DACTL REWRITING IS CATEGORICAL

R. Banach

Computer Science Department, Manchester University,

Manchester, M13 9PL, U.K.

Abstract

The graph-manipulating core of the general term graph rewriting language DACTL, namely contraction building and redirection, is reexamined from a categorical viewpoint. The essentials of this rather complex two-phase operational semantics is recast as a Grothendieck opfibration of a category of graph rewrites over a base of rewrite rules. This generalises previous attempts to categorise contractum building and redirection as pushouts and is able to describe more DACTL rewrites than pushout models. The full operational core model conforms to a more restricted version of this construction and is able to successfully cope with examples such as the infamous circular I example a : I[a].

1 INTRODUCTION

The general term graph rewriting language DACTL arose as an attempt to provide an intermediate language for graph rewriting based implementations of contemporary programming paradigms such as functional or logic. It features multiple parallel redirections as the chief updating mechanism for the computational objects of the model, general term graphs. Parallel redirection provides a powerful metaphor for substitution in a graph-oriented environment, so there is an incentive to find a clean formulation for the whole of the semantics. The heart of the problem is the contraction building and redirection phases of a rewrite. Partial success in this area has been obtained by describing graph rewrites as certain kinds of pushout. See Kennaway (1987), Hoffmann and Plump (1988), Habel Kreowski and Plump (1988) and Kennaway (1991). Nevertheless none of these constructions successfully describe the whole of the operational models of Barendregt et al. (1987) where term graph rewriting was introduced, or of its generalisation in the general term graph rewriting language DACTL itself (Glauert et al. (1988a, b, 1990), Kennaway (1990)). The main stumbling blocks for all of these attempts have been examples such as the I combinator I[x] $\Rightarrow$ x when applied to a circular instance of itself a : I[a]. None of the hitherto proposed categorical formulations of TGR adequately capture the DACTL version of the rewrite (which is, reasonably enough, a null action), nor do they give a convincing story of their own (generally speaking the result of the rewrite is undefined). The aim of this paper is to describe how these deficiencies may be overcome by using a different approach to the categorical semantics of rewriting. Instead of pushouts, we use a Grothendieck opfibration. Here, the base category has patterns as objects and the arrows are the rewrite rules. Above each object is a category of its instances in graphs. These fibers are glued together using a functor from the base, to yield a Grothendieck category whose arrows are rewrites between graphs, governed by the appropriate rule. Now Grothendieck opfibrations have strong universal properties, too strong to be applicable to all DACTL rewrites. Accordingly, a less universal construction describes the full operational core of DACTL rewriting. It turns out that the circular I example sits in between these two extremes.

In outline, the rest of the paper is as follows. Section 2 describes an abstracted version of DACTL operational semantics. A significant issue here is what aspects of the language are *not* being described, so these are listed. Section 3 describes the categorical construction and how it yields a Grothendieck opfibration. It turns out that the garbage retention feature of DACTL is the key to the success of the construction. Section 4 relates this construction to “real systems” by discussing some of the issues omitted from section 3.

Section 5 reconsiders true DACTL rewriting and outlines the universal construction that describes it, including the circular I example. Since this paper is an abridged version of Banach (1991), proofs are omitted from section 3, and the discussion in sections 4 and 5 is more superficial. Section 6 is a conclusion.

2 DACTL ABSTRACTED

In this section, we define that part of a DACTL rewrite which we consider to be the rewriting core of the model. The aspects of DACTL that we ignore are those to do with the markings on the graph, and the pattern calculus. Thus for us, rule selection and reduction strategy are outside the remit of rewriting. Accordingly all our graphs and patterns will bear no markings. In addition, our terminology may appear a little unusual to those familiar with DACTL. Suppose an alphabet of **node symbols $\mathbf{S}$** = $\{S, T\ldots\}$ to be given.

Definition 2.1 A **term graph** (or just **graph**) G is a triple (N, σ, α) where

(1) N is a set of nodes,

(2) σ is a map with signature $N \to \mathbf{S}$,

(3) α is a map with signature $N \to N^*$.

Thus σ maps a node to the node symbol that labels it, and α maps each node to its sequence of successors. We write $A(n)$, the **arity** of a node, for the domain of $\alpha(n)$. Note that $A(n)$ is a set of consecutive natural members starting at 1, or empty. We allow ourselves to write $x \in G$ instead of $x \in N(G)$ etc. Each successor node determines an arc of the graph, and we will refer to arcs using the notation (p_k, c), to indicate that the child c is the k'th child of the parent p, ie. that $c = \alpha(p)[k]$ for some $k \in A(p)$. When we speak of several graphs (or patterns, see below) simultaneously, as we will do in a moment, we will subscript N, σ and α with the name of the graph in question in order to clarify which map we are refering to. Moreover, to be quite unambiguous when dealing with disjoint unions, the elements of such a union will always be tagged with either $\langle 1, -\rangle$ or $\langle 2, -\rangle$ to indicate their origin.

Let there be a symbol Any, not normally considered to be in $\mathbf{S}$. We will assume the following invariant holds subsequently:

(ANY) $\sigma(x) = \mathsf{Any} \;\Rightarrow\; A(x) = \varnothing$.

A node labelled with Any is called **implicit**, a node labelled with a member of $\mathbf{S}$ is called **explicit**.

Definition 2.2 A **pattern** is a term graph containing zero or more implicit nodes.

Thus every graph is a pattern (if we choose to regard it as such) but not vice versa.

Definition 2.3 A rule D is a triple $\langle P, r, Red\rangle$ where

(1) P is a pattern (called the full pattern of the rule).

(2) r is an explicit node of P called the root. If $\sigma(r) = F$ then D is called a **rule for** F. The subpattern L of P, consisting of nodes and arcs accessible from (and including) r is called the left subpattern of (the full pattern P of) the rule D. All implicit nodes of P must be nodes of L.

(3) Red is a set of pairs (called redirections) of nodes of P. These satisfy the invariants (RED-1), (RED-2) and (RED-3) below:

(RED-1) Red is the graph (in the set theoretic sense) of a partial function on P.

(RED-2) $\langle l', r'\rangle \in Red \;\Rightarrow\; l'$ is an explicit node of L.

(RED-3) Let $\langle l_1, r_1\rangle, \langle l_2, r_2\rangle \in Red$. If $l_1 = l_2$ and there is a Z such that there is a homomorphism (see 2.4 below) $h : P \to Z$ such that $h(l_1) = h(l_2)$, then $r_1 = r_2$.

The three invariants (RED-1) – (RED-3) assure the existence of rewrites as described below. In practice the technically rather involved (but nevertheless decidable) (RED-3) may be replaced by the slighty stronger but more readable (RED-3′):

(RED-3′) $\langle l_1, r_1\rangle, \langle l_2, r_2\rangle \in Red$ and $l_1 \neq l_2 \Rightarrow \sigma(l_1) \neq \sigma(l_2)$.

To define the rewriting model, we must first define the notion of homomorphism of patterns and graphs.

Definition 2.4 Let P, Z be patterns. A map $h : P \to Z$ is a homomorphism if for all explicit $x \in P$

(1) $\sigma(x) = \sigma(h(x))$,

(2) $A(x) = A(h(x))$,

(3) for all $k \in A(x)$, $h(\alpha(x)[k]) = \alpha(h(x))[k]$.

In addition a homomorphism is **strict** if the image of every implicit node is implicit. Note that 2.4 serves as well for graphs as it does for patterns.

In brief, a rewrite of a graph G (G could just as easily be a pattern) according to a rule $D = \langle P, r, Red\rangle$ proceeds through three stages. Firstly a homomorphism of the left subpattern L of P into G is located. This is the redex. Then copies of the other nodes and arcs of P are added to G in order to extend the homomorphism to one from the whole of P. Finally arcs whose destination is the image of the LHS of a redirection pair $\langle l, r\rangle \in Red$, are swung over to arrive instead at the image of the corresponding RHS. More formally we have the definitions below.

Definition 2.5 Let $D = \langle P, r, Red\rangle$ be a rule. Let G be a graph. Let L be the left subpattern of P. Let $m : L \to G$ be a homomorphism. Then $m(L)$ is a redex in G and $m(r)$ is the root of the redex. The homomorphism is called a **matching** of L to G.

Definition 2.6 Assume the notation of 2.5. Let the graph G' be given by

(1) $N_{G'} = (N_G \uplus N_P)/\approx$ which is the disjoint union of N_G and N_P factored by the equivalence relation $\approx$, where $\approx$ is the smallest equivalence relation such that $\langle 1, x\rangle \approx \langle 2, n\rangle$ whenever $m(n) = x$.

(2) $\sigma_{G'}(\{\langle 1, x\rangle\}) = \sigma_G(x)$,
$\sigma_{G'}(\{\langle 2, n\rangle\}) = \sigma_P(n)$,
$\sigma_{G'}(\{\langle 1, x\rangle, \langle 2, n_1\rangle \ldots \langle 2, n_q\rangle\}) = \sigma_G(x)$.
Thus G' aquires symbols in such a way as to agree with both G and P; the representative in G' of an implicit node of P aquiring a symbol according to its image under m.

