ELSEVIER

Theoretical Computer Science 152 (1995) 305–320

Theoretical Computer Science

Locating the contractum in the double pushout approach

R. Banach

Department of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, UK

Received September 1993; revised July 1994
Communicated by G. Rozenberg

Abstract

Double pushout (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 double pushout graph rewriting.

1. Introduction

Graph rewriting in the double pushout approach (referred to below for brevity as 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 [5–7]. See also [19].

Term graph rewriting arose rather more recently [3] 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 [18].

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.

SSDI 0304-3975(94)00301-7

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 [11, 12, 17], so one might speculate that the two approaches are not so far apart.

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 rewriting world. A contrasting approach to the reconciliation question may be found in [16, 13].

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 is 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 short example. Section 6 examines the algebraic and TGR models of rewriting more closely, and argues that the analogy between them should not in fact be pushed too far. 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 [7, 8, 10]. Since the main point that this paper makes is algebraic in nature, resting as it does on the construction of a cube of pushouts as we will see later, it too can be adapted to many such categories – specifically, categories possessing finite colimits can be expected to support a version of the construction below. 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 able to construct the appropriate generalisations as required.

Let $\mathscr{DG}$ be the category of directed graphs and graph morphisms. An object G of $\mathscr{DG}$ 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 $\mathscr{DG}$ is a map $g: N_G \to N_H$ such that

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

Like many categories of graph-like systems, $\mathscr{DG}$ has all pushouts. (See e.g. [4] for definitions of pushouts and other categorical apparatus.) 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 $k \in N_K$ 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 varsion of $u \in [x]_P$.

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 $\mathscr{DG}$:

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

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 $L \leftarrow K \to R$ is an arrow $g: L \to G$, and the rewrite proceeds by constructing the diagram in Fig. 1 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 provides 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 aformentioned $g \circ l(K)$.

The first stage attempts to construct the object D and the arrows $d: K \to D$, $l^*: D \to G$, 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) $\mathscr{DG}$ has all pushouts. It is standard lore in algebraic graph rewriting theory that a unique smallest pushout complement exists if

(INJ-O) $l: K \to L$ 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 $\{x, y\} \cap g(N_L) \neq \emptyset$

$\Rightarrow \{x, y\} \cap g(N_L) \subseteq g(l(N_K))$.

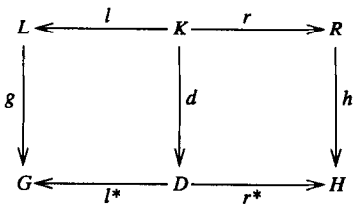

Fig. 1.

(INJ-O), which we have assumed already, ensures that a pushout complement with unique $d(K)$ exists if one exists at all. (In other categories of graph-like systems, INJ-like conditions give rise to other forms of uniqueness.) (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 $\mathscr{DG}$, 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 $d: K \to D$ is given by

$$d: N_K \to N_D : x \mapsto g(l(x))$$

with the obvious extension to arcs. Arrow $l^*: D \to G$ is just the inclusion on $N_G - g(N_L - l(N_K))$ again with the obvious extension to arcs. In the sequel, it is always the unique smallest pushout complement that we mean when we speak of pushout complements in $\mathscr{DG}$.

For examples of algebraic graph rewriting, see the literature cited in the introduction.

3. Term graph rewriting

Just as most applications of algebraic graph rewriting use categories of objects with a richer structure than $\mathscr{DG}$, 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 $\mathscr{DG}$, 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 $\mathscr{DG}^{(*)}$, whose objects and arrows are those of $\mathscr{DG}$, except that each nonempty object G, optionally has a distinguished node, the root of G, $root_G$. In fact $\mathscr{DG}$ occurs as full a subcategory of $\mathscr{DG}^{(*)}$. Each object G of $\mathscr{DG}$ occurs both "as is" in $\mathscr{DG}^{(*)}$, 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 $\mathscr{DG}^{(*)}$ is like an arrow in $\mathscr{DG}$ except that if G has a root $root_G$, then H must have one, $root_H$, and we must have

$$g(root_G) = root_H.$$

Under these circumstances, readers can check that $\mathscr{DG}^{(*)}$ 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 $g: N_K \to N_Y$, (see the appendix). 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 $\mathscr{DG}^{(*)}$ 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 $g: L \to (G, root_G)$, where G must have a root, $root_G$; except that we must have

(LIVE) Each node $g(x)$ occurring in the image of a redex $g: L \to G$ 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. Consequently, all arcs in the redex image are also live.

