A Fibration Semantics for Extended Term Graph Rewriting

R. Banach

7.1 INTRODUCTION

In this chapter, we re-examine the problem of providing a categorical semantics for the core of the general term graph rewriting language DACTL. Partial success in this area has been obtained by describing graph rewrites as certain kinds of pushout. See [Ken87, HP88, HKP88, Ken91]. Nevertheless, none of these constructions successfully describe the whole of the operational models of [BvEG$^+$87] where term graph rewriting was introduced, or of its generalization in the language DACTL itself [GKSS88, GHK$^+$88, GKS91, Ken90]. The main stumbling blocks for all of these attempts have been examples such as the I combinator `root:I[a]` $\Rightarrow$ `a` when applied to a circular instance of itself `x:I[x]`. 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 chapter 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. 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 chapter is as follows. Section 7.2 describes the free rewriting core of the original DACTL model. Section 7.3 describes the categorical construc-

Term Graph Rewriting: Theory and Practice, eds. M.R.Sleep, M.J.Plasmeijer and M.C. van Eekelen
©1993 John Wiley & Sons Ltd

tion and how it yields a Grothendieck opfibration. Surprisingly, it turns out that the garbage retention feature of DACTL is the key to the success of the construction. Section 7.4 reconsiders true DACTL rewriting and outlines the universal construction that describes it, including the circular I example. Section 7.5 concludes.

7.2 DACTL ABSTRACTED

In this section, we define the free rewriting core of DACTL rewriting, i.e. we ignore all issues pertaining both to markings, and (for simplicity) the pattern calculus. 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 **7.2.1** *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 $N \to S$,*
(3) *α is a map $N \to N^*$,*

Thus $\sigma(x)$ maps a node to the node symbol that labels it, and $\alpha(x)$ maps each node to its sequence of successors. We write $A(x)$, the arity of a node, for the domain of $\alpha(x)$. Note that $A(x)$ 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, i.e. that $c = \alpha(x)[k]$ 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, $\sigma(x)$ and $\alpha(x)$ 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 (1, -) or (2, -) to indicate their origin.

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

(ANY) $\sigma(x) = \texttt{Any} \Longrightarrow A(x) = \emptyset$.

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

DEFINITION **7.2.2** *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 **7.2.3** *A rule D is a triple (P, r, Red) where*

(1) *P is a pattern (called the full pattern of the rule).*
(2) *r is a 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) $(l', r') \in Red \Longrightarrow l'$ *is an explicit node of L.*
(RED-3) *Let $(l_1, r_1), (l_2, r_2) \in Red$. If $l_1 \neq l_2$ and 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. To highlight the left subpattern of a rule, we will often write rules as $(incl : L \to P, root, Red)$.

To define the rewriting model, we must first define the notion of homomorphism of patterns and graphs. Note that 7.2.4 serves as well for graphs as it does for patterns.

DEFINITION **7.2.4** *Let P, Z be patterns. A map $h : P \to Z$ is a homomorphism if for all explicit $x \in P$*

$$\sigma(x) = \sigma(h(x)),\ A(x) = A(h(x)),\ \text{and for all } k \in A(x), h(\alpha(x)[k]) = \alpha(h(x))[k].$$

In brief, a rewrite of a graph G (G could just as easily be a pattern) according to a rule $D = (P, r, Red)$ 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 $(l, r) \in Red$, are swung over to arrive instead at the image of the corresponding RHS. More formally we have the definitions below.

DEFINITION **7.2.5** *Let $D = (P, r, Red)$ 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 **7.2.6** *Assume the notation of 7.2.5. 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 $(1, x) \approx (2, n)$ whenever $m(n) = x$.*

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

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

LEMMA **7.2.7** *There is a homomorphism $m' : P \to G'$. Disregarding pedantry, m' extends $m : L \to G$. We call m' the extended matching.*

DEFINITION **7.2.8** *Assume the notation of 7.2.5 – 7.2.7. Let H be the graph given by*

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

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

(3)

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

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

$$\alpha_H(\{(1,x),(2,n_1)\ldots(2,n_q)\})[k] =$$

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

It is easy to show that this construction is consistent, by (RED-1) and (RED-3).