(3) $\alpha_{G'}(\{\langle 1, x\rangle\})[k] = \{\langle 1, \alpha_G(x)[k]\rangle \ldots\}$ for $k \in A(x)$,
$\alpha_{G'}(\{\langle 2, n\rangle\})[k] = \{\langle 2, \alpha_P(n)[k]\rangle \ldots\}$ for $k \in A(n)$,
$\alpha_{G'}(\{\langle 1, x\rangle, \langle 2, n_1\rangle \ldots \langle 2, n_q\rangle\})[k] = \{\langle 1, \alpha_G(x)[k]\rangle \ldots\}$.
Thus G' aquires arcs so as to agree with both G and P. The … on the RHS of these cases indicate that the equivalence classes concerned need not be singletons.

Lemma 2.7 There is a homomorphism $m' : P \to G'$. Disregarding pedantry, m' extends $m : L \to G$.

Proof. Define $m'(n) = \{\langle 2, n\rangle \ldots\}$. That m' has the properties stated follows readily from the properties of m and $\approx$. We call m' the **extended matching**. ☺

Definition 2.8 Assume the notation of 2.5 – 2.7. Let H be the graph given by

(1) $N_H = N_{G'}$,

(2) $\sigma_H = \sigma_{G'}$,

(3) $$\alpha_H(\{\langle 1,x\rangle\})[k] = \begin{cases} \{\langle 2, y\rangle \ldots\} & \text{if } \langle u, y\rangle \in Red \text{ for some } y \in P \text{ and } u \in m'^{-1}(\alpha_{G'}(\{\langle 1, x\rangle\})[k]) \\ \alpha_{G'}(\{\langle 1, x\rangle\})[k] & \text{otherwise} \end{cases}$$

$$\alpha_H(\{\langle 2,n\rangle\})[k] = \begin{cases} \{\langle 2, y\rangle \ldots\} & \text{if } \langle u, y\rangle \in Red \text{ for some } y \in P \text{ and } u \in m'^{-1}(\alpha_{G'}(\{\langle 2, n\rangle\})[k]) \\ \alpha_{G'}(\{\langle 2, n\rangle\})[k] & \text{otherwise} \end{cases}$$

$$\alpha_H(\{\langle 1, x\rangle, \langle 2, n_1\rangle \ldots \langle 2, n_q\rangle\})[k] = \begin{cases} \{\langle 2, y\rangle \ldots\} & \text{if } \langle u, y\rangle \in \mathit{Red} \text{ for some } y \in P \text{ and } u \in m'^{-1}(\alpha_{G'}(\{\langle 1, x\rangle \ldots\})[k]) \\ \alpha_{G'}(\{\langle 1, x\rangle, \langle 2, n_1\rangle \ldots \langle 2, n_q\rangle\})[k] & \text{otherwise} \end{cases}$$

This construction is consistent by (RED-1) and (RED-3). (RED-1) ensures that for any u there is at most one y such that $\langle u, y\rangle \in \mathit{Red}$. (RED-3) ensures that if there are several distinct $\langle 2, u\rangle$'s in $m'^{-1}(\ldots)$ then their corresponding $\langle 2, y\rangle$'s are identical. (RED-3′) ensures the same thing by precluding more than one $\langle 2, u\rangle$ from membership of any $m'^{-1}(\ldots)$.

Definition 2.9 Let G be a graph, D be a rule, and m a matching of the left subpattern L of (the full pattern of) D to G. The graph H constructed via 2.5 – 2.8 is the result of the rewrite of G at the redex $m(L)$ according to D.

Remark. It is worth mentioning here that the model of DACTL rewriting just presented differs from the DACTL of the final specification (Glauert et al. (1988a)) in minor details. In particular the final specification has moved from DACTL's original position, particularly in the area of what combinations of redirections are permissible, as a result of the influence of the categorical semantics of papers already cited. Since our aim is to present a categorical semantics for as "pure" a form of DACTL rewriting as is possible, our model is closer in spirit to slightly earlier versions. The reader unaquainted with these subtleties will not find this an obstacle to understanding the rest of this paper.

The rewriting model just described is a lot easier to understand using pictures rather than the formidable construction above. Accordingly we present a couple of examples.

Example 2.10 In fig1.(a) – (e) we illustrate a rule D and its action on a graph G. The full pattern of the rule P, is given in fig1.(b). The root of the rule is the node r, whence the left subpattern is the pattern L given in fig 1.(a). There is clearly an injection $i : L \to P$. The redirections of D are $\mathit{Red} = \{\langle \mathsf{r}, \mathsf{r}'\rangle, \langle \mathsf{c}, \mathsf{c}'\rangle\}$. The graph G is given in fig 1.(c) and it contains an instance of L which is given by $m : L \to G$ where the (set theoretic) graph of m is $\{\langle \mathsf{r}, \mathsf{y}\rangle, \langle \mathsf{c}, \mathsf{z}\rangle, \langle \mathsf{a}, \mathsf{t}\rangle\}$. This m clearly satisfies the conditions of 2.4. The rewrite proceeds by creating the graph G' (contractum building) given in fig 1.(d), where we have avoided pedantry by dropping the $\langle 1, –\rangle$ and $\langle 2, –\rangle$ labels of the disjoint union and have just introduced copies of the non-L nodes and arcs. The extended matching is given by $m' = m \cup \{\langle \mathsf{r}', \mathsf{y}'\rangle, \langle \mathsf{c}', \mathsf{z}'\rangle, \langle \mathsf{p}, \mathsf{p}'\rangle, \langle \mathsf{l}, \mathsf{l}'\rangle\}$. The final stage of rewriting performs the redirections. The images of the redirection pairs Red are located in G'. These are respectively $\langle \mathsf{y}, \mathsf{y}'\rangle = m'(\langle \mathsf{r}, \mathsf{r}'\rangle)$ and $\langle \mathsf{z}, \mathsf{z}'\rangle = m'(\langle \mathsf{c}, \mathsf{c}'\rangle)$. The arcs of G' whose targets are y and z are redirected so that their targets are y' and z' respectively. This gives the graph H of fig 1.(e).

Example 2.11 As a further example we treat the circular I rewrite discussed already. Fig 2.(a) illustrates the rule. In this rule the left subpattern L is identical to the full pattern of the rule P, and the redirections are $\mathit{Red} = \{\langle \mathsf{r}, \mathsf{a}\rangle\}$. The graph G of fig 2.(b) contains an instance of L by $m = \{\langle \mathsf{r}, \mathsf{x}\rangle, \langle \mathsf{a}, \mathsf{x}\rangle\}$. Since $L = P$, there are no nodes to be added at the contraction building stage so $G = G'$. To perform the redirections and complete the rewrite we look for the image of $\langle \mathsf{r}, \mathsf{a}\rangle$ in G'. This is $\langle \mathsf{x}, \mathsf{x}\rangle$. Therefore all nodes targeted at x must be redirected to x, a null action. So $H = G' = G$.

One feature of this model stands out, which is that no node is ever destroyed during rewriting. This means that copious quantities of garbage are generated, even though we have not considered how to distinguish the live part from the garbage in any graph. The attempts to describe graph rewriting using pushouts fail on the circular I example, precisely because they do some partial garbage collection in the arrows of the categories used. As the next section will show, the garbage retention feature of DACTL turns out to be an inspired design decision.

We will return to reconsider garbage briefly in section 4, although the limited space available to us in this paper will not really enable us to do the topic justice.