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. (By an "honest" pushout, we mean one that arises in the usual manner as a colimit of a pair of arrows, rather than by construction of a complement.) 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 $g'(x)$ (where g' is the extension of g provided by the pushout of the first stage). (Note that unlike the original reference [3], we are permitting multiple simultaneous redirections. There is no technical reason why we should not.) Having performed the redirections, 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 $\mathscr{DG}^{(*)}$, since of the three graphs involved in *incl* and g, only G has a root. Obviously G' has a root, such that $incl'(root_G) = root_{G'}$ (see Fig. 2).

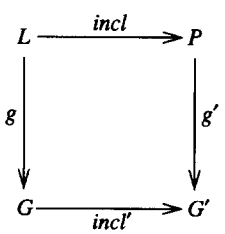

Fig. 2.

$P - L$, 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) $\quad \langle x', y'\rangle \in Red' \quad and \quad \langle x', z'\rangle \in Red' \Rightarrow y' = z'.$

(Actually there is an alternative approach, where one nondeterministically selects a maximal functional subrelation of the general relation Red', and subsequently works with that; but we will ignore such a possibility in this paper.) 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 give a graph G''.

$$N_{G''} = N_{G'}$$

$$A_{G''} = (A_{G'} - A^{LHS}_{Red'}) \cup A^{RHS}_{Red'},$$

$$root_{G''} = \text{If } \langle root_{G'}, y\rangle \in Red' \text{ for some } y \in N_{G'} \text{ then } y$$

$$\text{else } root_{G'},$$

where

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

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

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

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

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

$$root_H = root_{G''}.$$

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 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''. We will return to this in Section 6.

For examples of term graph rewriting, see the literature cited in the introduction; further examples appear in [2].

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 $L \xrightarrow{l} K \xrightarrow{r} R$ be an algebraic rule. It consists of two arrows of $\mathscr{DG}$ with common domain K. Therefore, we can form the pushout shown in Fig. 3. 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 pushout cube shown in Fig. 4.

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.

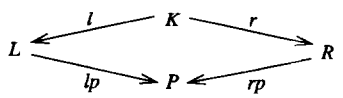

Fig. 3.

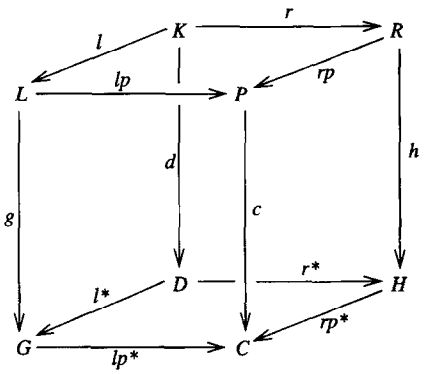

Fig. 4.

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 $P - lp(L)$ 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.

We will discuss the analogy between term graph rewriting and algebraic rewriting some more in Section 6. For now we turn to the technical details of the new construction.

Because $\mathscr{DG}$ has all 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* $\mathscr{DG}$*, 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. Before proving that (I) $\Leftrightarrow$ (II) we show that the conditions quoted in (II) are sufficient for guaranteeing that an inward rewrite of $g: L \to G$ exists. Since $\mathscr{DG}$ has all pushouts, we merely need to show that the analogous conditions for the smallest pushout complement of $rp: R \to P$ and $c: P \to C$ hold. These are:

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

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

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

Clearly (INJ-I) $\Rightarrow$ (INJ-I)^C.

