is removed from G, no arc is left dangling without a source or target node. These remarks make a little more sense when we see the explicit construction of D.

$$N_D = N_G - g(N_L - l(N_K)),$$

$$A_D = A_G - g(A_L - l(A_K)),$$

The arrow $d : K \rightarrow D$ is given by

$$d : K \rightarrow D : x \mapsto g(l(x))$$

with the obvious extension to arcs. Arrow $l^* : D \rightarrow G$ is just the inclusion on $N_G - g(N_L - l(N_K))$, again with the obvious extension to arcs.

3 Term Graph Rewriting

Just as most applications of algebraic graph rewriting use categories of objects with a richer structure than DG, so too with term graph rewriting, where normally, the category is that of term graphs, i.e. graphs consisting of nodes and arcs, where the nodes are labelled by the symbols from some alphabet, and the out-arcs of each node are labelled by consecutive positive integers $[1 \ldots n]$, each node having an arity as in term rewriting. Other markings may adorn the nodes and arcs depending on the application.

We will however continue to work with DG, which contains (almost) enough structure to enable us to achieve our algebraic objectives for term graph rewriting, albeit in a more austere setting.

In fact we will work with the category $\mathsf{DG}^{(*)}$, whose objects and arrows are those of DG, except that each non-empty object G, optionally has a distinguished node, the root of G, root_G. In fact DG occurs as a full subcategory of $\mathsf{DG}^{(*)}$. Each object G of DG occurs both "as is" in $\mathsf{DG}^{(*)}$, and also in a collection of objects with roots, once for each choice of root from N_G. We can write such objects as (G, root_G) when we want to highlight the root, writing (G, ε) if we want to emphasise that G does not have a root.

An arrow $g : G \to H$ in $\mathsf{DG}^{(*)}$ is like an arrow in DG except that if G has a root root_G, then H must have one, root_H, and we must have

$$g(\mathrm{root}_G) = \mathrm{root}_H.$$

Under these circumstances, readers can check that $\mathsf{DG}^{(*)}$ has all pushouts of $f : K \to X$ and $g : K \to Y$ unless if X and Y both have roots, root_X and root_Y, and root_X, root_Y do not both occur in the same equivalence class in the usual formula for the set-theoretic pushout of $f : N_K \to N_X$ and $g : N_K \to N_Y$. This is more than adequate for our needs in the rewriting construction.

A rule Q is now given by a pair $\langle incl : L \to P, Red \rangle$. The first component is the inclusion of an object L of $\mathsf{DG}^{(*)}$ into another object P. Neither L nor P may have a root. Red is a set of pairs $\langle x, y \rangle$ of nodes, such that $x \in N_L$ and $y \in N_P$.

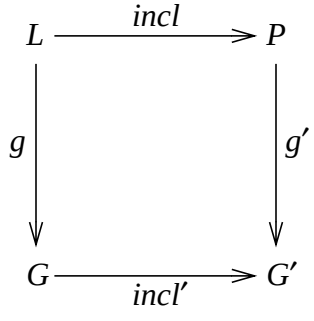
A redex for a rule $Q = \langle incl : L \to P, Red \rangle$ is an arrow $g : L \to (G, \mathrm{root}_G)$ where G must have a root, root_G, except that we must have

(LIVE) Each node $g(x)$, (and arc $(g(x), g(y))$) occurring in the image of a redex $g : L \to (G, \mathrm{root}_G)$ is accessible from root_G.

When we say that x is accessible from r, we mean of course that there is a directed path from r to x in the graph.

Rewriting is a three stage process. Intuitively, the first stage of a rewrite glues a copy of P into G along L. This is just an honest pushout of g and $incl$ which always exists by our remarks above. The second phase, redirection, takes all in-arcs of nodes $g(x)$ where $\langle x, y \rangle \in Red$, and redirects them so that they become in-arcs of $g'(y)$ (where g' is the extension of g provided by the pushout of the first stage). Having done this, the third phase removes everything not accessible from the root, completing the rewrite.

In more detail, stage one constructs the following pushout, whose existence is unproblematic in $\mathsf{DG}^{(*)}$, since of the three graphs involved in $incl$ and g, only G has a root. Obviously G' has a root, such that $incl'(\mathrm{root}_G) = \mathrm{root}_{G'}$.