DEFINITION **7.2.9** *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 7.2.5 – 7.2.8 is the result of the rewrite of G at the redex $m(L)$ according to D.*

As an example of rewriting, we treat the circular I rewrite discussed already. In this rule the left subpattern $L =$ `root:I[a:Any]` is identical to the full pattern of the rule P, and the redirections are $\textit{Red} = \{($`root, a`$)\}$. The graph $G =$ `x:I[x]` contains an instance of L. Since $L = P$, there are no nodes to be added at the contractum building stage so $G' = G$. To perform the redirections and complete the rewrite we look for the image of $\{($`root, a`$)\}$ in G'. This is $\{($`x, x`$)\}$. 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. 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.

7.3 GRAPH REWRITING AS GROTHENDIECK OPFIBRATION

In this section we will recast some of the preceding constructions into a categorical form using a Grothendieck opfibration.

DEFINITION **7.3.1** *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) *$i : L \to R$ is a symbol/arity-preserving injection that is invertible on the implicit subpatterns of L and R, (i.e. "no new variables are introduced in the RHS of the rule").*

(RED) *$r : L \to R$ may only disagree with $i : L \to R$ on explicit nodes and satisfies*

$$\exists \text{ a homomorphism } h : L \to Z \text{ such that } h(x) = h(y)$$
$$\iff [r(x) = r(y), \text{ or } r(x) = i(x) \text{ and } r(y) = i(y)].$$

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

Identities are just pairs of identities and composition is componentwise. It is trivial to check that $\mathcal{P}$ is a category.

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.

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

(RED-P) x, y explicit and $\exists$ a homomorphism $h : L \to Z$ such that $h(x) = h(y)$
$\implies [(x, t) \in \mathit{Red} \iff (y, t) \in \mathit{Red}]$.

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

CONSTRUCTION **7.3.2** *Let $(\mathit{incl} : L \to P, \mathit{root}, \mathit{Red})$ be a DACTL$^{\mathcal{P}}$ rule. Now DACTL rewriting semantics can just as easily be applied to instances of L in patterns as to instances in graphs. So 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*

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

Modulo the pedantry of disjoint unions, we have just applied the redirections *Red* to the pattern P.

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 **7.3.3** *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 $\implies g(x) \neq g(y)$.*

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 $(1,x) \approx (2,n)$ whenever there is a $p \in L$ such that $x = g(p)$ and $n = i(p)$. Thus N_H is the pushout in $\mathcal{S}et$ of $R \xleftarrow{i} L \xrightarrow{g} G$.*

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

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

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

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

LEMMA **7.3.4** *Definition 7.3.3 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 $G \iff (j(x)_k, s(y))$ an arc of $H]$.*

LEMMA **7.3.5** *In the notation of 7.3.3, 7.3.4, $j \circ g = h \circ i$ and $s \circ g = h \circ r$.*

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

THEOREM **7.3.6** *Let $(incl : L \to P, root, Red)$ be a DACTL$^{\mathcal{P}}$ rule and let $(i,r) : L \to R$ be the corresponding $\mathcal{P}$ 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.*

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

THEOREM **7.3.7** *Using the notation of 7.3.3 – 7.3.5, 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 $G \iff (j'(x)_k, s'(y))$ 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 $g(a) = g(b) \Longrightarrow s'(g(p)) = s'(g(q))$,*

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 $\iff (\theta(p)_l, \rho(c))$ an arc of $H']$,*
(c) *(θ, ρ) extend to a homomorphism on $h(R)$,*
(d) *$j' = \theta \circ j$, $s' = \rho \circ s$, $h' = \theta \circ h = \rho \circ h$, and $\rho = \theta$ on $H - (s(G) \cup h(R))$.*

Theorem 7.3.7 shows the pushout-like nature of the $\mathcal{P}$ rewriting construction. The graph H that it creates is universal up to isomorphism among ways of completing the squares referred to in Lemma 7.3.5 according to the conditions stated.

