Theoretical Computer Science 131 (1994) 29–94
Elsevier

Fundamental Study

Term graph rewriting and garbage collection using opfibrations

R. Banach

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

Communicated by M. Nivat
Received January 1992
Revised June 1993

Abstract

Banach, R., Term graph rewriting and garbage collection using opfibrations, Theoretical Computer Science 131 (1994) 29–94.

The categorical semantics of (an abstract version of) the general term graph rewriting language DACTL is investigated. The operational semantics is reformulated in order to reveal its universal properties. The technical dissonance between the matchings of left-hand sides of rules to redexes, and the properties of rewrite rules themselves, is taken as the impetus for expressing the core of the model as a Grothendieck opfibration of a category of general rewrites over a base of general rewrite rules. Garbage collection is examined in this framework in order to reconcile the treatment with earlier approaches. It is shown that term rewriting has particularly good garbage-theoretic properties that do not generalise to all cases of graph rewriting and that this has been a stumbling block for aspects of some earlier models for graph rewriting.

Contents

Correspondence to: R. Banach, Computer Science Department, University of Manchester, Manchester, M13 9PL, UK. Email: banach@cs.man.ac.uk.

SSDI 0304-3975(93)E0132-N

1. Introduction

Since Wadsworth's [32] work, the manipulation of graphs has been seen as one source of efficient implementations of functional programming languages. More recently, there have been efforts to put this often pragmatic work onto a more sound mathematical basis. Thus, in [4, 18, 19, 21, 23, 28–31], various formulations of term graph rewriting appear. Also, in [12, 13, 14] an experimental language for general term graph manipulation, DACTL, is described.

In the work of Raoult and Kennaway in particular, rewrites are viewed as pushouts in a category where objects are graphs and arrows are partial morphisms. By contrast, in the work of Plump and coauthors, rewrites are viewed as double pushouts, after the

algebraic graph grammar tradition of Ehrig and coworkers (see e.g. [9, 10, 27]). Kennaway [23] has given a general framework of hypergraphs and partial morphisms in which both of these approaches may be placed. More recently, Lowe and Ehrig [25] and van den Broek [7] have offered formulations of graph rewriting that can be compared to that of Kennaway [23].

On the other hand, work of Barendregt et al. [4], and its expression in the concrete general term graph rewriting language DACTL, has taken a much more operational view of the semantics of a graph rewrite. This operational view coincides with the more abstract algebraic view in sufficiently benign cases, but it is fair to say that none of the categorical approaches has thus far given the semantics of all DACTL rewrites in an entirely satisfactory way. The communal graveyard of these approaches has been the I combinator I[x]⇒x, when applied to the circular graph a:I[a]. DACTL quite clearly states that the rewrite is to be a null operation, which seems quite natural in a graph-oriented world, whereas the other constructions either fail to give a meaning to the rewrite or declare its result to be an unlabelled node.

The purpose of the present paper is to give a categorical semantics for DACTL rewrites including a treatment of the troublesome example above. Why bother doing this at all? For the author, an elegant universal formulation helps clarify the meaning of a concrete definition, particularly a complicated one, by using the associative nature of categorical constructions to bring out clearly the compositionality of potentially diverse aspects of a rather "flat" operational construction. By abstracting away from irrelevant detail (in the case of graph rewriting, much irrelevant detail arises from specific models of disjoint union), a simpler and more convincing picture usually emerges.

To achieve our aim we have to depart from the earlier paradigms. The problems with these seem to lie in two areas. The first problem concerns the placing of both the rules and the graphs that they act on within the same category. This leads naturally to the view that a rewrite should be some kind of pushout, but the fact that it becomes necessary to impose different conditions on the two arms of the pushout to establish its existence is a clue that all is perhaps not well. What one does when one places both the rules and the graphs that they act on in the same category is to blur the distinction between syntax and semantics. More explicitly, rules are intrinsically syntactic objects – they give us instructions about what to do during an execution of the system. Graphs, conversely, are intrinsically semantic objects – they are the objects that are transformed by the rules. While treating them as the same kind of thing is tempting, considering the term rewriting origins of much of the work on graph rewriting, it is liable to lead to the kind of difficulties known since Tarski's investigations in the 1920s.

The second problem area that causes trouble in the coventional paradigms is garbage collection. In the purely term world, where the objects being manipulated are trees, it is clear, for topological reasons, precisely what garbage is created when a rewrite is performed, and this is implicitly incorporated into the operational semantics of term rewriting. When genuine graphs are the computational objects

involved, and sharing of subgraphs can occur in an essentially arbitrary manner, the same is no longer the case. At least two attitudes to this situation are permissible depending on the level of defensiveness of the model in question. The most defensive attitude is not to discard anything from the computation, even things that are obviously garbage, such as nodes that are rewritten by a rule. This is the position for the DACTL model which thus features garbage retention rather than garbage collection; in DACTL, collection of genuine garbage is left to the implementation.

A less defensive attitude removes those nodes which are definitely garbage, more specifically the rewritten nodes. This is the position of the categorical models above. In principle, static analysis of a rule system might reveal other parts of graphs which are garbage and which thus may be safely removed by the action of the rule, but to the author's knowledge no system goes this far.

Surprisingly, the garbage retention feature of DACTL turns out to be an inspired design decision as far as the categorical semantics of this paper is concerned. As will become apparent in Section 6, the semantics would have been considerably more awkward to describe had any garbage collection been incorporated in the structure of a rule. It is garbage retention more than anything else that allows the smooth treatment of the circular I example, both in (the operational semantics of) DACTL itself and in the categorical description below.

In outline, the rest of this paper is as follows. Section 2 gives a brief and informal introduction to DACTL for purposes of orientation, while Section 3 defines the "traditional" operational semantics of a DACTL rewrite more formally. Strictly speaking, we define an abstracted version of the semantics, leaving out the control markings and pattern operators that are used to influence reduction strategy and rule selection in real DACTL. Some examples are given and their reformulation according to different rewriting models is discussed. Section 4 reformulates and generalises this semantics so that a rewrite appears as the universal completion of a commuting square, just as for a pushout. However, this square does not consist of objects and arrows from a single category. Section 5 explains why not, showing that the obvious category built on the redirection couple idea from Section 4 gives rise to an inappropriate pushout-based rewriting model.

Now some of the operational aspects of DACTL, in particular the priority mechanism for determining the targets of redirections, prevent the extension of the universality of Section 4 to a truly categorical treatment. But when these aspects are suitably circumscribed, a categorical treatment results in which rewrite rules appear as arrows in the base category of an opfibration, and rewrites themselves appear as arrows in the associated Grothendieck category. The universal properties of the pushouts of earlier treatments are replaced by the universal properties of the split opfibration inherent in the Grothendieck construction. This is described in Section 6.

Section 7 explains how one can easily pass from the world of "all conceivable rewrites" to those belonging to a given system of interest. To allow a fair comparison with preceding formulations, the problem of garbage has to be faced, and a categorical treatment of garbage is given within the confines of the Grothendieck construction in

Section 8. Section 9 contains a relatively brief treatment of term rewriting using the preceding methods, pointing out especially that the strong topological properties of terms lead to a particularly clean treatment of garbage that fails for graphs in the general case. Term rewriting is thus best done "without garbage", while graph rewriting is best done "with garbage". Section 10 is a conclusion and also contains a mild "manifesto" statement to the effect that all computational models have features that enable similar constructions to be made.

2. A brief introduction to DACTL rewriting

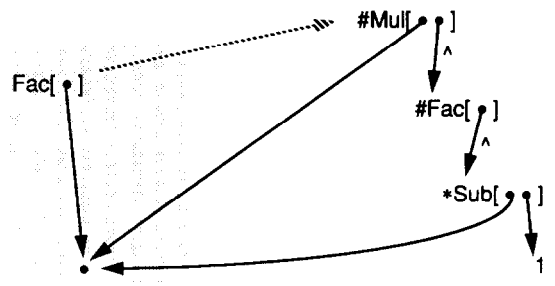
DACTL grew out of a desire to place many of the low-level operational features found in implementations of declarative languages onto a more formal basis. This inevitably means dealing with graphs rather than terms, since these implementations often feature a sharing of subexpressions, and so term rewriting becomes superseded by graph rewriting. The desire to keep as close as possible to reality meant that the nodes of the graph look as much like parts of a term as is reasonable. So term graphs, as they were called, have nodes which are labelled with a symbol, and have a sequence of out-arcs to other nodes in the graph.

The dynamic feature of term rewriting, substitution, is handled by the elegant notion of arc redirection, whereby all the in-arcs pointing to a node may be redirected to point at some other node. Of course, for this to make sense, the other node has to be in the graph already, and so when substitution by a completely new subgraph is desired, redirection has to be preceded by formally incorporating the new material into the existing graph: contractum building. It is the universal properties of this two-phase construction that we will study in depth for most of this paper.

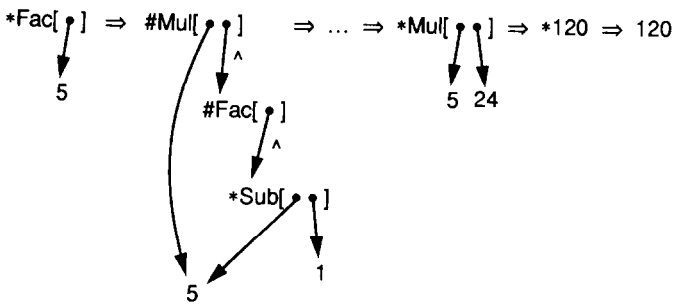
Redirection turns out to be a versatile concept, and can be applied to any node of the graph, not just roots of redexes. This generalises the term graph rewriting model, allowing many ideas from imperative programming to be smoothly expressed in the same formalism as is used for the declarative side. All in all a flexible methodology for describing computation results.

Let us now examine a couple of examples. Since our concern in this paper is with semantics, we eschew discussion of the concrete syntax of DACTL, and draw term graphs directly. Nevertheless, some conventions from DACTL's syntax creep in to our pictures.