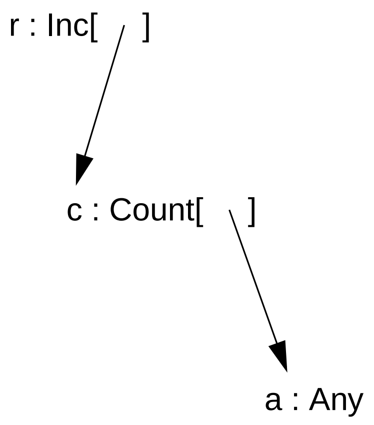


Fig. 1.(a) *L*

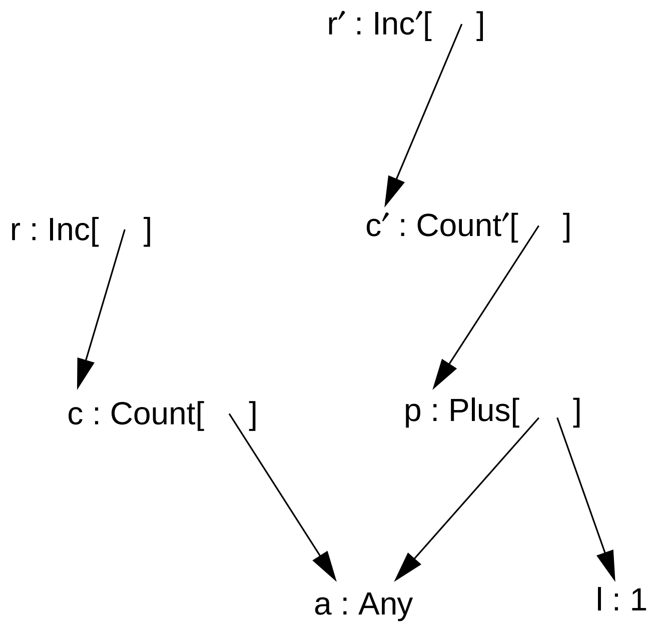


Fig. 1.(b) *P*

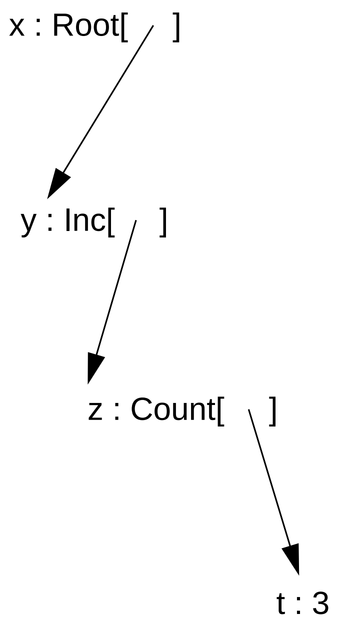


Fig. 1.(c) *G*

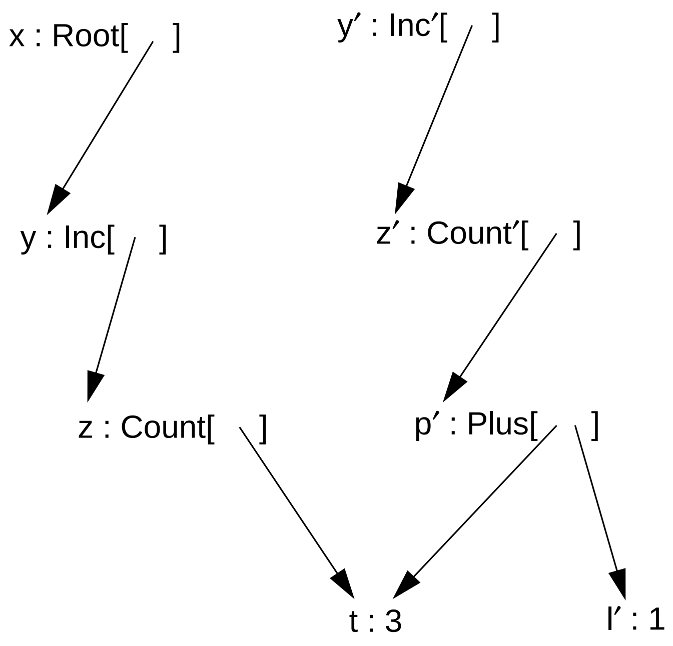


Fig. 1.(d) *G′*

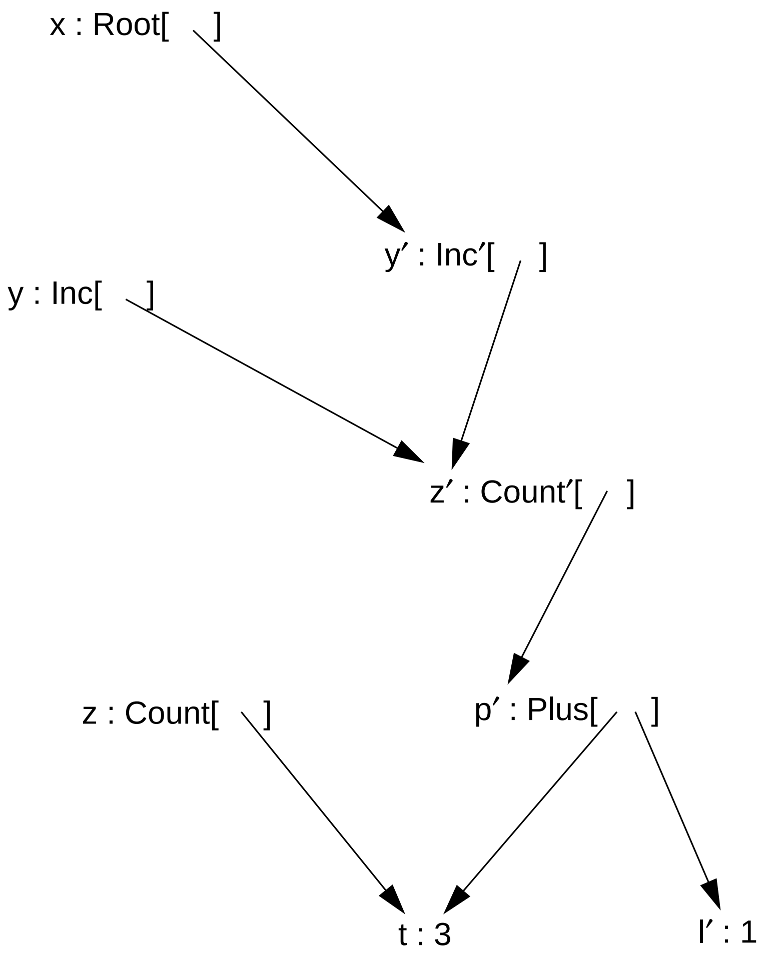
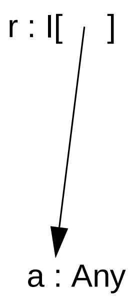


Fig. 1.(e) *H*



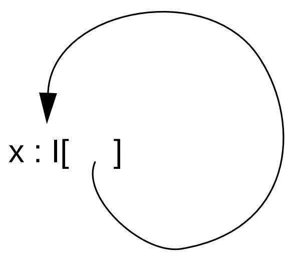


Fig. 2.(a) $L = P$

Fig. 2.(b) $G = G' = H$

3 GRAPH REWRITING AS GROTHENDIECK OPFIBRATION

To avoid excessive technical doudgery, we will state significant categorical results in terms of abstract patterns and graphs, which are representatives of the isomorphism classes of patterns and graphs. Where we need to give an explicit construction, we will tend to use "normal" or "concrete" patterns and graphs, since life will be easier with an explicit model of disjoint union. We will be a bit more cavalier in illustrations, modelling disjoint union by adding nodes to one of the summands.

Definition 3.1 Let $\mathcal{P}$ be the category whose objects are abstract patterns and whose arrows are rules depicted by pairs of functions $(i, r) : L \to R$ satisfying the invariants (INJ), (RED), (HOM) below. L and R are called the left and right patterns of the rule (arrow) $(i, r) : L \to R$.

(INJ) (a) $x \neq y \;\Rightarrow\; i(x) \neq i(y)$,
(b) x implicit $\;\Rightarrow\;$ $i(x)$ implicit,
(c) y implicit $\;\Rightarrow\; \exists$ implicit x such that $y = i(x)$,
(d) $\sigma(i(x)) = \sigma(x)$,
(e) $A(i(x)) = A(x)$.

(RED) (a) $r(x) \neq i(x) \;\Rightarrow\;$ x explicit,
(b) x implicit $\;\Rightarrow\; r(x) = i(x)$,
(c) x, y explicit and $\exists$ a homomorphism $h : L \to Z$ such that $h(x) = h(y) \;\Rightarrow\;$ [$r(x) = r(y)$, or $r(x) = i(x)$ and $r(y) = i(y)$].

(HOM) (p_k, c) an arc of $L \;\Leftrightarrow\; (i(p)_k, r(c))$ an arc of R.

As usual, this looks a lot more complicated than it is. (INJ) just states that $i : L \to R$ is a symbol/arity-preserving injection that is invertible on the implicit subpatterns of L and R (ie. "no new variables are introduced in the RHS of the rule"). (RED) states that the redirection function $r : L \to R$ may only redirect explicit nodes and has to behave unambiguously with respect to homomorphic images. As previously we can replace (RED).(b) with the stronger but more transparent

(RED) (b′) x, y explicit and $\sigma(x) = \sigma(y) \;\Rightarrow\;$ [$r(x) = r(y)$, or $r(x) = i(x)$ and $r(y) = i(y)$].

For an object P, the identity in $\mathcal{P}$ is $(id_P, id_P) : P \to P$ and composition of the arrows $(i_1, r_1) : P_1 \to P_2$, $(i_2, r_2) : P_2 \to P_3$ is given by $(i_2 \circ i_1, r_2 \circ r_1) : P_1 \to P_3$. It is trivial on the basis of known properties of $\mathcal{S}et$, to check that all the stated invariants are preserved by this composition.

We see that the arrows of $\mathcal{P}$ provide a component i that mimics the inclusion of the left subpattern into the full pattern of a DACTL rule of the previous section, and also a component r that mimics the redirection pairs of the previous treatment. Note the careful distinction in nomenclature. While it is possible to identify the left subpattern of a DACTL rule with the left (abstract) pattern of an arrow of $\mathcal{P}$, the same cannot be done with the full pattern of a DACTL rule and the corresponding right pattern of an arrow.

A subset of DACTL rules can be easily mapped to $\mathcal{P}$ arrows. This subset is characterised by the property (RED-$\mathcal{P}$) below. We call this subset DACTL$^{\mathcal{P}}$.