$P - L$, which generally contains dangling arcs, is called the contractum of the rule, and the pushout construction just mentioned, is called contractum building, as up to isomorphism, the pushout is just the process of gluing a copy of the contractum into G. This paper is mainly concerned with finding an analogue of this process in the algebraic world.

The second stage requires a further condition to hold. Let

$$Red' = \{\langle g'(x), g'(y)\rangle \mid \langle x, y\rangle \in Red\}$$

The condition is that Red' is the (set-theoretic) graph of a function.

(FUNC) $\quad \langle x', y'\rangle \in Red'$ and $\langle x', z'\rangle \in Red' \Rightarrow y' = z'$

Assuming (FUNC) holds, it makes unambiguous sense to redirect all in-arcs of LHS members of Red', and make them point to the corresponding RHS nodes. This gives a graph G''.

$$N_{G''} = N_{G'},$$
$$A_{G''} = (A_{G'} - A^L{}_{Red'}) \cup A^R{}_{Red'},$$
$$\text{root}_{G''} = \text{If } \langle \text{root}_{G'}, y\rangle \in Red' \text{ for some } y \in N_{G'} \text{ then } y \text{ else root}_{G'}$$

where

$$A^L{}_{Red'} = \{(t, g'(x)) \in A_{G'} \mid \text{for some } g'(y), \text{ there is a } \langle g'(x), g'(y)\rangle \in Red'\},$$

$$A^R{}_{Red'} = \{(t, g'(y)) \mid \text{there is a } (t, g'(x)) \in A^L{}_{Red'} \text{ and } \langle g'(x), g'(y)\rangle \in Red'\}.$$

Note that where an arc $(t, g'(x))$ in G' is redirected to $(t, g'(y))$ and there was already a $(t, g'(y))$ arc in G', the two become one arc in G''. (This is at variance with the usual situation in term graph rewriting.) Note also that, unlike in algebraic graph rewriting, where the only nodes and arcs of G manipulated by the rewrite are in the redex, there is no (DANGL)-like condition to prevent the node t in an arc $(t, g'(x))$ which is to be redirected, from being outside $g'(L)$. This is because the removal of arcs and introduction of new ones implicit in redirection, do not involve any removal of nodes, the only origin of any threat of dangling arcs.

Thus far, rewriting can only increase the size of a graph. To enable graphs to shrink, i.e. for rewriting to be able to garbage collect, the third stage defines the graph H by

$$N_H = \{x \in N_{G''} \mid x \text{ is accessible from root}_{G''}\},$$

$$A_H = \{(x, y) \in A_{G''} \mid \{x, y\} \subseteq N_H\},$$

$$\text{root}_H = \text{root}_{G''}.$$

Thus the third, or garbage collection stage, discards anything not accessible from the root of G''. H is the result of the rewrite. Note that H is such that any redex $h : M \to H$ for the first stage of the next rewrite automatically satisfies (LIVE).

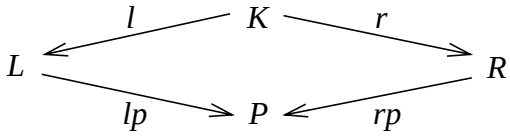
It is worth noting at this point that whereas garbage collection is a purely local phenomenon in algebraic graph rewriting — the garbage is collected during the construction of the pushout complement, in term graph rewriting garbage collection is a global phenomenon — being defined by a condition over the whole of G''.