Figure 1(a) illustrates the non-base-case rule for factorial. The shaded part is the left-hand side, which is pattern-matched to the graph being evaluated. The nonshaded part is the contractum, a copy of which is glued to the redex when a match is found. The broken arrow represents the single redirection of this rule, which stipulates that, in the redirection phase, in-arcs of Fac are to be redirected to the new Mul node. The markings (#, ^, *) are DACTL's mechanism for controlling evaluation strategy. Briefly, a node marked with a number of #'s must wait for that number of "notifications" to arrive from those of its children to which it is connected along out-arcs



(a)



(b)

Fig. 1. (a) Rule for Fac[n] ⇒ # Mul[n ^ # Fac[^ * Sub[n1]]]; (b) an evaluation of Fac[5].

marked with ^. Each time such a notification arrives, the ^ on the arc in question is deleted, and the number of #'s on the parent is decremented. When the latter reaches zero, the parent is marked with *, becoming active.

DACTL stipulates that only active nodes may be roots of redexes, and that selection of which active node to reduce next is nondeterministic. When an active node is selected for rewriting, one of two things happens. If there is a rule, the root of whose LHS will match at the given active node, the active marking is removed from the root of the redex and rule execution proceeds as indicated above. If there is no such rule, the active marking is removed as in the other case, and notifications are sent up along all ^-marked in-arcs of the node. Normally, the parents of these nodes at the tails of such in-arcs are waiting for these notifications by being #-marked.

Note that the metaphor of sleeping while waiting for a subcomputation to complete, before being woken and continuing, is fairly universal in computing, being found in implementations of declarative languages, in the procedure call/return mechanism of imperative languages, and at higher levels of abstraction too. This at least partly explains DACTL's suitability for a wide variety of applications.

Figure 1(b) illustrates some of the stages of the evaluation of Fac[5], where, for clarity, garbage is collected "as we go". In reality, DACTL does not say anything at all about garbage, a fact we will draw out much further later on. Despite this seemingly glaring shortcoming, DACTL is a "real language" in that it provides mechanisms for things like rule selection, inbuilt operators for common data types such as integers and booleans, a module discipline, interfacing between modules and to the outside world, and so on. Discussing these matters would, however, take us far from the point of this paper.

Figure 2 shows some basic rules for semaphore handling. Among other things, it illustrates the utility of nonroot redirection, particularly in imperative programming. Figure 2(a) shows how a free semaphore is claimed, and how the success of the operation is communicated to the waiting parent. (It is assumed that symbol Succeeded has no rules.) In Fig. 2(b), if the semaphore is Busy, the suspension mechanism using # and ^ markings achieves the required semantics. In Fig. 2(c), the active Free symbol (also assumed to have no rules) will, when it rewrites, notify any suspended GetSemaphores, which will then have an opportunity to claim the resource required.

3. DACTL **abstracted**

In this section we restate the definition of a DACTL rewrite more formally. As mentioned before, we concentrate exclusively on the two-phase construction that

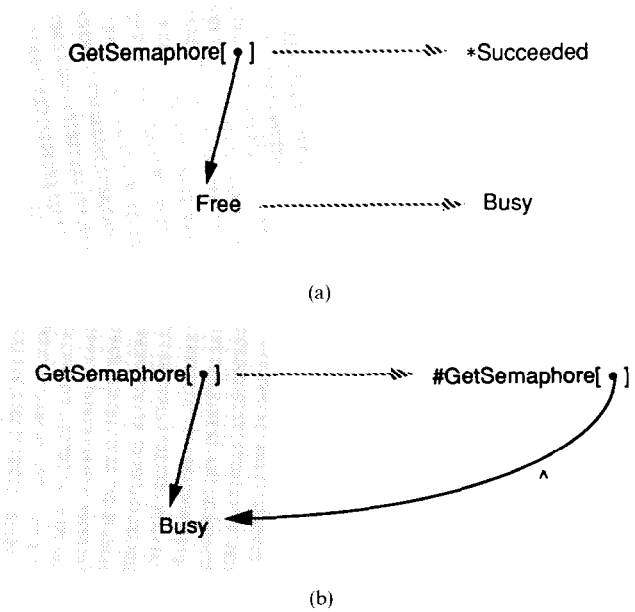
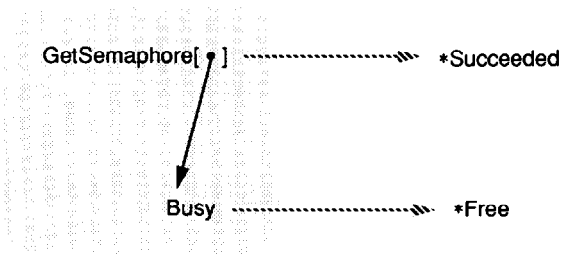


Fig. 2. (a) Rule for obtaining a free semaphore; (b) rule for waiting for a busy semaphore.



(c)

Fig. 2. (c) Rule for releasing a semaphore.

generates the new graph structure from the old. Readers already familiar with DACTL might find the terminology used slightly unconventional. This is in order to conform with the usage of later sections.

3.1. The DACTL rewriting model

Suppose an alphabet of node symbols $\mathscr{S}=\{S, T\ldots\}$ is given.

Definition 3.1. A term graph (or just graph) G is a triple $\langle N, \sigma, \alpha\rangle$, where
(1) N is a set of nodes,
(2) $\sigma: N\to\mathscr{S}$ is a map which labels each node,
(3) $\alpha: N\to N^*$ maps each node to its sequence of successors (or children).

We write $A(x)$, the arity of a node x, for the domain of $\alpha(x)$. Note that $A(x)$ is a set of consecutive natural numbers starting at 1, or is empty. When dealing with more than one graph (or pattern – see below), we subscript the objects defined in cases (1)–(3) above with the name of the graph in question, to avoid ambiguity. Also, we allow ourselves to write $x\in G$ instead of $x\in N(G)$ or $x\in N_G$, 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 c is the kth child of p, i.e. that $c=\alpha(p)[k]$ for some $k\in A(p)$.

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

$$\text{(ANY)}\quad \sigma(x)=\mathsf{Any} \Rightarrow A(x)=\emptyset,$$

i.e. x has no successors.
A node labelled with Any is called implicit and a node labelled with a member of $\mathscr{S}$ is called explicit.

Definition 3.2. 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.)

Patterns have a natural notion of homomorphism.

Definition 3.3. Let P, Q be patterns. A node map $h: P \to Q$ is a homomorphism iff for all explicit nodes

(1) $\sigma(x) = \sigma(h(x))$, i.e. h is label-preserving,

(2) $A(x) = A(h(x))$, i.e. h is arity-preserving,

(3) for all $k \in A(x)$, $h(\alpha(x)[k]) = \alpha(h(x))[k]$, i.e. h is order-preserving.

In addition, a homomorphism is strict iff the image of every implicit node is implicit. Note that Definition 3.3 serves just as well for graphs as it does for patterns since graphs just have all their nodes explicit. Homomorphisms are also called matchings.

Equipped with homomorphisms as arrows, patterns form a category $\mathscr{H}$, but this will be of marginal importance for the time being.

Definition 3.4. A rule D is a triple $\langle P, root, Red \rangle$, where:

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

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

(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 the nodes of P.

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

(RED-3) $\langle l_1, r_1 \rangle, \langle l_2, r_2 \rangle \in Red$ and $\exists$ a pattern Z and a homomorphism $h: L \to Z$ such that $h(l_1) = h(l_2) \Rightarrow r_1 = r_2$.

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 slightly stronger but more readable (RED-3'):

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

Below we will make much use of disjoint unions when making explicit constructions of graphs, and to be completely unambiguous about this, each element of the union will be tagged with the flag 1 or 2 to indicate its origin. Elements of disjoint unions are thus pairs $\langle 1, - \rangle$ or $\langle 2, - \rangle$. When speaking more informally, we will be less pedantic.

In brief, a rewrite of a graph G according to a rule $D = \langle P, root, Red \rangle$ proceeds through three stages. (Note: G could just as easily be a pattern.) 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 domain of the homomorphism to the whole of P. This is contractum building. 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. This is redirection.

Now we come to the precise specification of DACTL rewriting. The definitions below are the formal counterparts of our intuitive description in Section 2.

Definition 3.5 (*Redex*). Let $D=\langle P, root, Red\rangle$ be a rule. Let G be a graph. Let L be the left subpattern of P. We will use the alternative notation $D=\langle incl: L\to P, root, Red\rangle$ below whenever convenient, where *incl* is the inclusion of L into P. Let $m: L\to G$ be a matching. Then $m(L)$ is a redex in G and $m(root)$ is the root of the redex. The homomorphism m is called a matching of L to G at $m(root)$.

The next definition may be more easily assimilated by noting that it is just a version of the pushout construction in the category $\mathscr{H}$ mentioned above.

Definition 3.6 (*Contractum building*). Assume the notation of Definition 3.5. Let the graph G' be given by:

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

(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' acquires symbols in such a way as to agree with both G and P. The last case ensures that the representative in G' of an implicit node $n\in P$ acquires a symbol according to its image under m. Any explicit nodes in the same equivalence class will be labelled with the same symbol because m is a homomorphism.

(3) $\alpha_{G'}(\{\langle 1, x\rangle\})[k]=\{\langle 1, \alpha_G(x)[k]\rangle \ldots\}$,
$\alpha_{G'}(\{\langle 2, n\rangle\})[k]=\{\langle 2, \alpha_P(n)[k]\rangle \ldots\}$,
$\alpha_{G'}(\{\langle 1, x\rangle, \langle 2, n_1\rangle \ldots \langle 2, n_q\rangle\})[k]=\{\langle 1, \alpha_G(x)[k]\rangle \ldots\}$.

Thus, G' acquires arcs so as to agree with both G and P. The ellipses on the RHSs of these cases indicate that the equivalence classes concerned need not be singletons. That this is consistent is again assured by the fact that m is a homomorphism.