(RED-$\mathcal{P}$) x, y explicit and $\exists$ a homomorphism $h : L \to Z$ such that $h(x) = h(y) \;\Rightarrow$
[$\langle x, t\rangle \in Red \;\Leftrightarrow\; \langle y, t\rangle \in Red$].

There is an easy mapping from DACTL$^{\mathcal{P}}$ rules into arrows of $\mathcal{P}$. It is given by the next construction.

Construction 3.2 Let $\langle incl : L \to P, root, Red\rangle$ be a DACTL$^{\mathcal{P}}$ rule. We have remarked that DACTL rewriting semantics can just as easily be applied to instances of L in patterns as to instances in graphs. Let R be the pattern resulting from rewriting the identity instance of L in itself according to the rule. Then the arrow of $\mathcal{P}$ corresponding to the rule is (the abstract version of) $(i, r) : L \to R$ with (i, r) given by

$i(x) = \{\langle 1, x\rangle, \langle 2, x\rangle\}$,
$r(x) = \{\langle 1, y\rangle, \langle 2, y\rangle\}$, where $y = x$ unless $\langle x, y\rangle \in Red$.

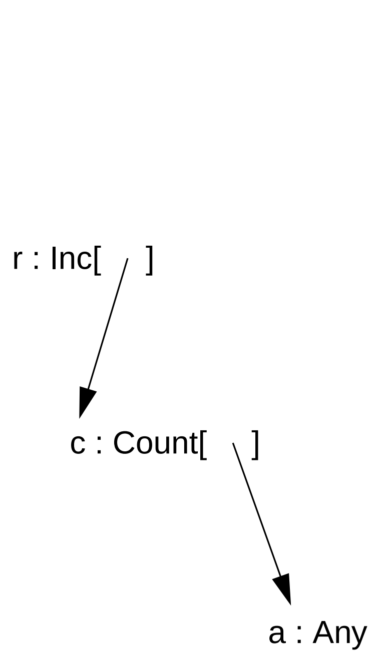


Fig. 3.(a) *L*

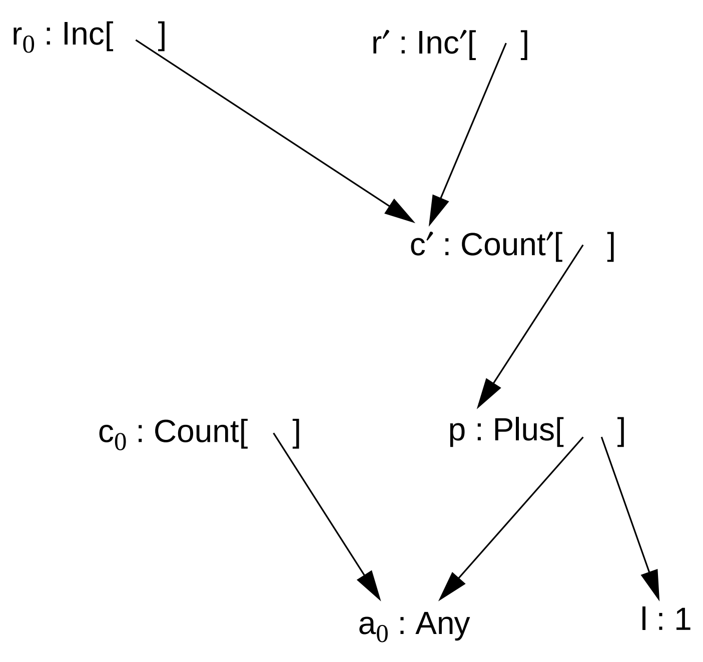


Fig. 3.(b) *R*

Modulo the pedantry of disjoint unions, we have just applied the redirections *Red* to the pattern *P*. Fig 3 shows the $\mathcal{P}$ version of the rule of example 2.10 (which is obviously a DACTL$^{\mathcal{P}}$ rule). The map *i* is just $\{\mathsf{r} \to \mathsf{r}_0, \mathsf{c} \to \mathsf{c}_0, \mathsf{a} \to \mathsf{a}_0\}$, and *r* is given by $\{\mathsf{r} \to \mathsf{r}', \mathsf{c} \to \mathsf{c}', \mathsf{a} \to \mathsf{a}_0\}$.

We now give the $\mathcal{P}$ version of graph rewriting which we call the $\mathcal{P}$ rewriting construction, to distinguish it from the DACTL rewriting construction of the previous section.

Definition 3.3 Let $\delta = (i, r) : L \to R$ be an arrow of $\mathcal{P}$, and let $g : L \to G$ be a rigid homomorphism of *L* into a graph *G*, by which we mean a homomorphism such that (RIG) below holds.

(RIG) x explicit, y implicit $\Rightarrow$ $g(x) \neq g(y)$.

Let the graph *H* be given by:

(1) $N_H = (N_G \uplus N_R)/\approx$ where $\uplus$ is disjoint union and $\approx$ is the smallest equivalence relation such that $\langle 1, x\rangle \approx \langle 2, n\rangle$ whenever there is a $p \in L$ such that $x = g(p)$ and $n = i(p)$. Thus N_H is the pushout in Set of $R \xleftarrow{i} L \xrightarrow{g} G$.

(2) $\sigma_H(\{\langle 1, x\rangle\}) = \sigma_G(x)$,
$\sigma_H(\{\langle 2, n\rangle\}) = \sigma_R(n)$,
$\sigma_H(\{\langle 1, x\rangle, \langle 2, n_1\rangle \ldots \langle 2, n_q\rangle\}) = \sigma_G(x)$.

Before defining α_H, we pause to define $(j, s) : G \to H$ and $h : R \to H$

$j(x) = \{\langle 1, x\rangle \ldots\}$,

$$s(x) = \begin{cases} \{\langle 2, r(p)\rangle \ldots\} & \text{if } \exists\, p \in L \text{ such that } x = g(p) \text{ and } r(p) \neq i(p) \\ \{\langle 1, x\rangle \ldots\} & \text{otherwise} \end{cases}$$

$h(n) = \{\langle 2, n\rangle \ldots\}$,

(3) $\alpha_H(\{\langle 1, x\rangle\})[k] = s(\alpha_G(x)[k])$,
$\alpha_H(\{\langle 2, n\rangle\})[k] = h(\alpha_R(n)[k])$,
$\alpha_H(\{\langle 1, x\rangle, \langle 2, n_1\rangle \ldots \langle 2, n_q\rangle\})[k] = s(\alpha_G(x)[k])$.

Lemma 3.4 Definition 3.3 is consistent. Furthermore

(a) j is a symbol/arity-preserving injection,

(b) h is a rigid homomorphism,

(c) (j, s) is a redirection couple i.e. [(x_k, y) an arc of $G \Leftrightarrow (j(x)_k, s(y))$ an arc of H].

Lemma 3.5 In the notation of 3.3, 3.4, $j \circ g = h \circ i$ and $s \circ g = h \circ r$, i.e. the squares of fig. 4 commute.

We do not need to look far for an example of this construction. Example 2.10 furnishes what we need. We have a P style rule for this rewrite in fig 3. Figs. 1.(c),(e) provide the graphs G and H. The maps $(i, r) : L \to R$ have been given, and $g : L \to G$ is just the map m of example 2.10. The map $j : G \to H$ is the obvious injection, and the map $s : G \to H$ has $\mathsf{y} \to \mathsf{y}'$, $\mathsf{z} \to \mathsf{z}'$ and otherwise coincides with j. The homomorphism $h : R \to H$ is $\{\mathsf{r}_0 \to \mathsf{y}, \mathsf{c}_0 \to \mathsf{z}, \mathsf{a}_0 \to \mathsf{t}, \mathsf{r}' \to \mathsf{y}', \mathsf{c}' \to \mathsf{z}', \mathsf{p} \to \mathsf{p}', \mathsf{l} \to \mathsf{l}'\}$.

In general, the two rewriting models agree. We have the following result.

Theorem 3.6 Let $\langle incl : L \to P, root, Red\rangle$ be a DACTL$^{\mathsf{P}}$ rule and let $(i, r) : L \to R$ be the corresponding P arrow. Let $g : L \to G$ be a rigid matching of L to a graph G. Then the abstract versions of the graphs H built by the two rewriting constructions are the same.

Let us compare the two rewriting constructions for a moment from the perspective of potential implementations. In practice one would never implement either of the constructions by forming explicit disjoint unions etc. Instead one might implement the DACTL construction by the steps (D1) – (D3).

(D1) Identify the redex $m : L \to G$ of rule $\langle incl : L \to P, root, Red\rangle$ in graph G.

(D2) Add copies of nodes and arcs of $P - L$ to G in order to make graph G'.

(D3) Redirect arcs of G' according to the specifications in Red to make graph H.

The P construction on a suitable redex might proceed as follows.

$$\begin{tikzcd} G \arrow[r, "j"] & H \\ L \arrow[u, "g"] \arrow[r, "i"] & R \arrow[u, "h"'] \end{tikzcd} \qquad \begin{tikzcd} G \arrow[r, "s"] & H \\ L \arrow[u, "g"] \arrow[r, "r"] & R \arrow[u, "h"'] \end{tikzcd}$$

Fig. 4

(P1) Identify the rigid redex $g : L \to G$ of rule $(i, r) : L \to R$ in graph G.