For (IDENT-I) $\Rightarrow$ (IDENT-I)^C suppose $c(x) = c(y)$. If $x = y$ we are done, so suppose not. Now C is the pushout of g and lp, and for nodes, this is just a pushout in $\mathscr{Set}$. Thus for some c_0, $lp^{*-1}(c_0)$ and $\{x,y\} \subseteq c^{-1}(c_0)$ are all in the same equivalence class of $G \uplus P$ under $\approx$ in the explicit pushout construction. Therefore there are distinct elements $\{\gamma_1, \gamma_2, \ldots, \gamma_n\} \subseteq G$, $\{x, \pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n, y\} \subseteq P$, $\{\lambda_{1x}, \lambda_{11}, \lambda_{12}, \lambda_{22}, \ldots, \lambda_{n-1n}, \lambda_{nn}, \lambda_{ny}\} \subseteq L$ such that

$$
\begin{aligned}
g(\lambda_{1x}) &= \gamma_1, & lp(\lambda_{1x}) &= x \\
g(\lambda_{11}) &= \gamma_1, & lp(\lambda_{11}) &= \pi_1, \\
&\vdots & &\vdots \\
g(\lambda_{ij}) &= \gamma_i, & lp(\lambda_{ij}) &= \pi_j, \\
&\vdots & &\vdots \\
g(\lambda_{n-1n}) &= \gamma_{n-1}, & lp(\lambda_{n-1n}) &= \pi_n, \\
g(\lambda_{nn}) &= \gamma_n, & lp(\lambda_{nn}) &= \pi_n, \\
g(\lambda_{ny}) &= \gamma_n, & lp(\lambda_{ny}) &= y.
\end{aligned}
$$

Evidently, each γ_i is the g image of a pair of distinct λ's, λ_{i-1i} and λ_{ii}. Thus by (IDENT-I) $lp(\lambda_{ij}) \in rp(N_R)$ for all i, j, and in particular $\{x,y\} \subseteq rp(N_R)$.

For (DANGL-I) $\Rightarrow$ (DANGL-I)^C, let $(x,y) \in A_C - c(A_P)$ (we say that (x,y) is dangling in C with respect to P), and say $x \in c(N_P)$. Then if $lp^*((\gamma_x,\gamma_y)) = (x,y)$, then (γ_x,γ_y) dangles in G with respect to L, otherwise chasing its preimage under g through L and P would force a contradiction. So $\gamma_x \in g(N_L)$. By (DANGL-I), $\gamma_x \in g(lp^{-1}(rp(N_R)))$, whence $x \in c(rp(N_R))$. This is sufficient as the argument for $y \in c(N_P)$ is similar.

We have shown that the smallest pushout complement of $rp: R \to P$ and $c: P \to C$ exists under the conditions stated. These will later turn out to be equivalent to the conditions (II), so the latter will form a set of conditions for the existence of inward rewrites, expressed solely in terms of the redex and the arrows in the rule. Now for the main part of the proof.

(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).

For (IDENT-O) $\Rightarrow$ (IDENT-I), let $g(x) = g(y)$ and $x \neq y$. Since $\{x,y\} \subseteq l(N_K)$, let $l(k_1) = x$ and $l(k_2) = y$. Then $lp(x) = rp(r(k_1))$; similarly for y, whence $\{lp(x), lp(y)\} \subseteq rp(N_R)$.

For (DANGL-O) $\Rightarrow$ (DANGL-I), let (x,y) dangle in G with respect to L, and $x \in g(N_L)$. By (DANGL-O) $x \in g(l(N_K))$, say $x = g(l(k))$. Then since $lp(l(k)) = rp(r(k))$, $l(k) \in lp^{-1}(rp(N_R))$ and $x \in g(lp^{-1}(rp(N_R)))$.

(II) $\Rightarrow$ (I). Suppose we have the hypotheses of (II). From (SURJ-I) and (INJ-I) and Theorem A.1 from the appendix, we can clearly form an essentially unique pushout kernel in $\mathscr{Set}$ of $lp: N_L \to N_P$, $rp: N_R \to N_P$; it is just the pullback of the two arrows in $\mathscr{Set}$. This yields $l: N_K \to N_L$, $r: N_K \to N_R$. There is an arc $(x,y) \in A_K$ whenever both $(l(x), l(y)) \in A_L$ and $(r(x), r(y)) \in A_R$. We get finally $l: K \to L$, $r: K \to R$. Since rp is injective, we get (INJ-O).