4 The Algebraic Contractum and the Pushout Cube

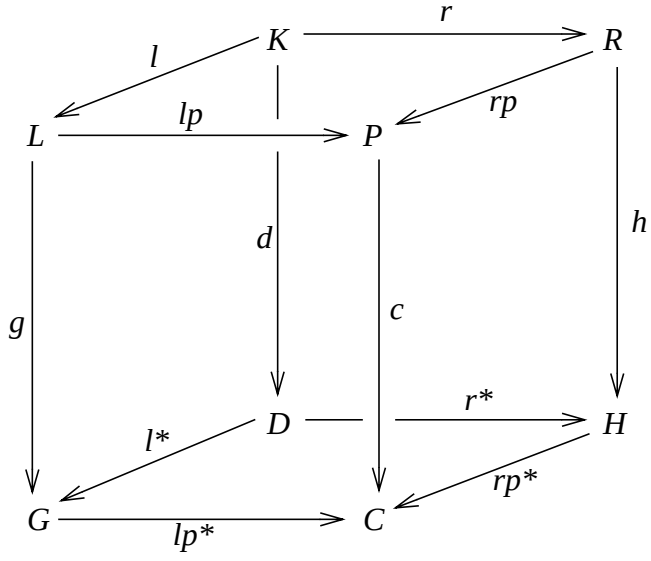
The basic differences between algebraic graph rewriting and term graph rewriting should now be clear. The former collects garbage first, and then replaces it with the new stuff, while the latter glues in the new stuff first, and only after redirection does the garbage get collected.

To bring the two styles of rewriting closer together, we recast algebraic graph rewriting into a form where the basic sequence of steps conforms more closely to that in term graph rewriting. Essentially we point out how contractum building can be done in the algebraic style.

To do so we employ a simple trick. Let $L \xleftarrow{l} K \xrightarrow{r} R$ be an algebraic rule. It consists of two arrows of DG with common domain K. Therefore we can form the pushout



In brief, we show that we can reformulate conventional algebraic graph rewriting using rules of the form $L \xleftarrow{l} K \xrightarrow{r} R$, into a new construction, using rules of the form $L \xrightarrow{lp} P \xleftarrow{rp} R$, and that this new form embodies a credible version of contractum building as the first stage of the rewriting process, allowing a closer comparison with term graph rewriting. We will call the original form of algebraic rules and the rewriting construction that goes with them, the outward form, and the new form and construction, the inward form. Both are named after the direction of the horizontal arrows. The whole thing turns on the construction of the following pushout cube.



In this cube, the colimit of $l : K \to L$, $r : K \to R$ and $d : K \to D$, in which all squares are pushouts, we see the conventional construction in the two rear faces, while the new construction will emerge as the two front faces. In each case we start with G, construct an intermediate graph (either D or C) and then finally construct H.

Since we work from left to right through the cube in both cases, the first stage of the inward form will be an honest pushout of the redex $g : L \to G$, and of the LHS branch of the inward rule $lp : L \to P$. This is the algebraic equivalent of contractum building, comparable to the first stage in term graph rewriting. As in section 3, we can call $P - lp(P)$ which in general will contain dangling arcs, the contractum of the rule; and the graph C constructed by the pushout, is the analogue of the graph G' in term graph rewriting.

After this "contractum building" the inward form of the algebraic rule forms a pushout complement of $c : P \to C$ and $rp : R \to P$, to give the result of the rewrite H. The conditions for this to work, are similar to those needed in constructing D in the outward form of the rule.

Now we turn to the technical details of the new construction. Because DG has small colimits, up to isomorphism, the pushout cube given above really does commute as required. A particular consequence of this is that the choice of unique smallest pushout complement in the outward form of rewriting corresponds to a similar choice of unique smallest pushout complement in the inward form. The main facts about inward and outward rewriting are the following.

Theorem 4.1 Inward and outward rewriting are dual in the following sense. Let $g : L \to G$ be an arrow of DG, serving as redex. Then statement (I) below which ensures the existence of an outward rewrite, and statement (II) below which ensures the existence of an inward rewrite, are equivalent.

(I) There is an outward rule $l : K \to L$, $r : K \to R$ satisfying

(INJ-O) $l : K \to L$ is injective.

(IDENT-O) $\{x, y\} \subseteq N_L$ and $g(x) = g(y) \Rightarrow [\ x = y$, or $\{x, y\} \subseteq l(N_K)\ ]$.

(DANGL-O) $(x, y) \in A_G - g(A_L)$, and $\{x, y\} \cap g(N_L) \neq \emptyset \Rightarrow \{x, y\} \cap g(N_L) \subseteq g(l(N_K))$.

(II) There is an inward rule $lp : L \to P$, $rp : R \to P$ satisfying

(SURJ-I) $P = \langle N_P, A_P \rangle = \langle lp(N_L) \cup rp(N_R),\ lp(A_L) \cup rp(A_R) \rangle$.