(P2) Add copies of nodes of $R - i(L)$ to G to make graph G_1. Determine the maps $(j, s) : G \to G_1$.

(P3) Redirect arcs of G_1 making graph G_2 so that [(p_k, c) an arc of $G_1 \Leftrightarrow (j(p)_k, s(c))$ an arc of G_2].

(P4) Add copies of arcs of $R - i(L)$ to G_2 to make H.

The differences are obvious. The DACTL construction is operationally much simpler. In addition, the intermediate object G' that it creates is a bona-fide graph as per 2.2. The P construction is more complex and its intermediate objects G_1 and G_2 aren't really graphs since they have "missing arcs". This is most acutely felt if we work in a world where symbols have fixed arities. In the P construction we are forced to "interleave (D3) into (D2)" since the pattern R comes with its arcs "already redirected". Incidentally this last point shows us that there is no inverse transformation from P rewriting back to DACTL$^{\mathsf{P}}$ rewriting, since in the latter a node which is redirected but is not itself the target of a redirection, definately has no parents, while in P rewriting, such a node can acquire parents during step (P4).

In view of the above, why bother with P rewriting at all? The answer comes with the simplicity of specifying P rewriting compared to DACTL rewriting. The universal properties of P rewriting make it much easier to reason about than DACTL rewriting, and the specification in 3.3 is considerably simpler than the combined force of 2.7 - 2.9. The payoff for using it is thus for the theoretitian.

We return to our primary objective of making a Grothendieck construction in the world of abstract patterns and graphs, via a universal property of P rewriting which makes it very reminiscent of a pushout.

Theorem 3.7 Using the notation of 3.3 – 3.5, let H' be a graph and suppose $(j', s') : G \to H'$ and $h' : R \to H'$ are such that

(1) j' is a symbol/arity preserving injection,

(2) (j', s') is a redirection couple i.e. [(x_k, y) an arc of $G \Leftrightarrow (j'(x)_k, s'(y))$ an arc of H'],

(3) h' is a homomorphism,

(4) $j' \circ g = h' \circ i$ and $s' \circ g = h' \circ r$,

(5) $i(a) = r(p)$ and $i(b) = r(q)$ and $g(a) = g(b) \Rightarrow s'(g(p)) = s'(g(q))$,

Then there is a unique pair of maps $(\theta, \rho) : H \to H'$ such that

(a) θ is a symbol/arity preserving node map,

(b) (θ, ρ) is a redirection couple i.e. [(p_l, c) an arc of $H \Leftrightarrow (\theta(p)_l, \rho(c))$ an arc of H'],

(c) θ, ρ extend to a homomorphism on $h(R)$,

(d) $j' = \theta \circ j$,
$s' = \rho \circ s$,
$h' = \theta \circ h = \rho \circ h$,
$\rho = \theta$ on $H - (s(G) \cup h(R))$.

I.e. fig. 5 commutes.

Theorem 3.7 shows the pushout-like nature of the P rewriting construction. The graph H that it creates is universal (up to isomorphism of course) among ways of completing the square in fig 5 according to the conditions stated.

Now we are in a position to proceed with the Grothendieck construction.

Definition 3.8 For each object P of P, we construct a category G^P. The objects of G^P are pairs $\langle G, g \rangle$. Here G is an abstract graph and $g : P \to G$ is a rigid homomorphism. (N.B. We could instead consider the category P^P of pairs $\langle K, g \rangle$ where this time K is an abstract pattern, along the lines of similar remarks above.) The arrows $\phi : \langle G, g\rangle \to \langle G', g'\rangle$ of G^P (or of P^P) are graph (or pattern) homorphisms ϕ which preserve the redex, i.e. $g' = \phi \circ g$, and also are rigid i.e. $g(P) = \phi^{-1}(g'(P))$. The two notions of rigidity should cause no confusion.

$$\begin{tikzcd}
 & & H' \\
G \arrow[r, "{(j, s)}"'] \arrow[rru, "{(j', s')}"] & H \arrow[ru, dashed, "{(\theta, \rho)}"] & \\
L \arrow[u, "g"] \arrow[r, "{(i, r)}"'] & M \arrow[u, "h"] \arrow[ruu, "h'"'] &
\end{tikzcd}$$

Fig. 5

Definition 3.9 Consider an arrow $\delta = (i, r) : L \to R$ in P, and $\langle G, g \rangle$ an object of G^L. Let $\langle H, h \rangle$ be the object of G^R such that H is the unique abstract graph isomorphic to the result of rewriting the instance $g : L \to G$ according to rule δ using the P rewriting construction, and $h : R \to H$ is the obvious homomorphism. Let $Rew^{\delta}(\langle G, g \rangle) = \langle H, h \rangle$.

Lemma 3.10 $Rew^{\delta} : \mathsf{G}^L \to \mathsf{G}^R$ extends to a functor. In other words fig. 6 commutes.

$$\begin{tikzcd}
 & G' \arrow[r, "{(j', s')}"] & H' & \\
 & G \arrow[u, "\phi"'] \arrow[r, "{(j, s)}"'] & H \arrow[u, dashed, "\psi"] & \\
L \arrow[ru, "g"'] \arrow[ruu, "g'"] \arrow[rrr, "{(i, r)}"'] & & & R \arrow[lu, "h"] \arrow[luu, "h'"']
\end{tikzcd}$$

Fig. 6

Theorem 3.11 There is a functor $Rew : \mathsf{P} \to \mathsf{Cat}$ such that

$$Rew(P) = \mathsf{G}^P$$

$$Rew(\delta : L \to R) = Rew^{\delta} : \mathsf{G}^L \to \mathsf{G}^R$$

The existence of $Rew : \mathsf{P} \to \mathsf{Cat}$ leads immediately to the construction of the Grothendieck category $\mathsf{G}(\mathsf{P}, Rew)$. The objects of $\mathsf{G}(\mathsf{P}, Rew)$ are pairs $\langle\langle G, g\rangle, L\rangle$ where L is an object of P and $\langle G, g\rangle$ is an object of $Rew(L)$. We can write such objects as $\langle g : L \to G\rangle$. The arrows of $\mathsf{G}(\mathsf{P}, Rew)$ are pairs $\langle\phi, \delta\rangle : \langle g : L \to G\rangle \to \langle h : R \to H\rangle$ where $\delta = (i, r) : L \to R$ is an arrow of P, and $\phi : Rew^{\delta}(G) \to H$ is an arrow of G^R. In slightly less combinatorial terms, an arrow $\langle\phi, \delta\rangle$ of $\mathsf{G}(\mathsf{P}, Rew)$ can be viewed as an abstract P rewrite of a redex $g : L \to G$ by a rule $\delta = (i, r) : L \to R$ giving $Rew^{\delta}(\langle G, g\rangle)$, composed with a homomorphism ϕ. Thus it can be given by a pair $(j, s) : G \to H$ where $j = \phi \circ j^*$, $s = \phi \circ s^*$, and $(j^*, s^*) : G \to Rew^{\delta}(G)$ represents the effect of Rew^{δ} on G. Clearly [(x_k, y) an arc of $G \Leftrightarrow (j(x)_k, s(y))$ an arc of H]. Such a pair (j, s) is strictly speaking a different thing from $\langle\phi, \delta\rangle$, but we will overlook this.

Composition of arrows $\langle\phi, \delta\rangle : \langle g : L \to G\rangle \to \langle h : M \to H\rangle$ and $\langle\chi, \varepsilon\rangle : \langle h : M \to H\rangle \to \langle k : N \to K\rangle$ is defined by

$$\langle\chi, \varepsilon\rangle \circ \langle\phi, \delta\rangle : \langle g : L \to G\rangle \to \langle k : N \to K\rangle \;=\; \langle\chi \circ Rew(\varepsilon)(\phi), \varepsilon \circ \delta\rangle$$

This situation is illustrated in fig. 7. Note that compared to our P rewriting construction, the arrows of $\mathsf{G}(\mathsf{P}, Rew)$ have an extra homomorphism "tacked onto the end". The fact that we can do this is purely a consequence of the fact that the individual Rew^{δ} functors mesh together to form the overall functor Rew, and thus that the square in fig. 7 commutes. The significance of these extra homomorphisms will be discussed in section 4 when we talk about real systems. The arrows of $\mathsf{G}(\mathsf{P}, Rew)$ will be called rewrites, or G rewrites if we wish particularly to distinguish them from P rewrites, or DACTL rewrites. Of course from a categorical viewpoint, we might justifiably prefer them to be called corewrites or oprewrites.

$$\begin{tikzcd}
 & & K \\
 & {} \arrow[r, "Rew(\varepsilon)"] & {} \\
{} \arrow[r, "Rew(\delta)"] & {} & \\
 & H \arrow[r] & Rew^{\varepsilon}(H) \arrow[uuu, "\chi"] \\
G \arrow[r] & Rew^{\delta}(G) \arrow[u, "\phi"] \arrow[r] & Rew^{\delta}(Rew^{\varepsilon}(G)) \arrow[u] \\
L \arrow[r, "\delta = (i, r)"] & M \arrow[r, "\varepsilon = (j, s)"] & N
\end{tikzcd}$$

Fig. 7

Continuing the development, $\mathsf{G}(\mathsf{P}, Rew)$ is a fibered category. The fibers are the G^L categories and the projection $F : \mathsf{G}(\mathsf{P}, Rew) \to \mathsf{P}$ takes objects $\langle g : L \to G\rangle$ to L and arrows $\langle\phi, \delta\rangle$ to δ. F is a split opfibration with splitting

$$\kappa(\delta, \langle g : L \to G\rangle) = \langle id_{Rew(\delta)(\langle G,\, g\rangle)}, \delta\rangle : \langle g : L \to G\rangle \to \langle Rew^{\delta}(g) : \delta(L) \to Rew^{\delta}(G)\rangle$$
$$= \langle g : L \to G\rangle \to \langle h : R \to H\rangle$$

where $\delta = (i, r) : L \to R$ and $Rew^{\delta}(\langle G, g\rangle) = \langle H, h\rangle$. Thus the components of our original P rewriting construction turn out to be precisely the components of the splitting κ of the opfibration F.