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

DEFINITION **7.3.8** *For each object P of $\mathcal{P}$, we construct a category $\mathcal{G}^P$. The objects of $\mathcal{G}^P$ are pairs (G, g). Here G is an abstract graph and $g : P \to G$ is a rigid homomorphism. The arrows $\phi : (G, g) \to (G', g')$ of $\mathcal{G}^P$ are graph homomorphisms 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.*

DEFINITION **7.3.9** *Consider an arrow $\delta = (i, r) : L \to R$ in $\mathcal{P}$, and (G, g) an object of $\mathcal{G}^L$. Let (H, h) be the object of $\mathcal{G}^R$ 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 $\mathcal{P}$ rewriting construction, and $h : R \to H$ is the obvious homomorphism. Let $Rew^\delta((G, g)) = (H, h)$.*

LEMMA **7.3.10** *$Rew^\delta : \mathcal{G}^L \to \mathcal{G}^R$ extends to a functor.*

THEOREM **7.3.11** *There is a functor $Rew : \mathcal{P} \to \mathcal{C}at$ such that*

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

The existence of $Rew : \mathcal{P} \to \mathcal{C}at$ leads immediately to the construction of the Grothendieck category $\mathcal{G}(\mathcal{P}, Rew)$. The objects of $\mathcal{G}(\mathcal{P}, Rew)$ are pairs $((G, g), L)$ where L is an object of $\mathcal{P}$ and (G, g) is an object of $Rew(L)$. We can write such objects as $(g : L \to G)$. The arrows of $\mathcal{G}(\mathcal{P}, Rew)$ are pairs $(\phi, \delta) : (g : L \to G) \to (h : R \to H)$ where $\delta = (i, r) : L \to R$ is an arrow of $\mathcal{P}$, and $\phi : Rew^\delta(G) \to H$ is an arrow of $\mathcal{G}^R$. In slightly less combinatorial terms, an arrow (ϕ, δ) of $\mathcal{G}(\mathcal{P}, Rew)$ can be viewed as an abstract $\mathcal{P}$ rewrite of a redex $g : L \to G$ by a rule $\delta = (i, r) : L \to R$ giving $Rew^\delta((G, g))$, 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 $G \iff (j(x)_k, s(y))$ an arc of $H]$. Such a pair (j, s) is strictly speaking a different thing from (ϕ, δ), but we will overlook this.

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

$$(\chi, \epsilon) \circ (\phi, \delta) : (g : L \to G) \to (k : N \to K) = (\chi \circ Rew(\epsilon)(\phi), \epsilon \circ \delta)$$

Note that compared to our $\mathcal{P}$ rewriting construction, the arrows of $\mathcal{G}(\mathcal{P}, Rew)$ 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. Readers unhappy about this can merely stick to the special case where all the $\mathcal{G}^P$ categories are discrete. The arrows of $\mathcal{G}(\mathcal{P}, Rew)$ will be called rewrites, or $\mathcal{G}$ rewrites if we wish particularly to distinguish them from $\mathcal{P}$ rewrites and DACTL rewrites. Of course from a categorical viewpoint, we might justifiably prefer them to be called corewrites or oprewrites.

Continuing the development, $\mathcal{G}(\mathcal{P}, Rew)$ is a fibered category. The fibers are the $\mathcal{G}^L$ categories and the projection $F : \mathcal{G}(\mathcal{P}, Rew) \to \mathcal{P}$ takes objects $(g : L \to G)$ to L, and arrows (ϕ, δ) to δ. This is an example of the canonical duality between split (op)fibrations and the Grothendieck categories built using the Grothendieck construction. Powerful universality properties that extend the universality properties that hold for pushouts pertain to this situation. We will not stop to describe them. For a little more discussion of these issues, see the preliminary version of this chapter in [Ban91], or the full version in [Ban93]. For a reasonably accessible description of the Grothendieck construction in its abstract form see [BW90].

7.4 TRUE DACTL REWRITING

In section 7.3 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$^{\mathcal{P}}$. The main problem encountered in applying the constructions of section 7.3 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 implicitly determines targets for redirection. Thus if $(x, t) \in Red$, and x and y both match the same graph node of the redex, but for no u do we have $(y, u) \in Red$, 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 $(x, t) \in Red$. Again the explicit redirection wins over the unstated identity redirection. Such phenomena prevent the commutativity needed for Lemma 7.3.5, and explain why the DACTL$^{\mathcal{P}}$ sublanguage contained specific conditions to prevent such behavior.