For (IDENT-I) $\Rightarrow$ (IDENT-O), it is clearly sufficient to show (IDENT-I)^C $\Rightarrow$ (IDENT-O). So suppose $g(x) = g(y)$ and $x \neq y$. Since by (IDENT-I)^C, $\{lp(x), lp(y)\} \subseteq rp(N_R)$, there are $\{\rho_1, \rho_2\} \subseteq N_R$ with $rp(\rho_1) = lp(x)$, $rp(\rho_2) = lp(y)$. Since LKRP is a pushout (and pullback), there must be $\{k_1, k_2\} \subseteq N_K$ such that $l(k_1) = x$, $l(k_2) = y$, $r(k_1) = \rho_1$, $r(k_2) = \rho_2$; whence $\{x,y\} \subseteq l(N_K)$.

For (DANGL-I) $\Rightarrow$ (DANGL-O), it is clearly sufficient to show (DANGL-I)^C $\Rightarrow$ (DANGL-O). So suppose (x,y) is dangling in G with respect to L, and $x \in g(N_L)$. Clearly $(lp^*(x), lp^*(y))$ dangles in C with respect to P, and $lp^*(x) \in c(N_P)$. By (DANGL-I)^C, $lp^*(x) \in c(rp(N_R))$. If $\lambda \in g^{-1}(x)$, then $lp(\lambda) \in rp(N_R)$, so since LKRP is a pushout (and pullback), there must be a $k \in N_K$ with $\lambda = l(k)$, and so $x \in g(l(N_K))$. We are done. □

We immediately find:

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

5. Example

We present a short example of the preceding considerations. Fig. 5 shows outward rule $L \leftarrow K \rightarrow R$ of $\mathscr{DG}$, with numbered nodes carrying the morphism information.

Forming the pushout of these two arrows, we arrive at the corresponding inward form of the rule shown in Fig. 6.

When we apply this to graph G we get the sequence shown in Fig. 7. In this two-step sequence (the completion of the pushout $G \leftarrow D \rightarrow H$ in conventional outward

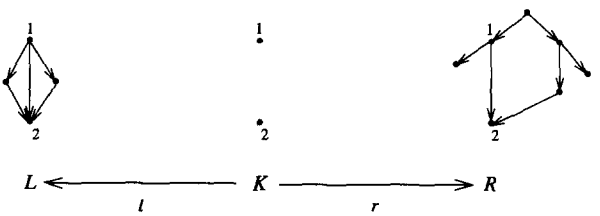

Fig. 5.

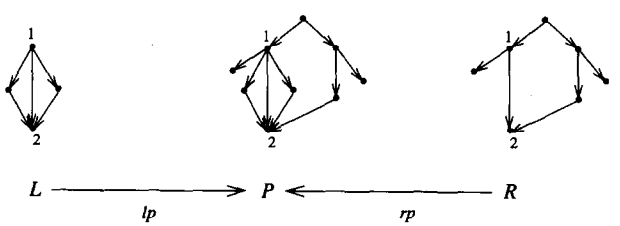

Fig. 6.

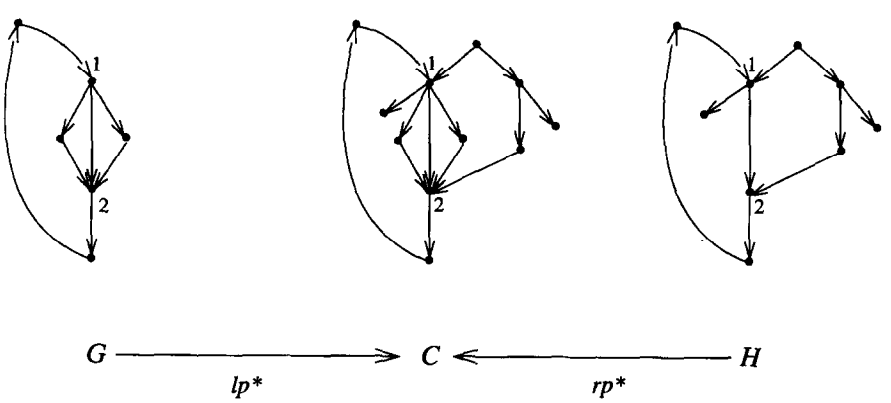

Fig. 7.