Definition 3.7 (*Extended matching*). The homomorphism $m': P\to G'$ that exists by virtue of Definition 3.6 being a pushout is called the extended matching.

The redirection phase of a DACTL rewrite, defined next, does not have such a tidy formulation categorically. That is the point of this paper.

Definition 3.8 (*Redirection*). Assume the notation of Definitions 3.5–3.7. Let H be the graph given by

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

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

The essence of redirection is the next clause. It says that, if a node x of G' is the image under the extended matching of the LHS of a redirection pair $\langle u, y\rangle \in Red$, then all its in-arcs are to be redirected to the image of y.

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

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

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

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

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 $\langle u, y\rangle \in Red$. (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 3.9. Let G be a graph, D a rule and m a matching of the left subpattern L of D to G. The graph H constructed via Definitions 3.5–3.8 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 [13] in minor details. In particular, the final specification has moved from DACTL's original position, particularly in the area of which combinations of redirections are permissible, partly at least, as a result of the influence of the categorical semantics of papers already cited. Since our objective is a categorical semantics for as "pure" a form of DACTL rewriting as is possible, our starting model is closer in spirit to slightly earlier versions of DACTL. The reader unacquainted with these subtle nuances will not find this an obstacle to understanding the rest of this paper. In Section 4 we show in what sense the DACTL model is universal.

3.2. Two examples

We present a couple of examples of DACTL rewriting. In the sequel, the notation $z{:}A[\cdots]$ refers to a node z with label A and successors enclosed in brackets.

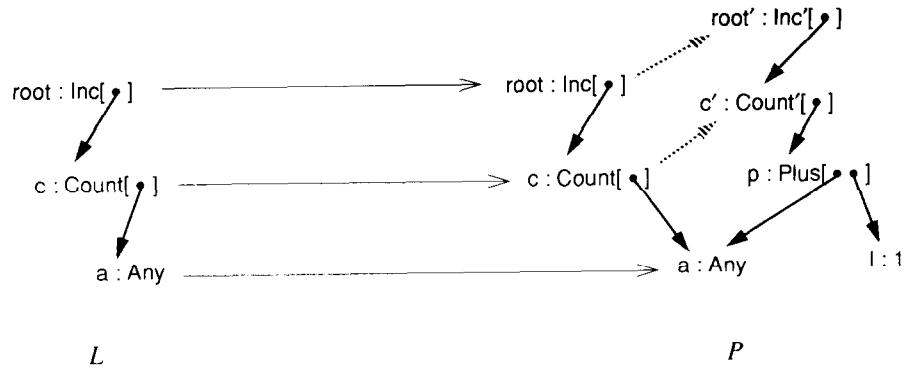


Fig. 3. A rule $\langle incl: L \rightarrow P, root, Red \rangle$.

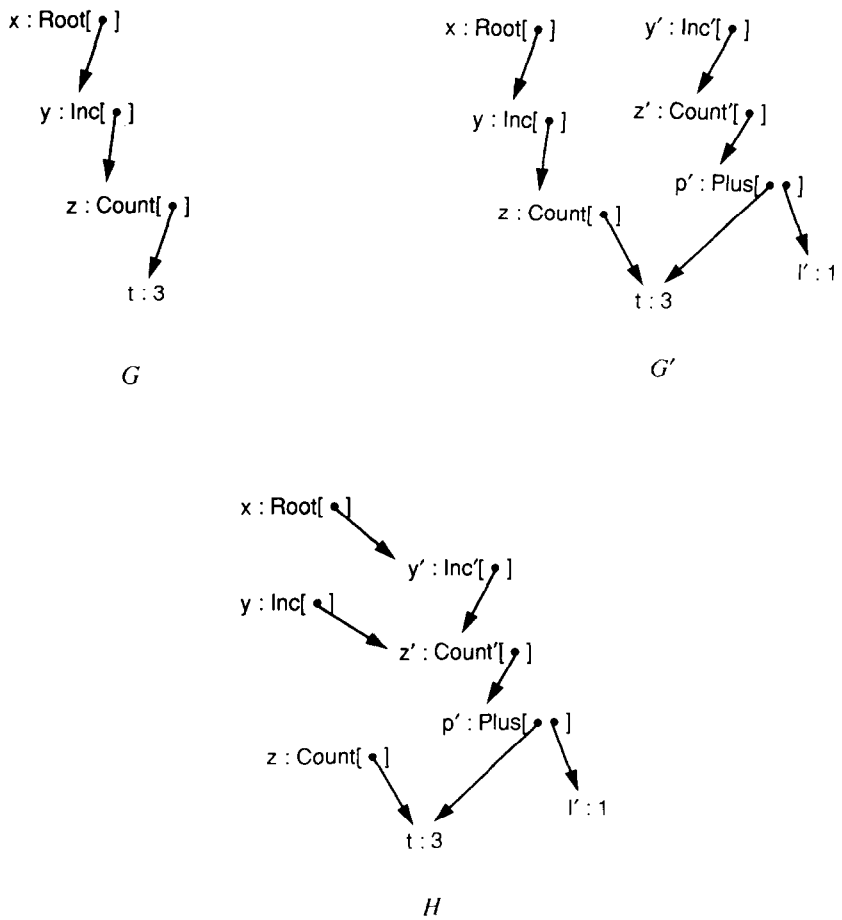


Fig. 4. A rewrite of a graph G to H, via the intermediate form G'.

$L = P$ $\qquad$ $G = G' = H$

Fig. 5. The rule for I, I[a] ⇒ a and the circular I rewrite.

Example 3.10. In Figs. 3 and 4 we illustrate a rule D and its action on a graph G. The names of the patterns and graphs illustrated there are intended to conform to our notation above. Thus, Fig. 3 illustrates the rule, with the inclusion $incl: L \to P$ shown explicitly and with the redirection pairs *Red* indicated by dashed arrows in P. Figure 4 shows the two stages of rewriting: first the intermediate graph G', the obvious pushout of P and G, and then the final result after redirection, H.

Example 3.11. As a further example we treat the circular I rewrite mentioned before. Figure 5 illustrates both the rule and its application. In this rule the left subpattern of the rule is identical to the full pattern, and there is just one redirection ⟨root, a⟩. The graph G contains a circular instance of L. Since $L = P$, there are no nodes to be added at the contractum building stage, so $G = G'$. In performing the redirection, we notice that its source and target nodes are the same, so it is a null action. So $G = G' = H$.

3.3. Review and comparison with other models

Let us review how the DACTL construction works. In the category $\mathscr{H}^0$ whose objects are term graphs and whose arrows are graph homomorphisms, pushouts exist. his follows essentially from the construction of pushouts in $\mathscr{S}et$. If one admits patterns as objects of the category (and thus pattern homomorphisms or matchings as the arrows), pushouts no longer necessarily exist. This is because in the diagram $B \leftarrow A \rightarrow C$, an implicit node of A may be mapped to roots of two incompatible subgraphs of B and C. However, if both arrows are strict, or if the pair is a co-strict pair (which is defined by the property that at most one of the pair of images in B and C of any implicit node of A is explicit), then the pushout can be constructed (see [23] for a more detailed discussion).

We have remarked that the contractum building phase of a DACTL rewrite is just such a pushout, and now it is clear why this works: we have imposed sufficient syntactic constraints on the situation to ensure that the two arrows $incl: L \to P$ and $m: L \to G$ form a co-strict pair.

It is harder, however, to accommodate the redirection phase into the same framework. The approach hitherto has been to weaken the notion of homomorphism to that of partial morphism; in other words, the homomorphic conditions given by Definition 3.3(1)–(3) are only required to hold at a selected subset of nodes.

Note that we are already using such a notion to allow the matching of implicit nodes to arbitrary nodes, but the desire to accommodate redirections forces us to allow the homomorphic condition to fail under wider circumstances, and in the corresponding map $p: L \to P$ of a rule, p is no longer a pure injection (and so cannot be optimised out of the data specifying a rewrite rule as we did in the description of DACTL rewrite rules above).

Now, the nodes at which p is not homomorphic are places where the graph is being rewritten. The passage to partial morphisms exaggerates the asymmetry of the conditions placed on the two arms of any pushout square which is required to describe the rewriting process. The arm specifying the rule is a strict partial morphism, while the arm specifying the redex is just a total homomorphism. This is for the Kennaway [21] model or the Lowe and Ehrig [25] model; similar remarks apply for the Hoffmann and Plump [19] model. It seems a bit of a strain to regard these constructions as true pushouts.

Figure 6 gives the rule of Example 3.10 in the framework of Kennaway [21], while Fig. 7 gives the same example in the framework of Hoffmann and Plump [19]; however, it should be pointed out that, since the rule features subroot redirection, Fig. 7 cannot be strictly regarded as a jungle rewrite rule. The most significant feature of the pattern P of Fig. 6 is that the garbage nodes root : Inc[···] and c : Count[···] of the pattern L are missing.

There is no separate redirection data in the model of Kennaway [21], so the presence of any such node in that graph would have been a mandate to construct a copy during the construction of the pushout; similarly for the Hoffmann and Plump

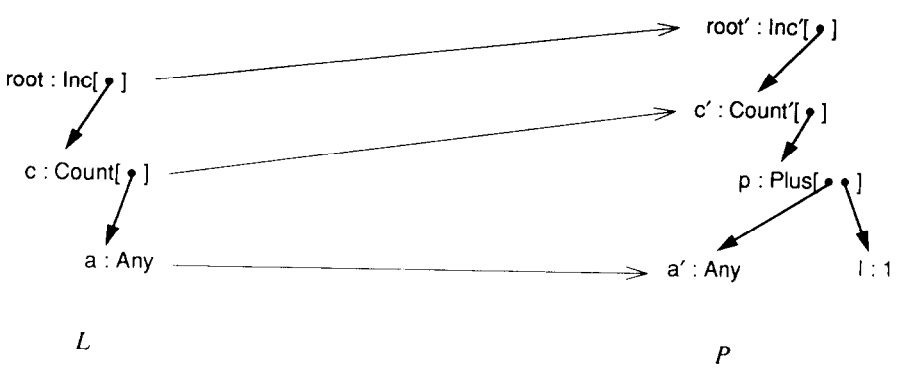


Fig. 6. The rule of Fig. 4 in the style of Kennaway [21].