(INJ-I) $rp : R \to P$ is injective; $lp : L \to P$ is injective on $L - lp^{-1}(rp(R))$.

(IDENT-I) $\{x, y\} \subseteq N_L$ and $g(x) = g(y) \Rightarrow [\ x = y$, or $\{lp(x), lp(y)\} \subseteq rp(N_R)\ ]$.

(DANGL-I) $(x, y) \in A_G - g(A_L)$, and $\{x, y\} \cap g(N_L) \neq \emptyset \Rightarrow$
$\{x, y\} \cap g(N_L) \subseteq g(lp^{-1}(rp(N_R)))$.

Proof sketch. The theorem claims that statement (II) is sufficient to guarantee that an inward rewrite of $g : L \to G$ exists. Since DG has all pushouts, we merely need to check that the analogous conditions for the pushout complement of $rp : R \to P$ and $c : P \to C$ hold. These are:

(INJ-I)$^{\mathrm{C}}$ $rp : R \to P$ is injective.

(IDENT-I)$^{\mathrm{C}}$ $\{x, y\} \subseteq N_P$ and $c(x) = c(y) \Rightarrow [\ x = y$, or $\{x, y\} \subseteq rp(N_R)\ ]$.

(DANGL-I)$^{\mathrm{C}}$ $(x, y) \in A_C - c(A_P)$, and $\{x, y\} \cap c(N_P) \neq \emptyset \Rightarrow \{x, y\} \cap c(A_P) \subseteq c(rp(N_R))$.

Clearly (INJ-I) $\Rightarrow$ (INJ-I)$^{\mathrm{C}}$. For (IDENT-I) $\Rightarrow$ (IDENT-I)$^{\mathrm{C}}$ we pull a collection of elements in G which witness $c(x) = c(y)$ up along g and use (IDENT-I) to get the result. That (DANGL-I) $\Rightarrow$ (DANGL-I)$^{\mathrm{C}}$, is a relatively straightforward diagram chase.

Thus the pushout complement of $rp : R \to P$ and $c : P \to C$ exists under the conditions stated. These conditions turn out to be necessary as well as sufficient, so we have a set of conditions for the existence of inward rewrites, expressed soley in terms of the redex and the arrows in the rule. The remainder of the argument is as follows.

(I) $\Rightarrow$ (II). Suppose we have the hypotheses of (I). Form the pushout of $l : K \to L$, $r : K \to R$ giving $lp : L \to P$, $rp : R \to P$. Then (SURJ-I) is immediate, and (INJ-I) follows from (INJ-O). Also (IDENT-O) $\Rightarrow$ (IDENT-I) and (DANGL-O) $\Rightarrow$ (DANGL-I) by easy diagram chases.

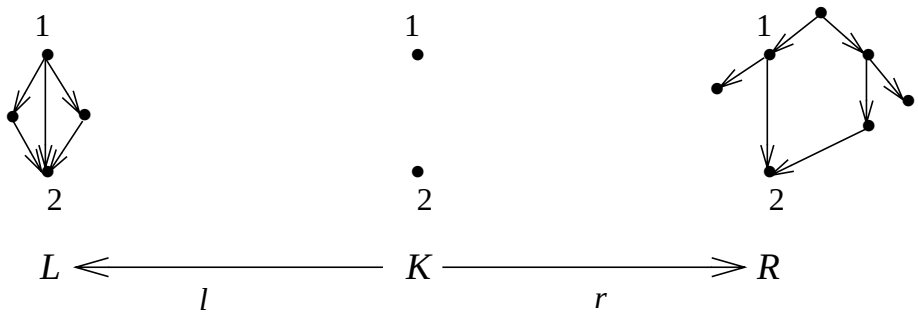
(II) $\Rightarrow$ (I). Suppose we have the hypotheses of (II). We need to construct the top pushout square "in reverse". It follows from work of Ehrig and Kreowski (1979) that (SURJ-I) and (INJ-I) are sufficient for this to be done in an essentially unique way so we get $l : K \to L$, $r : K \to R$ from $lp : N_L \to N_P$, $rp : N_R \to N_P$ and (INJ-O) follows from (INJ-I). Now to get (II) $\Rightarrow$ (I) it is sufficient to show (IDENT-I)$^{\mathrm{C}}$ $\Rightarrow$ (IDENT-O) and (DANGL-I)$^{\mathrm{C}}$ $\Rightarrow$ (DANGL-O). These are again easy diagram chases. ☺