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 arc $(1,2)$ in K, but this essentially says that outward rules ought to have the property of being a pullback image (see the appendix), as well as what we already demand of them.

6. More on the algebraic contractum and inward rules

We have pointed out in Section 4 how the first pushout, of $g: L \to G$ and $lp: L \to P$ in the inward form of an algebraic rewrite, can be viewed as the algebraic analogue of the contractum building process in term graph rewriting. This is a largely accurate intuition, but there is one property of contractum builiding in TGR that is not necessarily shared by its algebraic cousin, namely the injectivity of $incl: L \to P$. In such a case, part of the job of the TGR redirection phase is being done by the algebraic contractum building phase. (The rest of redirection may be viewed in the algebraic world as being done by replacement of (copies of) arcs in L by (copies of) arcs in R.)

Where $lp: L \to P$ is not injective in the algebraic world, we can search for a closer correspondence by attempting to factor lp thus

$$lp: L \to P = (lp^0: P' \to P) \circ (lp': L \to P'),$$

where $lp': L \to P'$ is injective and $image(lp^0) = image(lp)$. In general there is no unique way to do this. For example, if nodes x and y in L are distinct, and map under lp to a node p in P, then if p is the head or tail of an arc in P, there are normally several ways of placing arcs in P' so that a suitable factorisation holds with lp' injective.

Sometimes, as in the category $\mathscr{DG}$, there is a "best way" of choosing these arcs, namely if there are several pre-images of $p \in P$ under lp^0, called $p'_1 \ldots p'_n$ say, they all receive a copy of every arc incident on $p \in P$. This amounts to constructing a weak final object $lp^0_F: P'_F \to P$ in the category of factorisations of lp with first factor injective (and the condition on images); where there is an arrow

$$\Pi_{12}: (lp^0_1: P'_1 \to P) \to (lp^0_2: P'_2 \to P)$$

if there is a factorisation

$$L \xrightarrow{lp'_1} P'_1 \xrightarrow{\pi_{12}} P'_2 \xrightarrow{lp^0_2} P.$$

In the specific category $\mathscr{DG}$, $lp^0_F: P' \to P$ collapses all the copies of an arc incident on p back down to a single arc in P.

On other occasions, notably in the category $\mathscr{J}$ of jungles [11, 12], there is no such weak final object, due to the existence of certain invariants which have to hold for objects or arrows. (In the case of $\mathscr{J}$, this is that a node may occur as the (single) in-node of at most one hyperedge.)

The presence of invariants for objects, as in $\mathscr{J}$, may preclude certain pushouts from existing, and this may apply to the top square of the pushout cube itself, giving rise to the situation where the outward form of a rule is legal, but the inward form is not. In such cases it is usually clear how to generalise the definition of the category so that both forms are legal, but often at the price of admitting objects and arrows (and rewrites), that were considered undesirable before. Usually the situation can be reformulated by recasting the rewriting mechanism in an opfibration framework, divorcing the objects and arrows in the base, from the rewrites themselves, allowing different invariants to hold in different parts of the description. See [2] for such a treatment of term graph rewriting.

So our analogy between algebraic graph rewriting and term graph rewriting is not a perfect one. This should not be surprising. 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.

This is further enhanced by the observations on garbage, in Section 3. In algebraic graph rewriting, the garbage in a rewrite is always locally and deterministically deducible from the redex. (However an element of nondeterminism does creep into rewriting because a redex does not guarantee the ability to do a rewrite, unless the conditions for a pushout complement are satisfied; although this can sometimes be forced by syntactic restrictions, it is usually inconvenient). By contrast in term graph rewriting, it is easy to arrange that each redex will lead to a rewrite but the garbage is now determined nonlocally; a search of the whole graph, not just the redex, is needed to separate the live part from the garbage.

In undemanding cases, such as the rewriting of acyclic graphs corresponding to well-behaved term rewrite systems, these more obscure phenomena are not called upon, and good agreement can be reached between TGR and algebraic descriptions of the rewriting process. Nevertheless, one should not expect such good agreement in all cases.

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 $\mathscr{DG}$. 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. However, in Section 6 particularly, a closer inspection revealed that one should not try to push the analogy between the two rewriting models too far.