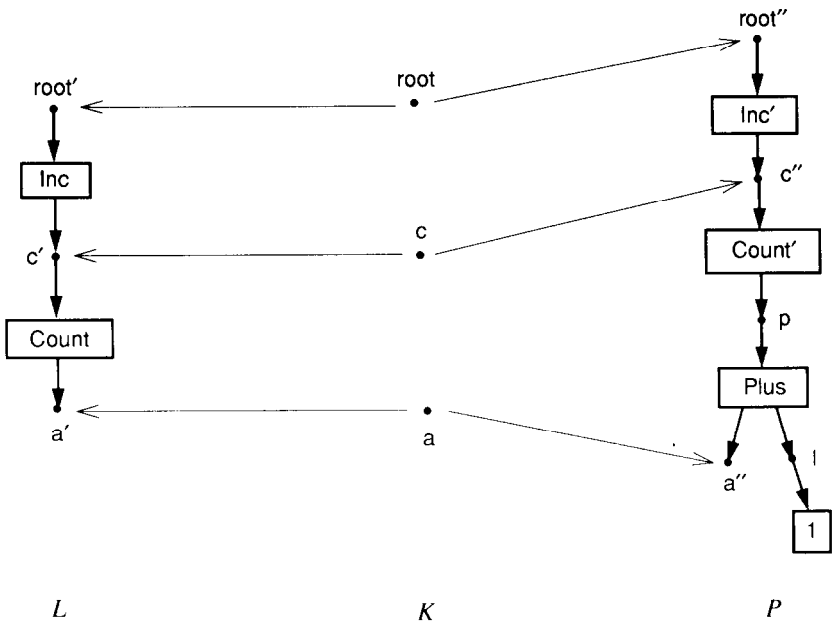


Fig. 7. The rule of Fig. 4 in the style of Hoffmann and Plump [19].

[19] version of the rule. The garbage hyperedges Inc and Count have disappeared from the right graph of the rule. Both observations are concomitant with the fact that a single function gives the relationship between the left- and right-hand sides of a rule. (This is obvious in Fig. 6, and can be reconstructed in Fig. 7 by inverting the $K \rightarrow L$ injection on nodes.)

Now a single function cannot both indicate that a node in the codomain is the relic of a given node of the domain and indicate that some different node in the codomain represents the node that the given domain node is to be rewritten to. Usually this does not matter since only one or the other of these roles is required of a particular point in the mapping. However, in the circular 1 example, both roles are required for the node root of the pattern L of Fig. 5.

DACTL semantics can accommodate this behaviour since the injection and the redirection data are specified separately. We see that this separation is at the heart of the success of the DACTL model here. The other models cannot do this. Note also that garbage retention played a vital role in ensuring by syntactic means that there was no ambiguity about whether the node x of G in Fig. 5 was to be deleted as being an image of the rewritten node root, or was to be kept as being the target of a redirection, namely a; DACTL just keeps everything. These two clues form the basis of the universal semantics for graph rewriting given in the following sections.

4. Universal properties of DACTL rewriting

In this section we will show how the operational semantics of the previous section can be recast into a form that has universal properties reminiscent of (but not equivalent to) pushouts.

4.1. Introduction

To turn the DACTL construction into a universal construction turns out to be a fairly big job. Before plunging into the details we give some indication of what is to come, informally.

The essence of the problem is redirection. If we are to make a universal construction, we need the effects of redirection to be built into the structure of rules in a "specificational" rather than "operational" manner (we use these words loosely). On the understanding that the LHS of a rule is a template for possible redexes, in a universal reformulation, the RHS should naturally be a template for the possible results of rewrites by the rule; thus, it should incorporate the redirections in some "after the event" manner. Maybe if we applied the redirections to the full pattern of a rule, we would get something that could act as a template for the results of rewrites, given that the full pattern itself acts universally as a template for the intermediate graph in the operational formulation, via the characterisation of contractum building as pushout.

This is a sound piece of intuition, but there are a number of obstacles to be overcome before it can really be made to work properly. Firstly there is the fact that just redirecting arcs in the full pattern may not be enough. Indeterminate numbers of arcs whose heads are in the redex, but which are otherwise outside the redex, also get redirected in the operational definition, so we need to ensure that any universal reformulation captures this. Secondly, there is a certain amount of ambiguity in the way that the DACTL operational model specifies redirections, which is resolved only by reference to individual redexes. In particular, suppose two nodes of the left subpattern a and b map to the same redex node x. So $m(a)=x=m(b)$. Suppose that a is redirected by the rule but b is not. Then Definition 3.8(3) states that x should obey the redirection pertaining to a rather than the unstated identity redirection pertaining to b, i.e. there is a priority mechanism in force which states that nonidentity redirections take precedence over identity redirections. (That two nonidentity redirections never disagree is assured by (RED-3) of Definition 3.4.)

The first problem is not too hard to solve. We specify rules as a pair of functions (i, r) between the left and right patterns. The i function is a symbol/arity-preserving injection which identifies how the nodes of the redex (and the tails of their sequences of out-arcs) are to be found in the result graph. On the other hand, the r function is a redirection function which specifies how all the in-arcs to a node in the redex are to be redirected to form the result graph. Suitably handled, this idea can be exploited to deal with both contractum building and redirection in a unified "specificational" style.

The second problem is trickier, and there are two basic approaches to solving it. The first is to impose sufficient syntactic restrictions so that all the troublesome cases simply do not occur. We do this in Section 6, and the outcome is a construction with strongly universal properties describable using an opfibration. But one of our goals is to characterise universally the original model of rewriting, including its darker corners, and this turns out to be possible using a little technical ingenuity.

The basic idea is to identify those nodes whose redirections are potentially ambiguous, and to fix up the rules so that their redirections are sufficiently flexible to be able to accommodate all the possibilities. The nodes in question are nodes such as b discussed above, which ordinarily have an identity redirection, but for which there are redexes $m: L \to G$ such that $m(b) = m(a)$ and a has a nonidentity redirection. Also we must include all implicit nodes of L, since these can always map to the same redex node as such an a, if the redex is sufficiently cyclic. How do we then accommodate the ambiguous redirections?

For every node of L prone to such ambiguity, say b, we introduce a new implicit node b' on the RHS of the rule (called the mate of b) and set the redirection function to send b to b'. Thus, the LHS and RHS versions of b are related by $i(b) = b$ (using the same names for the injective images of nodes), while $r(b) = b'$. All the in-arcs of b on the LHS get redirected to b' on the RHS, while any redirection whose intended target was the original node b gets turned into a redirection to $i(b)$ on the RHS. The fact that the mate $r(b)$ is implicit, and can therefore match anything, gives us enough flexibility in designing the construction of the universal square, to enable it to deal with both the identity and nonidentity redirection cases.

Of course, it is possible to construct mechanically the universal form of a DACTL rule from the conventional operational form. Basically, one takes P, finds all the nodes which need mates and redirects all in-arcs of these nodes to the mates. One then applies the DACTL redirections to the result, taking care always to redirect to the nonmate node. The end product is the RHS of the D rule, and the (i, r) functions can be found easily. Obviously, one needs to show that the two constructions agree. They do, but it turns out that the universal form is marginally more expressive than the operational form. This is because of the latter's sequential nature; some aspects of contractum building can be undone by redirection, while the "specificational" flavour of the universal construction does not suffer from such a shortcoming. All this is covered below, and illustrated in the examples which follow.

To get some more feeling for how this works, readers could skip ahead to Example 4.8, which shows a rule exhibiting the mentioned ambiguities. The RHS of the rule in universal formulation including the mate nodes is presented in the pattern Q, and a rewrite of a redex using the rule is shown. However, to appreciate the example fully, one has to absorb the details of the explicit constructions to which we now turn.

4.2. The D rewriting construction

The following definition sets out the structure of our DACTL rules in universal form, called D rules. This boils down to a relation given by pairs of functions (i, r) on pairs of patterns $(i, r): L \to Q$, and subject to a long list of restrictions. It is given in this form in order to facilitate comparison with later definitions, but its main points may be stated in English as follows.

(INJ) just states that $i: L \to Q$ is a symbol/arity-preserving injection that preserves the implicit or explicit nature of nodes. It thus provides a component that mimics the inclusion of the left subpattern into the full pattern of a DACTL rule of the previous section. (IMP-D) states that all implicit nodes of Q are i or r images of L nodes, as one would expect (i.e. "no new variables on the RHS"). The redirection function $r: L \to Q$ mimics the redirection pairs of the DACTL formulation, and the nature of the condition (RED-D) (which sets out the rules of engagement regarding mate nodes) is clarified in Lemma 4.2. Finally, (HOM) is perhaps the most crucial condition, generalising the conventional criterion for a homomorphism to that of a redirection couple, the more flexible notion at the heart of all our constructions in this paper.

Definition 4.1 (*D rule*). Let $\mathscr{P}_D$ be the class of patterns (as in Definition 3.2), together with a binary relation $\to$ given by a pair of functions (i, r), such that if $(i, r): L \to Q$ then the following invariants hold:

(INJ) (a) $x \neq y \Rightarrow i(x) \neq i(y)$,
(b) x implicit $\Leftrightarrow$ $i(x)$ implicit,
(c) $\sigma(i(x)) = \sigma(x)$,
(d) $A(i(x)) = A(x)$.

(IMP-D) y implicit $\Rightarrow \exists\, x$ such that $y = i(x)$ or $y = r(x)$.