For full details of the proof see Banach (1996). We immediately find:

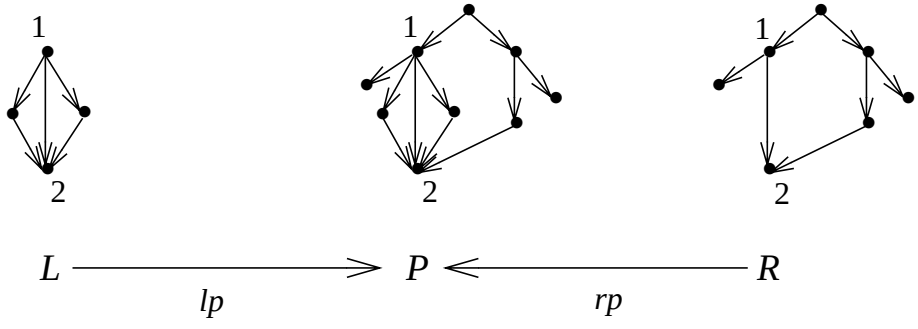
Theorem 4.2 Let $g : L \to G$ be a redex. Let r^{O} be an outward rule satisfying 4.1.(I) and r^{I} be an inward rule satisfying 4.1.(II), and such that r^{O} and r^{I} form a pushout in DG. Then H can be derived from G using r^{O} iff H can be derived from G using r^{I}.

5 An Example

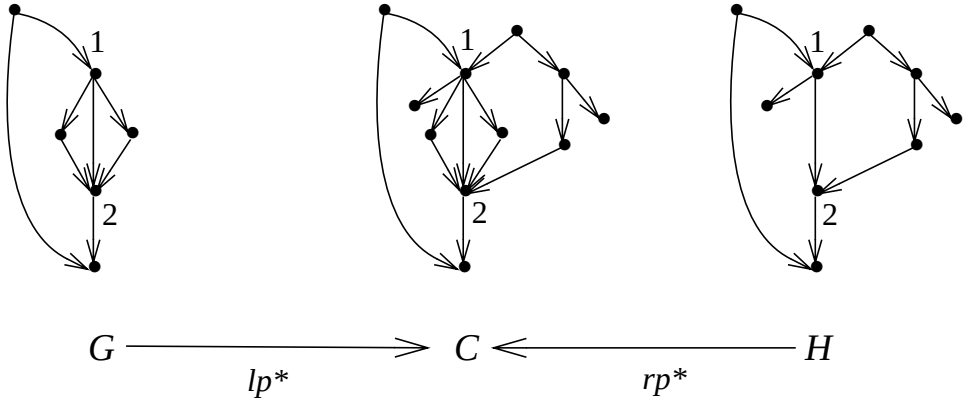
We present a short example of the preceding considerations. Below is an outward rule $L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ of DG, with numbered nodes carrying the morphism information.



Forming the pushout of these two arrows, we arrive at the corresponding inward form of the rule



When we apply this to the following graph G we get the sequence



In this two step sequence (the completion of the pushout $G \leftarrow D \rightarrow H$ in conventional outward rewriting, as the reader can check), we first bolt in the contractum, and then remove what needs to be removed in terms of the image of the LHS graph L. Note that the inward form embodies a small optimisation compared with the outward form, namely that the outward form first removes the arc (1, 2) in the construction of D, and then replaces it when the pushout with R is performed; this does not happen in the inward form. Of course one can prevent this inefficiency in the outward form by including the the arc (1, 2) in K, but this essentially says that outward rules ought to have the property of being a pullback image, as well as what we already demand of them.

6 An Application: Acyclic Rewriting

In this section we give a brief presentation of a topic where we claim that the inward rewriting approach has some advantages over the conventional outward form. Since the two approaches are entirely equivalent by theorems 4.1 and 4.2, there is of course nothing here that cannot ultimately be done in the outward style.