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 accommodate it and to rescue the required commutativity. We thus make the syntactic form of rules reflect the actual ambiguity that comes from the semantics. For lack of space we just outline the construction informally.

CONSTRUCTION **7.4.1** *Let $(incl : LP, root, Red)$ be a DACTL rule.*

[1] *Apply construction 7.3.2 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 $(x,t) \in$ 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 us call it the D rewriting construction), similar to $\mathcal{P}$ rewriting can be designed that accurately reflects DACTL rewriting. We do not describe it in detail, but instead state the relevant universal property.

THEOREM **7.4.2** *In Theorem 7.3.7, let the rule under consideration be $(i,r) : L \to R$ as manufactured in 7.4.1. Replace the (unstated) reference to $\mathcal{P}$ 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 $\Longrightarrow s'(g(p)) = s'(g(q))$ for any such q.*

Then the theorem holds true in the modified form.

Theorem 7.4.2 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 7.3. 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 7.4.1. The pattern R contains mates while L does not. 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 is remarkable that 7.4.2 holds at all.

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 self evident 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 $\mathcal{P}$ rewriting, and the mates and local universality of D rewriting. The conditions that are imposed to make $\mathcal{P}$ 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 summarized in the invariant (W-RIG) which stands for weak rigidity.

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

The circular I instance of the I combinator rule in $\mathcal{P}$ rewriting form clearly satisfies (W-RIG). It follows that the $\mathcal{P}$ rewriting construction is consistent for it, and describes the locally universal properties of this rewrite. Thus the $\mathcal{P}$ version of the rule is given by $(i,r) : L \to R$ with L and R the same pattern described in the introduction; i being the identity, and r having $r(\texttt{root}) = r(\texttt{a}) = \texttt{a}$.

7.5 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 gut 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" (contained in the base category) from "semantics" (in the Grothendieck category above) 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

[Ban91] R. Banach. DACTL rewriting is categorical. *Proc. SemaGraph-91* Vol. II, Nijmegen Tech. Rep. 91-25, Dept. of Informatics, University of Nijmegen, 1991.

[Ban93] R. Banach. Term graph rewriting and garbage collection using opfibrations. Theor. Comput. Sci., to appear.

[BvEG+87] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R Kennaway, M.J. Plasmeijer and M.R. Sleep. Term graph rewriting. *Proc. PARLE-87* Vol. II, LNCS 259, pp. 141-158, Springer-Verlag, 1987.

[BW90] M. Barr, C. Wells. *Category Theory for Computing Science.* Prentice-Hall, 1990.

[GKSS88] J.R.W. Glauert, J.R. Kennaway, M.R. Sleep, G.W. Somner. *Final Specification of DACTL.* Internal Report SYS-C88-11, School of Information Systems, University of East Anglia, Norwich, UK.

[GHK+88] J.R.W. Glauert, K. Hammond, J.R. Kennaway, G.A. Papdopoulos, M.R. Sleep. *DACTL: Some Introductory Papers.* School of Information Systems, University of East Anglia, Norwich, UK.

[GKS91] J.R.W. Glauert, J.R. Kennaway, M.R. Sleep. DACTL: An experimental graph rewriting language. *Graph Grammars and their Application to Computer Science*, LNCS 532, pp. 378-395, Springer-Verlag, 1991.

[HKP88] A. Habel, H-J. Kreowski, D. Plump. Jungle evaluation. *Proc. Fifth Workshop on Specification of Abstract Data Types*, LNCS 332, pp. 92-112, Springer-Verlag, 1988, *also Fund. Inf.* 15, pp. 37-60, 1991.

[HP88] B. Hoffmann, D. Plump. Jungle evalulation for efficient term rewriting. *Proc. International Workshop on Algebraic and Logic Programming*, Mathematical Research 49, Akademie-Verlag, Berlin, 1988.

[Ken87] J.R. Kennaway. On "on graph rewritings". *Theor. Comput. Sci.* 52, pp. 37-58.

[Ken90] J.R. Kennaway. Implementing term rewrite languages in DACTL. *Theor. Comput. Sci.* 72, pp. 225-250.

[Ken91] J.R. Kennaway. Graph rewriting in some categories of partial morphisms. *Graph Grammars and their Application to Computer Science*, LNCS 532, pp. 490-504, Springer-Verlag, 1991.