(RED-D) (a) $r(x)$ explicit $\Rightarrow$ x explicit,
(b) x implicit $\Rightarrow$ $r(x)$ implicit and $r(x) \neq i(x)$,
(c) x, y implicit and $x \neq y \Rightarrow r(x) \neq r(y)$,
(d) $r(x)$ implicit $\Rightarrow r(x) \in \mathrm{rng}(i)$ or $r^{-1}(r(x)) = \{x\}$,
(e) 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 $r(y) = i(y)$, or at least one of $r(x), r(y)$ is implicit and not in $\mathrm{rng}(i)$].

(HOM) (i, r) is a redirection couple, i.e. [(p_k, c) an arc of $L \Leftrightarrow (i(p)_k, r(c))$ an arc of Q].

According to (RED-D) (d) in particular, the nodes y of Q split into two classes: those which are explicit or in $\mathrm{rng}(i)$, called nonmates, and those implicit nodes not in $\mathrm{rng}(i)$ (for which $\mathrm{card}(r^{-1}(y)) = 1$), called mates (of the unique corresponding node $i(r^{-1}(y))$).

We call a $\to$-related pair $(i, r): L \to Q$ a D rule, and for every $x \in L$, $r(x)$ is the redirection of x. L and Q are called the left and right patterns of the D rule.

Note the distinction in nomenclature; while it is possible to identify the left subpattern of a DACTL rule with the left pattern of a D rule, the same cannot be done

with the full pattern of a DACTL rule and the corresponding right pattern of a D rule. There is, however, a mapping from DACTL rules into D rules which we will give in Procedure 4.8.

We note that we may replace (RED-D) (e) by the slightly simpler

(RED-D) (e′) x, y explicit and $\sigma(x)=\sigma(y) \Rightarrow r(x)=r(y)$, or $r(x)=i(x)$ and $r(y)=i(y)$,

by analogy with the corresponding construction in the previous section.

Lemma 4.2. *Let $(i, r): L \to Q$ be a D rule. Then*

(1) *the explicit nodes x of L partition into equivalence classes $E(x)$ such that $y \in E(x)$ $\Leftrightarrow \exists$ a homomorphism $h: L \to Z$ for some Z such that $h(x)=h(y)$.*

For each equivalence class E, exactly one of (2.1) *or* (2.2) *holds:*

(2.1) *There is a nonmate $y \in Q$ such that for at least one $x \in E$, $r(x)=y$, $r(x) \neq i(x)$, and for any $z \in E$ such that $r(z) \neq y$, there is an implicit $z' \in Q$ (the mate of $i(z)$) such that $r(z)=z'$.*

(2.2) *For each $x \in E$, either $r(x)=i(x)$, or $r(x)=x'$, where x' is the mate of $i(x)$.*

For each implicit $y \in Q$, exactly one of (3.1) *or* (3.2) *holds:*

(3.1) *$y=i(x)$ for some unique implicit x (and there may be some $z \in L$ such that $y=r(z)$).*

(3.2) *$y \notin \mathrm{rng}(i)$ but $y=r(x)$ for some unique $x \in L$. (These are the mate nodes.)*

Proof. We can establish (1) on the basis of the existence of pushouts of strict pairs of arrows in the category of patterns and pattern homomorphisms. The rest is an easy exercise in naive set theory. □

Informally, Q contains all the nodes of L, plus perhaps some explicit new nodes (which we can call contractum nodes by analogy with the DACTL situation), plus perhaps some new implicit mate nodes for some of the nodes of Q.

We can see the role of the mate nodes much more clearly when we consider the D rewriting construction itself, which is given next. As in DACTL rewriting, the graph G below could just as easily be taken to be a pattern.

Definition 4.3 (*D rewrite*). Let $(i, r): L \to Q$ be a D rule. Let $g: L \to G$ be a matching to a graph G. Let N_Q^{nm} be the set of nonmate nodes of Q. Let the graph H be given by

(1) $N_H=(N_G \uplus N_Q^{\mathrm{nm}})/\approx$, 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 $g(p)=x$ and $i(p)=n$.

(2) $\sigma_H(\{\langle 1, x\rangle\})=\sigma_G(x)$,

$\sigma_H(\{\langle 2, n\rangle\})=\sigma_Q(n)$,

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

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

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

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

$$h(n) = \begin{cases} \{\langle 2, n\rangle \ldots\} & \text{if } n \in N_Q^{\mathrm{nm}}, \\ s(g(p)) & \text{if } n \text{ is the mate of } i(p). \end{cases}$$

(3) $\alpha_H(\{\langle 1, x\rangle\})[l] = s(\alpha_G(x)[l])$,

$\alpha_H(\{\langle 2, n\rangle\})[l] = h(\alpha_Q(n)[l])$,

$\alpha_H(\{\langle 1, x\rangle, \langle 2, n_1\rangle \ldots \langle 2, n_q\rangle\})[l] = s(\alpha_G(x)[l])$.

Theorem 4.4. *Definition* 4.3 *is consistent. Furthermore,*

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

(b) *(j, s) is a redirection couple, i.e. $[(x_l, y)$ an arc of $G \Leftrightarrow (j(x)_l, s(y))$ an arc of $H]$,*

(c) *h is a homomorphism.*

Proof. Clearly N_H exists since Definition 4.3(1) is a pushout in $\mathscr{S}et$ and $\mathscr{S}et$ has all pushouts. Similarly, σ_H is well defined; in particular, no nonsingleton node of H can contain more than one $\langle 1, x\rangle$ element from G.

Thus, j is a symbol-preserving injection. We also see that s is well defined, and coincides with j, except when the first case in its definition applies (and sometimes also even then). Suppose then x is such that there is a $p \in L$ such that $g(p) = x$ and $r(p)$ is not a mate. There may be several such p, but Lemma 4.2 tells us that they all belong to an equivalence class satisfying Lemma 4.2(2.1) or 4.2(2.2). If Lemma 4.2(2.1) is satisfied then all the $r(p)$ are equal, while if Lemma 4.2(2.2) is satisfied then all the $g(p)$ are the same, whence all possible $\{\langle 2, r(p)\rangle \ldots\} = \{\langle 2, i(p)\rangle \ldots\} = \{\langle 1, g(p)\rangle \ldots\}$ are the same and the first clause of s is unambiguous. Thus, s and, clearly, h are well defined, and so is α_H, whence j also preserves arities. That (j, s) is a redirection couple is immediate from the definition of α_H.

We have to check that h is a homomorphism. Suppose n is explicit in Q. Then $h(n) = \{\langle 2, n\rangle \ldots\}$ and $\sigma(h(n)) = \sigma(\{\langle 2, n\rangle \ldots\})$, which is either $\sigma(n)$ if $\{\langle 2, n\rangle \ldots\}$ is a singleton, or $\sigma(x)$ if not. In the latter case, the definitions of homomorphism and of $\approx$ ensure that $\sigma(x) = \sigma(n)$. Also, $A(h(n)) = A(n)$ by the definition of α_H.

Suppose now that (n_l, m) is an arc of Q. We must show that $\alpha_H(h(n))[l] = h(\alpha_H(n)[l])$. There are a number of cases depending on whether $h(n)$ is a singleton or not, and whether $h(m)$ is a singleton, a nonsingleton or a mate.

If $h(n)$ is a singleton, then the second clause of the definition of α_H is what is required.

Suppose then that $h(n)$ is a nonsingleton $\{\langle 1, x\rangle, \langle 2, n\rangle \ldots\}$, with $p \in L$ such that $g(p) = x$, $i(p) = n$. Let (p_l, q) be the unique arc of L such that $(i(p)_l, r(q)) = (n_l, m)$, which exists by Theorem 4.4(a) and (b). Then $(g(p)_l, g(q))$ is an arc of G, and

$(j \circ g(p)_l, s \circ g(q))$ is an arc of H. Now $j \circ g(p) = \{\langle 1, x\rangle, \langle 2, n\rangle \ldots\}$. We must show that $s \circ g(q) = h(m)$.

Suppose $h(m)$ is a singleton $\{\langle 2, m^n\rangle\}$. Then $m \notin \mathrm{rng}(i)$. So m is either a mate or not. If m is not a mate then $h(m) = \{\langle 2, m\rangle \ldots\} = \{\langle 2, m\rangle\}$ by definition. Also, if m is not a mate, then the first clause of the definition of s applies to $g(q)$, and so $s(g(q)) = \{\langle 2, r(q)\rangle \ldots\} = \{\langle 2, m\rangle \ldots\}$. But this is just $h(m)$. If m is a mate, then $h(m) = s(g(q))$ by definition. Either way we have what we need.

Finally, suppose $h(m)$ is a nonsingleton $\{\langle 1, y\rangle, \langle 2, m^{\sim}\rangle \ldots\}$. Since m is certainly not a mate, $h(m) = \{\langle 2, m\rangle \ldots\}$ by definition. So we may take $m^{\sim} = m$ for convenience. As before, since m is not a mate, the first clause of the definition of s applies to $g(q)$, and $h(m) = s(g(q))$. We are done. □

We gave the above proof in full detail to illustrate the techniques typically needed for proving the results of this paper. One should particularly note the interplay that often arises between the conditions arising from redirection couples, where arcs are mapped using two functions, and those arising from homomorphisms, where only one function is needed. Aside from this, the proof amounts to fairly unexciting diagram chasing and case analysis. In future, we will, for the most part, only sketch the main points of a proof, omitting the boring details. For the enthusiastic reader, these can be found in [2].

4.3. Universality of D rewriting

Theorem 4.5. *In the notation of Definition* 4.3 *and Theorem* 4.4, $j \circ g = h \circ i$ *and* $s \circ g = h \circ r$, *i.e. the squares of Fig.* 8 *commute.*

Proof. Straightforward. □

The next result is the main property of D rewriting.