Split opfibrations have universal properties by virtue of being opcleavages. In the particular case of $\mathsf{G}(\mathsf{P}, Rew)$, this comes down to the following. Let $\delta = (i, r) : L \to M$ be an arrow of P and $\langle g : L \to G\rangle$ be an object of $\mathsf{G}(\mathsf{P}, Rew)$. The arrow $\langle id_{Rew(\delta)(\langle G,\, g\rangle)}, \delta\rangle = \kappa(\delta, \langle g : L \to G\rangle)$ of $\mathsf{G}(\mathsf{P}, Rew)$ has the property that if $\langle\chi, \gamma\rangle : \langle g : L \to G\rangle \to \langle k : N \to K\rangle$ is an arrow of $\mathsf{G}(\mathsf{P}, Rew)$ such that $F(\langle k : N \to K\rangle) = N$, then for any arrow $\varepsilon = (j, s) : M \to N$ of P such that $F(\langle\chi, \gamma\rangle) = \varepsilon\circ\delta$, there is a unique arrow $\langle\theta, \tau\rangle : \kappa(\delta, \langle g : L \to G\rangle)(\langle g : L \to G\rangle) \to \langle k : N \to K\rangle$ such that $\langle\theta, \tau\rangle\circ\kappa(\delta, \langle g : L \to G\rangle) = \langle\chi, \gamma\rangle$ and $F(\langle\chi, \gamma\rangle) = \varepsilon$. The degenerate case of this when $\varepsilon = id_M$ is just the abstract version of theorem 3.7. For a fuller description of the Grothendieck construction in its abstract form see Barr and Wells (1990).

Note the pleasing way that the Grothendieck construction has separated syntax and semantics while nevertheless keeping a very close connection between them. The syntactic objects, patterns, and rules between patterns, live in the base category. The semantic objects, graphs and the rewrites between them, live in the Grothendieck category above. The fact that we use two distinct categories related by the projection functor F and split opfibration κ, rather than a single category, gives us just the technical elbow room we need to treat similar objects slightly differently as the situation demands. This segregation of syntax and semantics was the main reason why we chose to work with G^P fibers rather than P^P fibers.

4 REAL SYSTEM ISSUES

The construction of the previous section belonged in the world of “all conceivable rewrites”. Real systems make use of only a small number of possibilities. In this section we discuss briefly the issues that curtail the choice.

The first thing we note is that arrows of $\mathsf{G}(\mathsf{P}, Rew)$ are pairs $\langle\phi, \delta\rangle$. While the δ part is manifestly uncontroversial, the presence of a non-trivial ϕ may be more questionable. First of all we note that ϕ is an arrow of G^P, so by changing the permitted arrows there we can modify the arrows of $\mathsf{G}(\mathsf{P}, Rew)$. Making the fibers G^P into discrete categories effectively gets rid of non-trivial ϕ’s. However non-trivial ϕ’s are useful for certain purposes. In Hoffmann and Plump (1988), Habel, Kreowski and Plump (1988), and Kennaway (1987), the desirablity of identifying identical subgraphs of the execution graph is expounded. This is useful for non-linear rewrite rules, or for maximal sharing of identical subgraphs to improve execution efficiency. By restricting the arrows of G^P to be (maximal) onto rigid homomorphisms we see that our general construction achieves at least some of these effects for free. Of course we don’t get *all* desirable homomorphisms this way because of the requirements of rigidity.

We now consider one aspect of the DACTL rewriting construction of section 2 that was quietly dropped in section 3. By definition all DACTL rules have rooted left subpatterns (in the terminology of section 2). Reflecting on this point opens up a whole tangle of related issues.

The left subpatterns have roots, but the full patterns themselves needn’t do so (see example 2.4). This suggests that the roots are connected with garbage collection, and its absence in the present rewriting models. However roots are also found in terms and term rewriting, an important application area for term graph rewriting, and there they play the role of distinguished node in the object being rewritten since there is no separate notion of garbage or its collection in term rewriting. In reality both aspects are connected via the notion of accessibility, since accessibility plays a role both in garbage collection and in the fact that the whole of a term is accessible from its root.

It turns out that the most elegant way of handling these issues is to formulate the concept of **notion of garbage**. A notion of garbage (perhaps better called a notion of liveness) is a proof system, which when supplied with a graph and perhaps some base case information, can be used to decide the liveness or garbageness of any node in the graph. For example, given a graph and a set of its nodes deemed live as base cases, and the inference rule "from x live and y a child of x deduce y live", we can deduce which nodes are live and which are garbage. So we have a notion of garbage; one related to roots and accessibility, since the base case nodes play the role of "roots as distinguished nodes", and accessibility is the sole rule of inference. Different notions of garbage are appropriate under different circumstances. When modeling term rewriting, one is interested in roots and accessibility, whereas when one uses the full DACTL language to model eg. communicating processes, a more complex notion using the graph markings, suppressed in this paper, is more suitable.

Garbage has to fit well with rewriting. Thus the garbage in a graph should be

(1) preserved unchanged and possibly added to by the rewriting mechanism,

(2) not be manipulated non-trivially by the rewriting mechanism.

In the context of Grothendieck rewriting, (2) means that garbage nodes should not aquire new parents, nor be involved in redirection as either left or right hand side. Formally expressed, (1) and (2) are preserved by composition of composable pairs of rewrites that satisfy them and lead to a subcategory of valid rewrites of the Grothendieck category $\mathsf{G}(\mathsf{P}, Rew)$. More specifically, let $\mathsf{G}(\mathsf{P}, Rew)_{\{\Theta\}}$ be the Grothendieck category constructed as in section 3, but embellishing the objects in the fibers with whatever additional data are necessary for the notion of garbage Θ to work properly. Then $\mathsf{G}(\mathsf{P}, Rew)_{\Theta}$ has the same objects as $\mathsf{G}(\mathsf{P}, Rew)_{\{\Theta\}}$ but only those arrows which are valid rewrites.

One cannot expect that $\mathsf{G}(\mathsf{P}, Rew)_{\Theta}$ is opfibered over P in the way that $\mathsf{G}(\mathsf{P}, Rew)$ is. The reason is that even given a fixed notion of garbage, it is not implicit in a graph rewrite rule, which of the redex nodes matched to its left pattern are going to be garbaged by the rewrite. Different redexes in different graphs give different answers. There is thus no way to project the distinction between live and garbage nodes in the objects of $\mathsf{G}(\mathsf{P}, Rew)$ down to the objects in the base. In addition, for an arbitrarily chosen notion of garbage, for a given object in the category of rewrites and arrow in the base, the required split opcartesian arrow needn't exist.

With a judiciously chosen notion of garbage, the Grothendieck construction yields a category of valid rewrites suitable for describing the class of graph rewrites used for modeling term rewriting; i.e. a category in which the objects are graphs equipped with a matching of a pattern, and whose live part is precisely that part of the graph accessible from a unique root node.

Categories such as $\mathsf{G}(\mathsf{P}, Rew)_{\Theta}$ are rich in data. Here is a typical arrow.

$$\langle \phi, (i, r) : L \to R \rangle : \langle g : L \to \langle G, Gar(G) \rangle\rangle \to \langle h : R \to \langle H, Gar(H) \rangle\rangle$$

The objects are pattern instances into "graphs with garbage", and the arrow data is a homomorphism ϕ and an arrow $(i, r) : L \to R$ of P. One can rearrange and forget various parts of this data to build categories that describe rewriting at various levels of abstraction. For instance one can identify graphs regardless of pattern instances, or identify objects up to isomorphism of their live subgraphs. Some minor technical points need to be taken care of in order to make these constructions functorial.

Given such identifications one can then locate subcategories that describe rewriting using only the rules of a given system, or subcategories that use only a given system and starting graph, or subcategories that use only a given system, starting graph and fixed reduction strategy.