Acyclicity is an important safety property of computing systems. Deadlock avoidance in resource allocation systems is the classic example, but many other safety properties can be represented in terms of the acyclicity of some directed structure that models the state of the system as it evolves. We give here a simple theorem that guarantees that rewriting of an acyclic graph via a suitable rule preserves acyclicity. We stick to the category DG whose objects possess an obvious notion of acyclicity, and to the notation of the pushout cube.

Theorem 6.1 Let $\Gamma = lp : L \to P \leftarrow R : rp$ be an inward rule with lp (as well as rp) injective, and let $g : L \to G$ be a redex with g injective, in an acyclic graph G. Let $K^* \subseteq P$ be the subgraph given by $K^* = lp(L) \cap rp(R)$, and let Γ have the property:

(Π) P is acyclic; and for every directed path π in P between nodes a and b of K^*, there is a path θ in P between the same nodes a and b, but lying entirely within $lp(L)$.

Let H be the result of rewriting G via the rule Γ. Then H is acyclic.

Proof. Let H arise via the arrows $lp^* : G \to C \leftarrow H : rp^*$ of the pushout cube. We claim that C is acyclic; which is sufficient since H arises via the inverse homomorphism $C \leftarrow H : rp^*$ and inverse homomorphisms preserve acyclicity.

To substantiate the claim, suppose there was a cycle $\Omega = [x,\ldots, x]$ in C. The cycle cannot lie entirely in $lp^*(G)$ since lp is injective, and neither can it lie entirely in $c(P)$ since g is injective, by properties of pushouts. Therefore it must lie in $lp^*(G) \cup c(P)$, and split into $[\alpha_1,\ldots, \beta_1]$, $[\beta_1,\ldots, \alpha_2]$, …, with $[\alpha_i,\ldots, \beta_i] \subseteq (c(P) - (lp^*(G) - c(K^*)))$, and $[\beta_i,\ldots, \alpha_{i+1}] \subseteq (lp^*(G) - (c(P) - c(K^*)))$, (and cyclically), and with the $\{\alpha_i, \beta_i\} \subseteq c(K^*) = lp^*(G) \cap c(P)$. Since c is injective, each $[\alpha_i,\ldots, \beta_i]$ is the c image of a unique path $\pi_i = [a_i,\ldots, b_i] \subseteq P$ with $\{a_i, b_i\} \subseteq K^*$, whence property ($\Pi$) supplies us with a corresponding path $\theta_i \subseteq lp(L)$ also from a_i to b_i. Replacing each $[\alpha_i,\ldots, \beta_i]$ by $c(\theta_i) \subseteq lp^*(G)$ in the cycle Ω, gives us a cycle entirely in $lp^*(G)$, a contradiction. ☺

One can see that to achieve the same thing in the outward approach would be somewhat more cumbersome. The inverse homomorphism of the pushout complement of the outward approach is less useful than that of the inward approach since one adds material to the graph subsequently, and one has to check that the new material does not inadvertently close a cycle. All the pieces required for the argument are present in the outward approach to be sure, but they lie scattered about in a number of different graphs so that building the contradiction is little less easy. Furthermore, in the inward approach, it is a lot more convenient to check whether a particular rule satisfies the condition (Π) by inspecting a diagram of the intermediate graph P; one can check whether any path π between nodes (a, b) of K^* and straying outside of $lp(L)$, has an alternative route between a and b entirely within $lp(L)$, at a glance.

An easy induction now gives:

Theorem 6.2 Let G_0 be an acyclic initial graph, and let R be an inward rule system in which each rule consists of a pair of injective arrows satisfying condition (Π) above. Then every graph generated from G_0 by rewriting injective redexes using rules from R is acyclic.

Theorem 6.1 applies to the example described in the previous section as is easily seen. The graph G is acyclic, the rule employed and the redex satisfy the relevant conditions, and as a consequence, the graph H is acyclic too.

As with all safety properties, by working harder and inventing more subtle invariants of the objects of interest, one can generalise and strengthen the above results in a number of different ways. In

addition, one can adapt the arguments to suit graph rewriting in other categories of graphs. However to do so would take us far outside the scope of this paper.