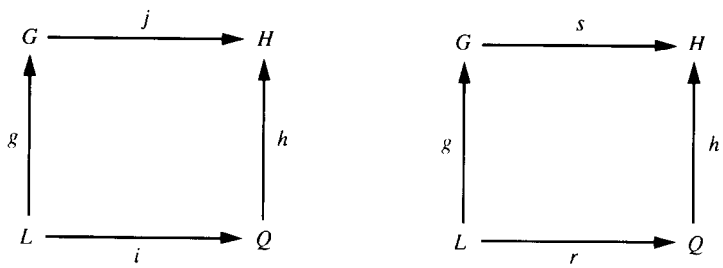


Fig. 8. Commuting squares for the D rewriting construction.

Theorem 4.6 (Universality of D rewriting). *Using the notation of Definition* 4.3 *and Theorem* 4.5, *let* H' *be a graph and suppose* $(j', s'): G \to H'$ *and* $h': Q \to H'$ *are such that*

(1) j' *is a symbol/arity-preserving node map,*

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

(3) h' *is a homomorphism,*

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

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

(6) $r(p) = i(a)$ *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 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* $h(Q)$,

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

i.e. Fig. 9 *commutes.*

Proof. The key thing is to define (θ, ρ) as follows:

$$\theta(j(x)) = j'(x),$$

$$\theta(h(n)) = h'(n) \quad \text{for } n \in N_Q^{\mathrm{nm}},$$

$$\rho(s(x)) = s'(x),$$

$$\rho(h(n)) = h'(n) \quad \text{for } n \in N_Q^{\mathrm{nm}},$$

$$\rho(j(x)) = \theta(j(x)) \quad \text{whenever } j(x) \notin s(G) \cup h(Q).$$

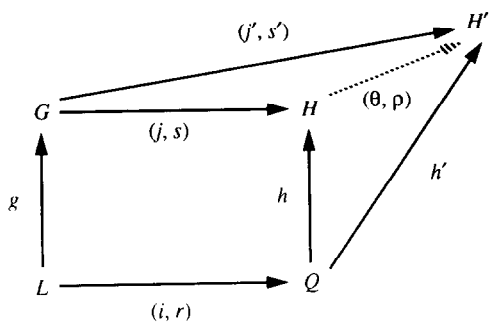


Fig. 9. Universality of D rewriting.

Now most of the properties needed for θ arise because N_H is a pushout in $\mathscr{Gel}$. The harder part is to show that ρ is well defined and behaves appropriately.

To show ρ is well defined on $s(G)$, if $x \neq y$ in G and $s(x)=s(y)$, we need to show that $s'(x)=s'(y)$. It turns out that both x and y must be in $g(L)$, so there are $p, q \in L$ with $x=g(p)$ and $y=g(q)$.

If $r(p)=r(q)$, then $s'(x)=s'(y)$ easily. If not, we must examine the cases for $s(x)$ and $s(y)$. If both fall into the first case for s, Theorem 4.6(5) is needed to show that $s'(x)=s'(y)$. If only one is in the first case, then Theorem 4.6(6) is needed. Thus, ρ comes to be well defined on $s(G)$.

Showing that ρ is well defined on the rest of H is straightforward, and showing that (θ, ρ) have the required properties is now an easy diagram chase. □

The significance of the above result comes with the following key theorem.

Theorem 4.7. *With the notation of Definition* 4.3 *and Theorems* 4.4–4.6, *let* $(j^*, s^*): G \to H^*$ *and* $h^*: Q \to H^*$ *satisfy conditions* (1)–(6) *of Theorem* 4.6. *Suppose further that, for any* $(j', s'): G \to H'$ and $h': Q \to H'$ *that also satisfy those conditions, there is a unique* $(\theta^*, \rho^*): H^* \to H'$ *satisfying Theorem* 4.6(a)–(d). *Then* H *and* H^* *are isomorphic.*

Proof. Routine. □

We thus see that, up to graph isomorphism, the D rewriting construction is universal as a way of completing the squares of Fig. 8 among redirection couples and homomorphisms satisfying the qualifying criteria given by Theorem 4.6(1)–(6).

4.4. Examples

We look at an example where the mate nodes play a role.

Example 4.8. Figure 10 gives an illustration of most of the points discussed above. The notation for graphs and patterns is as previously described, except that we suppress node names for brevity, other than for implicit nodes, whose labels we suppress instead. As usual, the patterns and graphs are named according to their standard roles. So L and P form a DACTL rule as before (the obvious injection is suppressed), while L and Q form the corresponding D rule, manufactured as per Section 4.5.

There is an obvious redex for the rule in the graph G. Following the DACTL construction, contractum building gives G', and applying the redirections yields the graph H.

To see how the D rewriting construction handles this situation, we claim that the pattern Q is the appropriate RHS in the D formulation. First of all, all the implicit nodes of P have acquired primed mates, and these have had all in-arcs from their

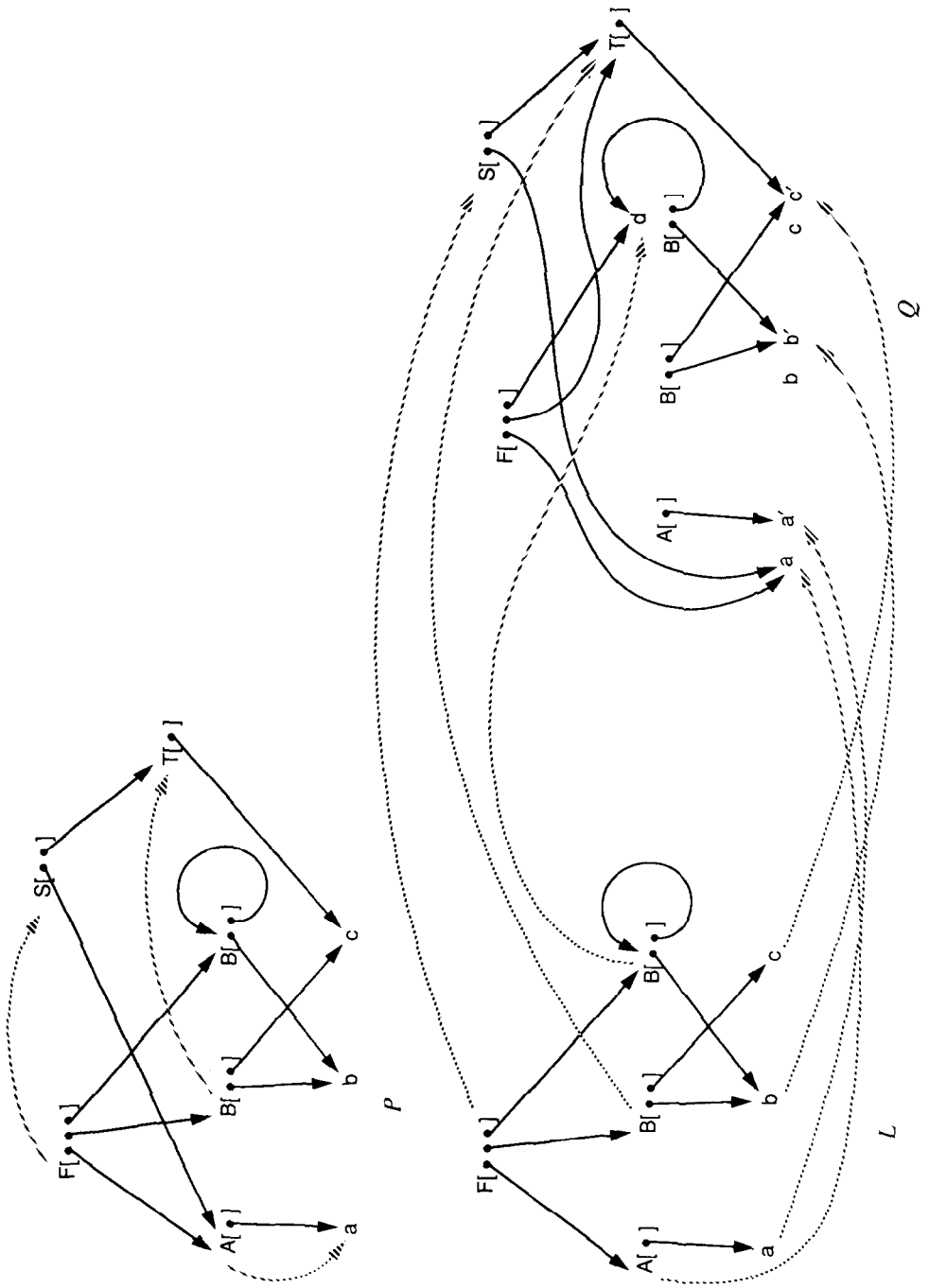


Fig. 10a, b.