It is interesting to compare the present treatment of rewriting with term rewriting. One can certainly model term rewriting in a "with garbage" Grothendieck construction using subterm copying rather than subgraph sharing as the matching and instantiation mechanism for variables. One can then take up a suitable notion of garbage and construct the analogue of $G(P, Rew)_\Theta$. Now it turns out for topological reasons, that one can construct a split opfibration with $G(P, Rew)_\Theta$ as Grothendieck category, and then construct a morphism of opfibrations that takes this to conventional term rewriting, which in turn can easily seen to be expressible as an opfibration. Thus we end up with a "garbage free" Grothendieck construction for term rewriting too. A similar strategy is much less convincing for graph rewriting and its potential success or failure depends on a number of technical questions. For a much more complete account of these and other related issues, see the previously advertised full version of this paper.

5 TRUE DACTL REWRITING

In section 3 we developed a categorical formulation for a sublanguage of DACTL. Readers may legitimately wonder to what extent the full DACTL language shares the properties of DACTL^P. The main problem encountered in applying the constructions of section 3 to the full language can be traced back to DACTL's capacity for ambiguous redirections. From the perspective of the Grothendieck construction, DACTL redirections can be ambiguous for two distinct reasons. The first concerns the priority mechanism that implicily determines targets for redirection. Thus if $\langle x, t\rangle \in Red$, and x and y both match the same graph node of the redex, but for no u do we have $\langle y, u\rangle \in Red$, then the redirection in Red wins over the unstated identity redirection of y. A similar thing happens for implicit nodes of the pattern when they match the same graph node as some x such that $\langle x, t\rangle \in Red$. Again the explicit redirection wins over the unstated identity redirection. Such phenomena prevent the right hand square in fig. 4 from commuting properly and is why the DACTL^P sublanguage contained specific conditions to prevent such behaviour.

It might be imagined from this that the prospects for describing the whole language categorically were bleak. However this is not quite the case, and it is precisely the libertarian tendencies of implicit nodes that come to the rescue. Whenever a node of the left pattern has the capacity to be redirected in more than one way, we introduce a fresh implict node in the right pattern of the rule to act as its "mate". With a little care, we can exploit the capacity of implicit nodes to "match anything" in order to ensure that whatever actual redirection takes place, the mate node is able to accomodate it and to rescue the commutativity of fig. 4. We thus make the syntactic form of rules reflect the actual ambiguity that comes from the semantics. For lack of space we outline the construction informally and back it up with an example or two.

Construction 5.1 Let $\langle incl : L \to P, root, Red\rangle$ be a DACTL rule.

[1] Apply construction 3.2 and call the resulting pattern $R2$. Rename each node (which is an equivalence class built up out of a single node x say of P), as the corresponding x.

[2] For every implicit node x of $R2$, introduce a mate node. Redirect all images in $R2$ of arcs of L to x, to the mate. Call the resulting pattern $R1$.

[3] For every explicit node y of $R1$ such that y is not redirected in Red but y could match the same graph node as some x for which $\langle x, t\rangle \in Red$, introduce a mate node. Redirect all images in $R1$ of arcs of L to x, to the mate. Call the resulting pattern R.

[4] Let $(i, r) : L \to R$ be the obvious maps.

A rewriting construction (let's call it the D rewriting construction), similar to P rewriting can be designed that accurately reflects DACTL rewriting. We do not describe it in detail, but instead state the relevant universal property.

Theorem 5.2 In theorem 3.7, let the rule under consideration be $(i, r) : L \to R$ as manufactured in 5.1. Replace the (unstated) reference to P rewriting, by reference to D rewriting, remove the (unstated) reference to the rigidity of the matching of the redex $g : L \to G$, and add an extra hypothesis

(6) $i(a) = r(p)$ and for all $q \in g^{-1}(g(a))$, $r(q)$ is a mate $\Rightarrow s'(g(p)) = s'(g(q))$ for any such q.

Then the theorem holds true in the modified form.