This being the case, one is tempted to ask if there is any real use for the new version of the algebraic construction, other than the rather minor optimisation we noticed in the example of Section 5. The answer turns out to be yes. In fact the inward form can yield facts about rules and rewrites that are rather more obscure when using the conventional form. Particular instances occur in the hypergraph rewriting formulation of the rewriting model for interaction nets [14, 15, 1], in the hypergraph formulation of multiplicative linear logic, and when arguing about acyclicity more generally. These phenomena will be described in other papers, as they would take us on an excessively lengthy detour from our main point if presented here.

Appendix. Pushout kernels and pullback images in $\mathscr{Set}$

If $f: K \to X$ and $g: K \to Y$ are arrows in $\mathscr{Set}$, it is well known that their pushout $s: X \to P$, $t: Y \to P$ is given by

> $P = X \uplus Y/\approx$, where $\uplus$ is disjoint union, and $\approx$ is the smallest equivalence relation such that $x \approx y$ if there is a $k \in K$ such that $f(k) = x$ and $g(k) = y$.

and $s: X \to P, t: Y \to P$ takes each x or y to its equivalence class in P. We can write an alternative form for the pushout – up to isomorphism. Let $\approx_K$ be the smallest equivalence relation on K such that $k_1 \approx_K k_2$ if $f(k_1) = f(k_2)$ or if $g(k_1) = g(k_2)$.

Let $[K]_K$ be the set of equivalence classes of K under $\approx_K$. Then the alternative form reads

$$P = (X - f(K)) \uplus (Y - g(K)) \uplus [K]_K.$$

It is also well known that if $s: X \to P$ and $t: Y \to P$ are two arrows of $\mathscr{Set}$, their pullback $f: K \to X$, $g: K \to Y$ is given by

$$K = \{\langle x, y\rangle \mid \exists p \in P \text{ such that } s(x) = p = t(y)\}$$

and $f: K \to X, g: K \to Y$ are projections that take each $\langle x, y\rangle \in K$ to its left and right component, respectively. Again there is an alternative form which reads

$$K = \bigcup\{s^{-1}(p) \times t^{-1}(p) \mid p \in s(X) \cap t(Y)\}.$$

Now we ask the questions: when is a pair of arrows $s: X \to P$. $t: Y \to P$ a pushout of some pair $f: K \to X$, $g: K \to Y$ which we call a pushout kernel of s and t; also when is a pair $f: K \to X$, $g: K \to Y$ a pullback of some pair $s: X \to P$, $t: Y \to P$ which we call a pullback image of f and g. The answers are provided by the following results.

Theorem A.1. *Let $s: X \to P$, $t: Y \to P$ be arrows in $\mathscr{Set}$. Then*

(1) *If $P \neq s(X) \cup t(Y)$, or s is not injective on $(X - s^{-1}(t(Y)))$, or t is not injective on $(Y - t^{-1}(s(X)))$, then there is no $f: K \to X$, $g: K \to Y$, of which s and t are the pushout.*

(2) *Else s and t do have a pushout kernel, but it is not unique.*

(3) *If s and t have a pushout kernel and t (say) is injective, then there is a unique smallest (up to isomorphism) pushout kernel of s and t, $f: K \to X$, $g: K \to Y$; f is injective, and the pushout kernel is the pullback of s and t.*

Theorem A.2. *Let $f: K \to X$, $g: K \to Y$ be arrows in $\mathscr{Set}$. Then*

(1) *f and g have a pullback image $s: X \to P$, $t: Y \to P$ iff*

(a) *For $x \in X$, $y \in Y$, $|\{k \mid f(k) = x \text{ and } g(k) = y\}| \leqslant 1$,* and

(b) *For $X' \subseteq X$, $|X'| \leqslant 2$, and $Y' \subseteq Y$, $|Y'| \leqslant 2$, $|\{k \in K \mid f(k) = x \in X', \text{ and } g(k) = y \in Y'\}| \neq 3$.*

(2) *If f and g have a pullback image s and t, then up to isomorphism they have a smallest one; given (predominantly) by the pushout of $f(K)$ and $g(K)$*

$$P = [f(K) \boxplus_K g(K)] \uplus D,$$

where $D = \emptyset$ if f and g are both surjective, D is a singleton if only one of f or g is surjective, and D is a doubleton if neither f nor g is surjective. The maps s and t are as for a pushout except that if neither f nor g is surjective and $D = \{a, b\}$ then $s(X - f(K)) = \{a\}$ and $t(Y - g(K)) = \{b\}$, the other cases being degenerate versions of this.