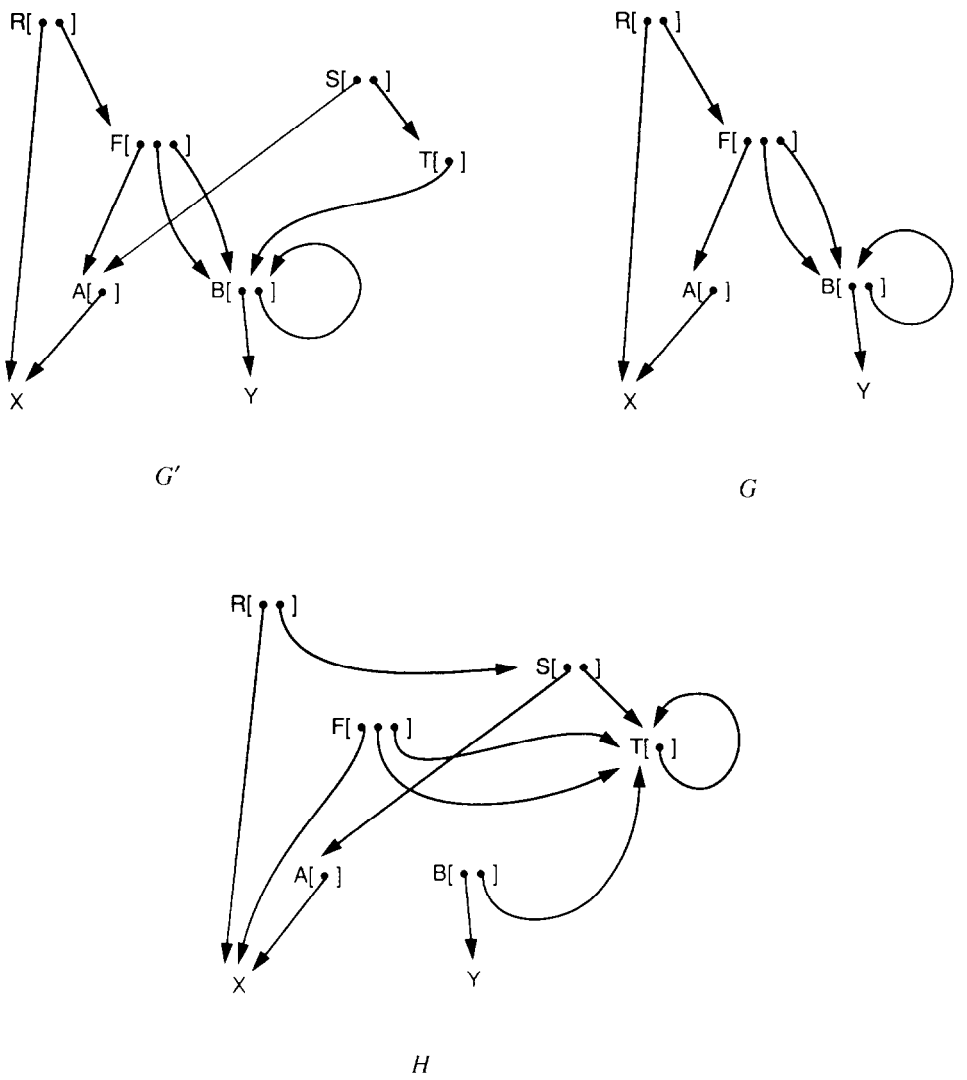


Fig. 10. A DACTL rule and its corresponding D rule, and a rewrite governed by these rules.

unprimed companions redirected to them, ready to model the various possible redirections that such arcs could undergo in a specific redex. The same applies to the right B node, which also has a potentially ambiguous redirection, as it could match the same redex node as the left B. Applying the redirections of the DACTL rule to this intermediate form, taking care always to redirect to the nonmate node, completes the picture. The (i, r) functions are now clear, and only the r function is illustrated.

Considering the redex $g: L \rightarrow G$, the maps $(j, s): G \rightarrow H$ are now obvious, as is the homomorphism $h: Q \rightarrow H$, once we realise that both a and a′ map to X, b and c map to

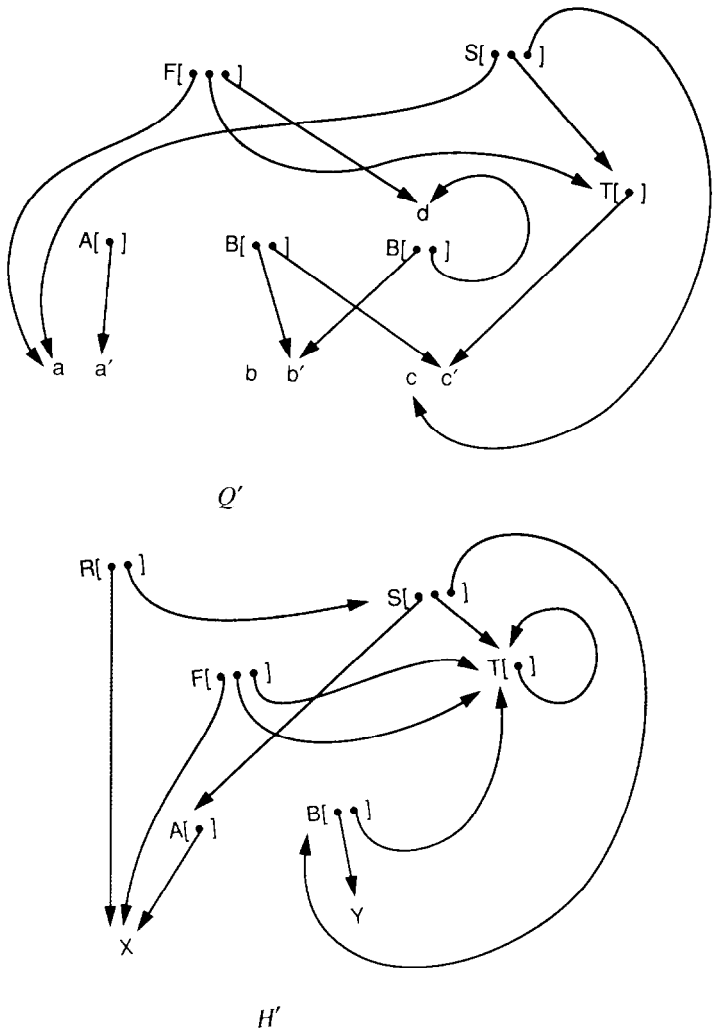


Fig. 11. A modified form of the rule and rewrite of Fig. 10.

B, and b′, c′ and d map to T. The two rewriting constructions agree, as they are supposed to.

Example 4.9. Let us consider a slight modification to the D rule of Example 4.8, namely introducing an extra arc into the pattern Q from S to the nonmate c. Figure 11 illustrates the new pattern Q'. Keeping the (i, r) functions as before gives us a perfectly serviceable D rule $(i, r): L \to Q'$. The graph H' produced by rewriting is also shown in Fig. 11. However, the new D rule is not the counterpart of any DACTL rule, and the rewrite it produces cannot be emulated by any DACTL rewrite. The reason is

that, in DACTL, redirection follows contractum building. The extra in-arc of c from S, if introduced during contractum building in a DACTL rewrite of the given redex, would be redirected to T, because c is matched to B in the redex, and all in-arcs of B get redirected. In DACTL we cannot build a new in-arc to a node of the redex that is subject to redirection, without that arc being itself redirected. The D construction, by simultaneously specifying contractum building and redirection in a universal construction, is able to accomplish this.

Example 4.10. For purposes of illustration, we re-examine the I combinator in the D framework. Figure 12 shows the rule. In fact, it turns out that, when this is applied to the circular instance $G = \mathsf{x} : \mathsf{I}[\mathsf{x}]$, the resulting $H = G$ is universal among all ways of completing the square $G \leftarrow L \rightarrow Q$, which satisfy the universality conditions given by Theorem 4.6(1)–(4). A moment's thought shows that Theorem 4.6(5) and (6) can safely be ignored for this redex since all LHS nodes map to the same image node in G. There will be more on the I combinator in Section 6.

4.5. Mapping DACTL rules to D rules

We now give the formal definition of the mapping from DACTL to D rules advertised earlier.

Procedure 4.11.

Input: A DACTL rule $\langle incl : L \rightarrow P, root, Red \rangle$.

Output: A D rule $(i, r) : L \rightarrow Q$.

(1) Let X be the subset of nodes of L such that $x \in X$ iff

(a) x is implicit, or

(b) there is no $\langle x, y \rangle \in Red$, but there is some homomorphism $g : L \rightarrow G$ such that, for some $u, v \in P$, $g(x) = g(u)$ and $\langle u, v \rangle \in Red$.

X is the set of nodes of L requiring mates. For each $x \in X$ add an isolated implicit mate node x' to P, yielding a fresh pattern P'.

(2) For all $x \in X$, insert $\langle x, x' \rangle$ into Red, yielding a fresh set of redirections Red'.

(3) Apply the redirections in Red' to P', yielding a pattern Q.

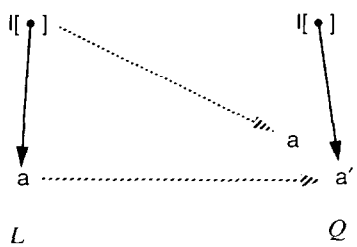


Fig. 12. The D rewriting version of the I combinator rule.

(4) For all $x \in L$, let $(i, r): L \to Q$ be defined as follows:

$$i(x) = x,$$

$$r(x) = \begin{cases} y & \text{whenever there is a pair } \langle x, y \rangle \in Red', \\ x & \text{otherwise.} \end{cases}$$

Lemma 4.12. *The construction of Procedure* 4.11 *is consistent.*

Proof. We have to check that, given a DACTL rule $\langle incl: L \to P, root, Red \rangle$ which satisfies the invariants of Section 3, the construction of Procedure 4.11 is unambiguous and generates a D rule $(i, r): L \to Q$ satisfying the invariants of Definition 4.1. The details of this are too unexciting to include here. We note in passing that the *root* node of the DACTL rule plays no part whatsoever in Procedure 4.11. □

The main fact about the construction of Procedure 4.11 comes next.

Theorem 4.13. *Let $\langle incl: L \to P, root, Red \rangle$ be a DACTL rule, and let $(i, r): L \to Q$ be the corresponding D rule constructed by Procedure* 4.11. *Let $g: L \to G$ be a redex. Let $H1$ be the graph constructed from G by applying the DACTL rule and the rewriting construction, and let $H2$ be the graph constructed by applying the D rule and the D rewriting construction. Then $H1 = H2$. (Note that we mean actual equality of graphs rather than mere isomorphism.)*

Proof. First of all, the two constructions give the same set of nodes for $H1$ and $H2$, as may be verified by consulting Definitions 3.6(1), 3.8(1) and 4.3(1), and noting that $N_Q^{nm} = N_P$. By Definitions 3.6(2), 3.8(2) and 4.3(2), these nodes are similarly labelled. It remains to verify that $\alpha_{H1} = \alpha_{H2}$. To do this we have to track an arbitrary arc of G, or an arbitrary contractum arc, through the two constructions, and check that it ends up in the same place either way. This involves a lot of case analysis, depending on where the head and tail of the arc lie, whether mates are involved or not, etc. All the cases are straightforward and we omit the details. $H1 = H2$ indeed. □

Theorem 4.14. *The function defined by Procedure* 4.11 *which maps any DACTL rule to an equivalent D rule has no inverse.*

Proof. It is sufficient to show that a D rule with the following properties is not the counterpart of any DACTL rule. Let $(i, r): L \to Q$ be the rule and let it satisfy the following:

(a) Neither L nor Q has any implicit nodes.