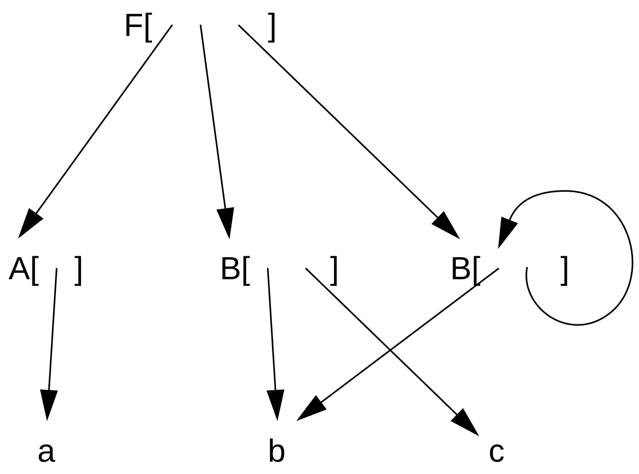


Fig. 8.(a) *L*

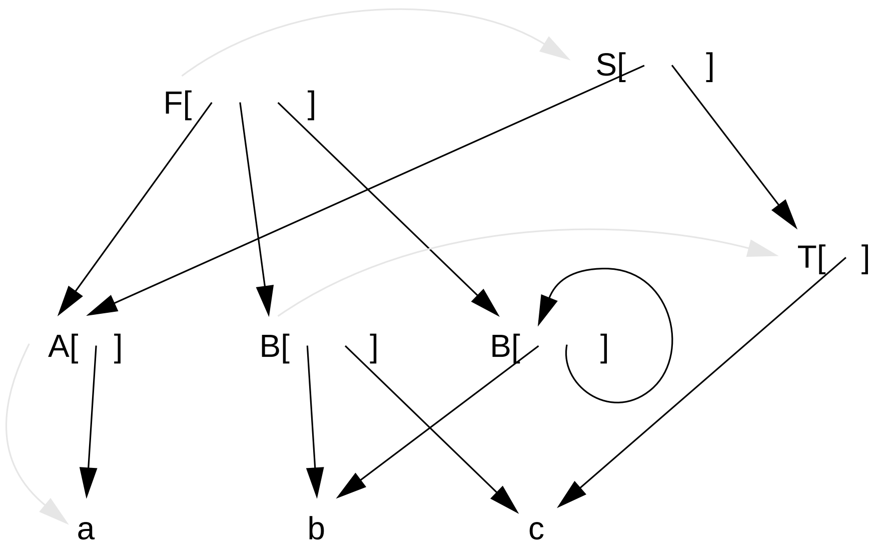


Fig. 8.(b) *P*

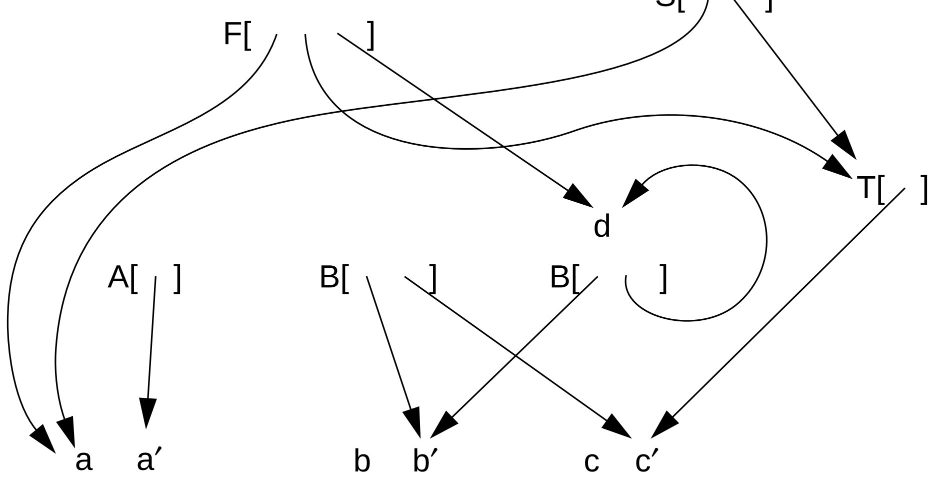


Fig. 8.(c) *R*

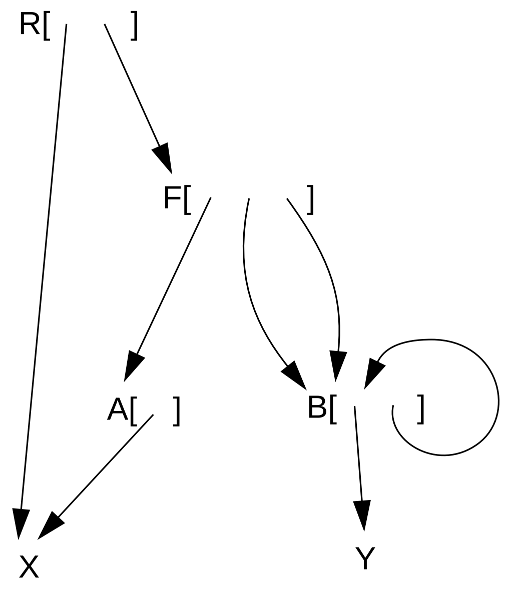


Fig. 8.(d) *G*

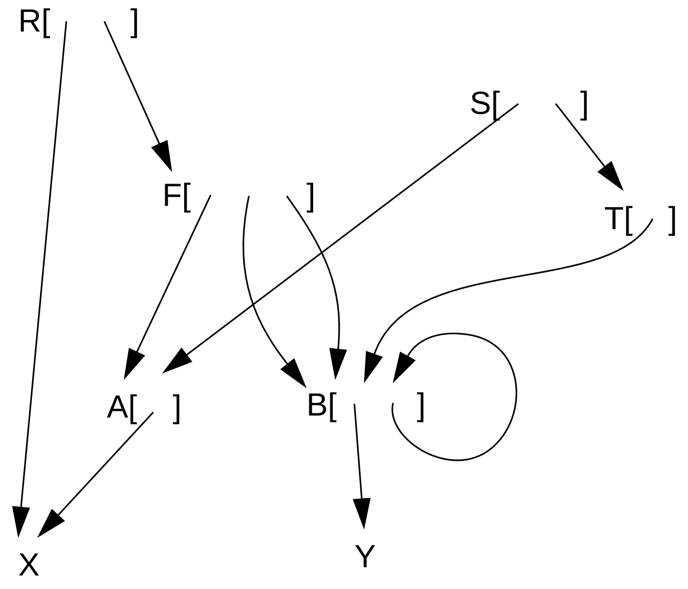


Fig. 8.(e) *G′*

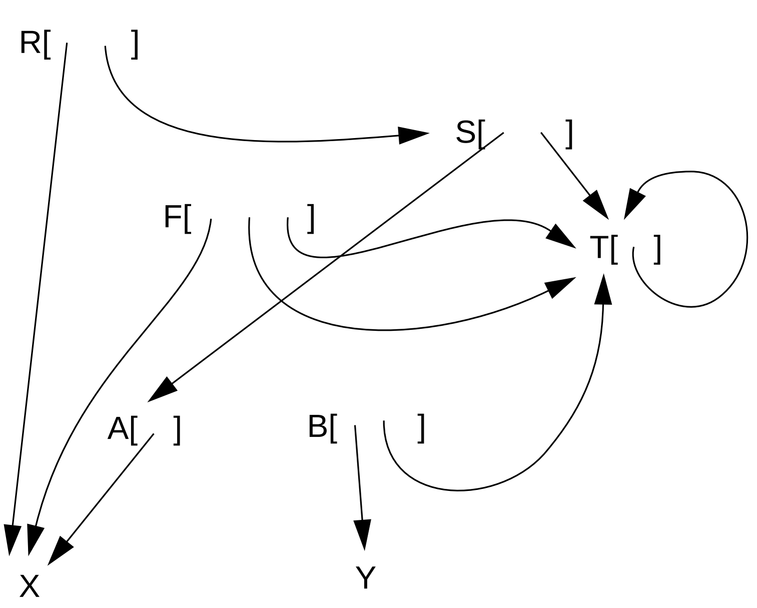


Fig. 8.(f) *H*

Theorem 5.2 describes a local form of universality that holds for D rewriting, but that does not extend to the global universality generated by a Grothendieck construction as discussed towards the end of section 3. The most important obstacle to the construction is the fact that there is an asymmetry between the patterns L and R of the rule constructed in 5.1. The pattern R contains mates while L doesn't. This blocks the translation of trivial DACTL rules (no contractum, empty redirections) to identity arrows in the base. Furthermore, even if $g : L \to G$ is a rigid redex, there is no guarantee that the corresponding $h : R \to H$ is rigid, again because of the mates. All things considered, it's remarkable that 5.2 holds at all.

Let us look at an example of D rewriting. Fig. 8 provides what we need. To avoid clutter, we have suppressed node names for all explicit nodes (which we refer to by the symbol that labels them), and suppressed the Any symbol for implicit nodes. Fig. 8.(a) gives a pattern L, which is a rooted subpattern of the pattern P of fig. 8.(b). The faint arcs in fig. 8.(b) represent the redirections, so figs. 8.(a),(b) together give a DACTL rule.

Fig. 8.(c) gives the corresponding D rewriting right pattern R. Note how a number of new implicit nodes have been introduced into R. These are the mates. Each of the original implicit nodes a, b, c has aquired a mate; they are the nodes a′, b′, c′ respectively. The images of the arcs of L which were targeted at a, b, c have been redirected to a′, b′, c′ respectively. However the images of the arcs which get redirected to a, b, c during the application of the original DACTL rule to its own full pattern remain targeted at a, b, c as before. In reality this only affects the node a. The right hand B node also aquires a mate d, because although it is not manifestly redirected itself, it could potentially match the same graph node as the left hand B node, which *is* manifestly redirected. Thus the images of the two arcs of L targeted at the right hand B node get redirected to d. The rule $(i, r) : L \to R$ is therefore given by the injection i which sends each node of L to its copy in R regardless of the presence or otherwise of mates, and the function r which can be read off from fig. 8.(c) given the invariant (HOM) — with the sole exception of r(F) which is of course S.

Figs. 8.(d) – (f) depict the rewriting of a graph G using the rule. There is obviously an instance of L in G, an instance that causes the two B nodes to match the same graph node. The DACTL construction proceeds by constructing first the graph G', and then performing the redirections to yield graph H. The D rewriting construction achieves the same effect without the intervening middle step, by characterising H as the universal completion of the diagram of fig. 5 up to isomorphism, under the conditions of theorem 5.2.

One final enigma remains to be resolved before we close this discussion of rewriting, and that is the status of the circular instance of the I combinator. The most selfevident feature of this example is that the matching $g : L \to G$ which defines the rewrite is not rigid. It seems therefore that we cannot but resort to the locally universal construction outlined above to describe it. While this is certainly possible, it is not the only thing that we can do. The circular I example actually occupies an intermediate position between the global universality of P rewriting, and the mates and local universality of D rewriting. The conditions that are imposed to make P rewriting globally universal are sufficient but not necessary to make it locally universal. The circular I example satisfies a set of weaker conditions, without having the rigidity property that gives global universality. These conditions can be summarised in the invariant (W-RIG) which stands for weak rigidity.

(W-RIG) p explicit, a implicit and $g(a) = g(p) \Rightarrow r(p) = i(p)$ or $r(p) = i(a)$.

The circular I instance of the I combinator rule in P rewriting form clearly satisfies (W-RIG). It follows that the P rewriting construction is consistent for it, and describes the locally universal properties of this rewrite. Thus the P version of the rule is given by $(i, r) : L \to R$ with L being the pattern in fig. 2.(a), R being the same pattern; $i : L \to R$ being the identity, and $r : L \to R$ having r(r) = r(a) = a (pardon the notation). Similarly, fig2.(b) gives graphs G, H for this rewrite. The other maps in the construction are the only possible ones.

6 CONCLUSIONS

In the previous sections we have described how the essentials of the rather complex and perhaps unintuitive DACTL graph rewriting model may be recast as a universal solution to a particular categorical problem, and a familiar one for category theorists at that. This is particularly gratifying for the author whose previous experience with the DACTL model gave rise to the strong intuitive feeling that despite its somewhat convoluted operational description, a robust, elegant and convincing model lay behind the drudgery of contractum-build+redirect. This is why the adjective unintuitive is used only hestitatingly in the first sentence of this paragraph. Grothendieck (op)fibrations lie behind may constructions in mathematics, and are increasingly found in theoretical computer science these days. Their usefulness in separating "syntax" from "semantics" is perhaps their most appealing feature; we use both terms in quotes since we refer to situations more general than those just involving actual syntax and semantics of programming languages — any situation where we have a collection of "objects" and for each object we have to deal with a collection of its "instances" is a good candidate for a Grothendieck construction.

References

Banach R. (1991), Term Graph Rewriting and Garbage Collection à la Grothendieck. *Submitted to TCS*.

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., Treleaven P.C. *eds*., Springer, Lecture Notes in Computer Science **259** 141-158.

Barr M., Wells C. (1990), Category Theory for Computing Science. Prentice-Hall.

Glauert J.R.W., Kennaway J.R., Sleep M.R. (1988a), Final Specification of DACTL. Report SYS-C88-11, School of Information Systems, University of East Anglia, Norwich, U.K.

Glauert J.R.W., Hammond K., Kennaway J.R., Papadopoulos G.A., Sleep M.R. (1988b), DACTL: Some Introductory Papers. Report SYS-C88-08, School of Information Systems, University of East Anglia, Norwich, U.K.

Glauert J.R.W., Kennaway J.R., Sleep M.R. (1991), DACTL: An Experimental Graph Rewriting Language., *in* Graph Grammars and their Application to Computer Science, Ehrig H., Kreowski H. Rozenberg G. *eds*., Lecture Notes in Computer Science **532** 378 - 395

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

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

Kennaway J.R. (1987), On "On Graph Rewritings". Theor. Comput. Sci. **52** 37-58.

Kennaway J.R. (1990), Implementing Term Rewrite Languages in DACTL. Theor. Comput. Sci. **72** 225-250.

Kennaway J.R. (1991), Graph Rewriting in Some Categories of Partial Morphisms, *in* Graph Grammars and their Application to Computer Science, Ehrig H., Kreowski H. Rozenberg G. *eds*., Lecture Notes in Computer Science **532** 490 - 504