The proofs of both of these theorems may be easily reconstructed from the facts established in [9], which deals with similar material.

References

[1] R. Banach, The algebraic theory of interaction nets, *Math. Systems Theory* (1993) submitted.

[2] R. Banach, Term graph rewriting and garbage collection using opfibrations, *Theoret. Comput. Sci.* **131** (1994) 29–94.

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

[4] M. Barr and C. Wells, *Category Theory for Computing Science* (Prentice-Hall, Englewood Cliffs, NJ, 1990).

[5] H. Ehrig, Introduction to the algebraic theory of graph grammars (A survey), in: Lecture notes in Computer Science, Vol. 73 (Springer, Berlin, 1979) 1–69.

[6] H. Ehrig, A tutorial introduction to the algebraic approach of graph grammars, in: *3rd Internat. Workshop on Graph Grammars*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1986) 3–14.

[7] H. Ehrig, A. Habel, H.-J. Kreowski and F. Parisi-Presice, From graph grammars to high level replacement systems, in: H. Ehrig, H.-J. Kreowski, G. Rozenberg, eds., *4th Internat. Workshop on Graph Grammars and their Applications to computer Science*, Lecture Notes in Computer Science, Vol. 532 (Springer, Berlin, 1991) 269–291.

[8] H. Ehrig, A. Habel, H.-J. Kreowski and F. Parisi-Presice, Parallelism and concurrency in high level replacement systems, *Math. Struct. Comput. Sci.* **1** (1991) 361–404.

[9] H. Ehrig and H.-J. Kreowski, Pushout properties: an analysis of gluing constructions for graphs, *Math. Nachr.* **91** (1979) 135–149.

[10] H. Ehrig, H.-J. Kreowski, and G. Taentzer, Canonical derivations for high-level replacement systems, in: Schneider, Ehrig eds., *Graph Transformations in Computer Science*, Lecture Notes in Computer Science, Vol. 776 (Springer, Berlin, 1992) 152–169.

[11] A. Habel, H. Kreowski, and D. Plump, Jungle evaluation, in: D. Sannella and A. Tarlecki, eds., *Proc. 5th Workshop on Specification of Abstract Data Types*, Lecture Notes in Computer Science, Vol. 332 (Springer, Berlin, 1988).

[12] B. Hoffmann and D. Plump, Jungle evaluation for efficient term rewriting, in: *Proc. Int. Workshop on Algebraic and Logic Programming*, Mathematical Research, Vol. 49 (Akademie-Verlag, Berlin, 1988).

[13] J.R. Kennaway, Graph rewriting in some categories of partial morphisms, in: H. Ehrig, H. Kreowski and G. Rozenberg, eds., *Graph Grammars and their Application to computer Science*, Lecture Notes in Computer Science, Vol. 532 (Springer, Berlin, 1991) 490–504.

[14] Y. Lafont, Interaction nets, in: *Proc. 7th ACM Symp. on Principles of Programming Languages* (1990) 95–108.

[15] Y. Lafont, The paradigm of interaction, manuscript, 1991.

[16] M. Lowe, Extended algebraic graph transformation, Ph.D. Thesis, Technical University of Berlin, 1990.

[17] P. Plump, Hypergraph rewriting: critical pairs and undecidability of confluence, in: M.R. Sleep et al., eds., *Term Graph Rewriting: Theory and Practice* (Wiley, New York, 1993) 201–213.

[18] M.R. Sleep, M.J. Plasmeijer and M.C.J.D. van Eekelen, eds., *Term Graph Rewriting: Theory and Practice* (Wiley, New York, 1993).

[19] TCS, Special Issue of Selected Papers of the International Workshop on Computing by Graph Transformation, Bordeaux, France, 1991; *Theoret. Comput. Sci.* **109** (1993) (1–2).