7 Conclusions

In the previous sections, we have reviewed double pushout algebraic graph rewriting and term graph rewriting, both from a conveniently uncluttered perspective, that of the category DG. By an algebraic trick, we were able to reformulate the former construction from its original outward form, into a new inward form, that bore comparison with term graph rewriting. Nevertheless, one should not try to push the analogy too far. Algebraic graph rewriting is "equational" in a way that term graph rewriting is not. Specifically, if in an algebraic rewrite, node x is to be merged with node y, and node y is to be merged with node z, then a pushout will ensure that in-arcs to all three nodes end up at the same node of the result. However, in term graph rewriting, if $\langle x, y\rangle$ and $\langle y, z\rangle$ are two redirections, then the in-arcs of x end up at y, and the in-arcs of y and z end up at z. To emulate the algebraic behaviour we would need $\langle x, z\rangle$ and $\langle y, z\rangle$. Thus there are phenomena in term graph rewriting that do not correspond to algebraic graph rewriting.

Finally, we showed off the potential advantages that the new approach has in certain areas, by giving a simple theorem on the preservation of acyclicity. The category of graphs used DG, is too austere for the given result to be of immediate and great value in real world applications, but the argument used in the proof is one that stands generalisation to more complex categories whose objects and arrows are much more suitable for representing real applications. Further elaboration of these ideas will appear in other papers.

References

Banach R. (1996); Locating the Contractum in the Double Pushout Approach, Theoretical Computer Science **156**, to appear.

Banach, R. (1994); Term Graph Rewriting and Garbage Collection Using Opfibrations, Theoretical Computer Science **131**, 29-94.

Barendregt H.P., van Eekelen M.C.J.D., Glauert J.R.W., Kennaway J.R., Plasmeijer M.J., Sleep M.R. (1987); Term Graph Rewriting, *in*: Proc. PARLE-87, de Bakker J.W., Nijman A.J., eds., Lecture Notes in Computer Science **259** 141-158, Springer, Berlin.

Ehrig H. (1979); Introduction to the Algebraic Theory of Graph Grammars (A survey), *in*: Lecture notes in Computer Science **73**, 1-69, Springer, Berlin.

Ehrig H. (1986); A Tutorial Introduction to the Algebraic Approach of Graph Grammars, *in*: Third International Workshop on Graph Grammars, Lecture Notes in Computer Science **291**, 3-14, Springer, Berlin.

Ehrig H., Habel A., Kreowski H-J., Parisi-Presice F. (1991a); From Graph Grammars to High Level Replacement Systems, *in*: Fourth Int. Workshop on Graph Grammars and their Applications to Computer Science, Ehrig, Kreowski, Rozenberg (eds.), Lecture Notes in Compter Science **532**, 269-291, Springer, Berlin.

Ehrig H., Habel A., Kreowski H-J., Parisi-Presice F. (1991b); Parallelism and Concurrency in High Level Replacement Systems, Mathematical Structures in Computer Science **1**, 361-404.

Ehrig H., Kreowski H-J. (1979); Pushout Properties: An Analysis of Gluing Constructions for Graphs, Mathematische Nachrichten **91**, 135-149.

Ehrig H., Kreowski H-J., Taentzer G. (1993); Canonical Derivations for High-Level Replacement Systems, in: Graph Transformations in Computer Science, Schneider, Ehrig (eds.), Lecture Notes in Computer Science **776**, 152-169, Springer, Berlin.

Habel A., Kreowski H., Plump D. (1988); Jungle Evaluation, *in*: Proc. Fifth Workshop on Specification of Abstract Data Types, Sannella D., Tarlecki A., eds., Lecture Notes in Computer Science **332**, Springer, Berlin.

Hoffman B., Plump D. (1988); Jungle Evaluation for Efficient Term Rewriting, *in*: Proc. International Workshop on Algebraic and Logic Programming, Mathematical Research **49**, Akademie-Verlag, Berlin.

Plump D. (1993); Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence, *in*: Term Graph Rewriting: Theory and Practice, Sleep et al. (eds.), John Wiley.

Sleep M.R., Plasmeijer M.J., van Eekelen M.C.J.D. (eds.) (1993); Term Graph Rewriting: Theory and Practice, John Wiley.

T.C.S. (1993); Special Issue of Selected Papers of the International Workshop on Computing by Graph Transformation, Bordeaux, France, 1991. Theoretical Computer Science, **109**, Nos. 1-2.