(b) $x, y \in L$ and $x \neq y \Rightarrow \sigma(x) \neq \sigma(y)$.

(c) There is a node $c \in L$ such that

(c.1) $i(c)$ has an in-arc which is not the (i, r) image of any arc of L,

(c.2) $r(c) \neq i(c)$,

(c.3) for all $c \neq x \in L$, $r(x) \neq i(c)$.

Given such a rule, it is easy enough to make a rigorous (and simpler) version of the argument presented in Example 4.9, whereby the image of c in a redex needs to acquire a new in-arc during the rewrite, but would be prevented from doing so in any putative DACTL counterpart of the rewrite, because the redirection phase would sweep it away. □

The D rewriting construction is thus strictly more general than the DACTL rewriting construction. The author's attitude to this is that the D construction is more natural and comprehensive than its sequentially specified predecessor. The lack of expressivity described is after all just a consequence of the temporal order of the two phases, and a different approach to this can overcome it, as shown in Section 4.6.

4.6. Operational aspects of DACTL and D rewriting

We can better understand the motives behind the introduction of the DACTL rewriting construction and its relationship to the D model by examining how prospective implementations might actually work. The DACTL construction might be implemented by the steps [DACTL-1]–[DACTL-3]:

[DACTL-1] Identify the redex $m: L \to G$ of rule $\langle incl: L \to P, root, Red \rangle$ in graph G.

[DACTL-2] Add copies of nodes and arcs of $P - L$ to G in order to make graph G'.

[DACTL-3] Redirect arcs of G' according to the specifications in *Red* to make graph H.

The D construction might proceed by [D-1]–[D-4]:

[D-1] Identify the redex $g: L \to G$ of rule $(i, r): L \to Q$ in graph G.

[D-2] Add copies of nonmate nodes of $Q - i(L)$ to G to make graph G_1. Determine the maps $(j, s): G \to G_1$.

[D-3] Redirect arcs of G_1 making graph G_2 so that [(p_l, c) an arc of $G_1 \Leftrightarrow (j(p)_l, s(c))$ an arc of G_2].

[D-4] Add copies of arcs of $Q - i(L)$ to G_2 to make H.

The DACTL construction is clearly operationally much simpler, and the intermediate object G' that it creates is a bona fide graph as per Definition 3.2. In the D construction, the intermediate objects G_1 and G_2 are not really graphs since they have "missing arcs"; i.e., in the D model, contractum building must be interleaved with redirection to avoid the pitfalls mentioned in Example 4.9 and Theorem 4.14. In the D model, the RHS of a rule comes with its arcs "already redirected".

One can thus understand why a term graph rewriting model designed with operational issues in mind might turn out looking like the DACTL model rather than the D model. Nevertheless, the full expressiveness of the D model is available operationally at a small price.

5. Intermezzo: local and global universality, and the category $\mathscr{R}^h$

5.1. Local and global universality

We remark how like a pushout Theorem 4.6 seems in some ways, and how unlike one in others. The unique factorisation property of pushouts is characteristically preserved. However, the objects and arrows in Fig. 9 do not come from a category. The objects at the lower level of Fig. 9 are patterns, while the objects at the upper level are graphs. More seriously, the vertical arrows are homomorphisms, while the horizontal arrows are redirection couples obeying a variety of restrictions; the diagonal arrow that factors them is some sort of compromise between the two.

It is thus not obvious how Theorem 4.6 can be placed in a categorical setting. One of the reasons for this is that there is a fundamental asymmetry between the left and right patterns of a D rule. The right pattern almost always has more implicit nodes than the left pattern, and these extra mate nodes make it impossible to define identity arrows in the natural way. For example, Fig. 13 illustrates how one would envisage the identity arrow for the left pattern of the I combinator; it is hardly an identity in any useful sense of the word. This feature prevents the D rules from forming the arrows of a category that arises naturally via Procedure 4.11.

A further consequence of the asymmetry between the left and right patterns of a D rule is that it is not easy to glue the universal commuting squares together horizontally to get commuting rectangles of the same kind. This is mainly due to the way in which mates are used; they mix up properties of the "objects" and "arrows" of the construction in such a way that a properly associative law of composition fails to hold. This is all disregarding the problem of how one avoids the profusion of (useless) mates as one attempts to compose D rules.

However, the ability to compose squares horizontally is a standard property of pushouts (and of the opfibrations used in the next section), so it seems that we are implicitly dealing with more than one notion of universality. To this end, by local universality we will mean the universality possessed by the D rewriting construction, namely the ability to complete a single square in a universal manner. By global

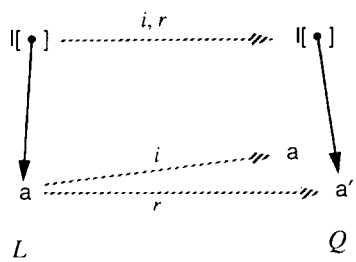


Fig. 13. The identity arrow for the I combinator in a prospective category built out of D rules.

universality we will mean the stronger property of being able to compose such squares horizontally, as for pushouts and opfibrations.

Despite the fact that Theorem 4.6 is not categorical, we should not imagine that its usefulness is diminished thereby. Once we acknowledge the asymmetry between the left and right sides of Fig. 9, we can throw off our inhibitions even further and enhance Theorem 4.6 to accommodate the DACTL markings which are truly asymmetrical in the two sides of a rewrite. It may even be possible to accommodate the pattern calculus of DACTL within this framework, although the author is not clear about this at the time of writing. However, the most useful property of Theorem 4.6 is the economy it brings to reasoning about DACTL systems. Such reasoning usually reduces to induction over the structure of execution sequences. The inductive step in such proofs consists of showing that rewrites arising from some system preserve some property or other. Formerly, it was necessary to trace explicitly this preservation through the various phases of a rewrite, a rather laborious process. The universal characterisation of rewrites available through Theorem 4.6 cuts this detail down very substantially. Thus, Theorem 4.6 raises the level of abstraction at which it is possible to think about DACTL rewrites.

5.2. The category $\mathscr{R}^h$

The thoughts of Section 5.1 suggest an alternative direction to take in looking for a categorical semantics for DACTL-style rewriting, which we explore briefly here. There is nothing to prevent us from associatively composing redirection couples, so let us make the following definition.

Definition 5.1. The category of pure redirections $\mathscr{R}^h$ has as objects all patterns, and as arrows all (h,r) function pairs such that if $(h,r): L \to R$ is an arrow then:

(INJ-$\mathscr{R}^h$) h is a symbol/arity-preserving node function on the explicit nodes of L.

(HOM) (h,r) is a redirection couple, i.e. $[(p_l, c)$ an arc of $L \Rightarrow (h(p)_l, r(c))$ an arc of R, and $(h(p')_l, r(c'))$ an arc of $R \Rightarrow \exists (p_l, c)$ an arc of L such that $(h(p)_l, r(c)) = (h(p')_l, r(c'))]$.

$\mathscr{R}^h$ has a subcategory $\mathscr{R}^i$ wherein (INJ-$\mathscr{R}^h$) is replaced by

(INJ-$\mathscr{R}^i$) h is a symbol/arity-preserving injection on the nodes of L.

Figure 14 shows two isomorphic objects in $\mathscr{R}^h$. Note that these are not isomorphic graphs in the usual sense of the word. In fact, using DACTL redirection notation, the redirections $\{\langle x, y\rangle, \langle y, x\rangle\}$ are a square root of the identity on both graphs. Thus, although the arrows of $\mathscr{R}^h$ potentially give us scope for a pushout-based graph rewriting formalism by reference to Section 4, the notion of isomorphism we end up with is not useful. This is in contrast to the category $\mathscr{H}$ of patterns with normal homomorphisms as arrows, where the notion of isomorphism is the right one, but the notion of pushout is too weak to describe rewriting usefully. Therefore, we can see the

Fig. 14. Two isomorphic graphs in $\mathscr{R}^h$.

problem of looking for an appropriate categorical formulation of DACTL-style graph rewriting as the search for a compromise between these two extremes.

Let us finish this section with a brief sketch of how pushout-based rewriting looks in $\mathscr{R}^h$. Both normal graph homomorphisms and the arrows of $\mathscr{R}^i$ are special cases of arrows of $\mathscr{R}^h$. So we would envisage a rewrite in $\mathscr{R}^h$ as being the pushout of a homomorphism (the redex) and an arrow of $\mathscr{R}^i$ (the rule).

To understand these pushouts most simply, it is best to look at $\mathscr{R}^h$ from a different angle. $\mathscr{R}^e$ is a retract of $\mathscr{R}^h$. Here $\mathscr{R}^e$ is a category defined as follows. The objects of $\mathscr{R}^e$ are pairs $\langle T, H\rangle$. Here T is a set of $(p_l, \sigma(p))$ items corresponding to the tails of arcs of a pattern, i.e. p is a node of the pattern, $l \in A(p)$ is an index in p's arity and $\sigma(p)$ is p's label. H is just a partition of T, and a set in H represents the set of heads of arcs of the pattern that are incident on some particular node. Because of the nature of isomorphisms in $\mathscr{R}^h$, it does not much matter which node any particular set of arcs is incident on; in fact, the retraction functor is just the functor that forgets this information.

If $A=\langle T_A, H_A\rangle$ and $B=\langle T_B, H_B\rangle$ are objects of $\mathscr{R}^e$, an arrow $(h,r): A \to B$ is a symbol/arity-preserving map $h: T_A \to T_B$, together with a redirection function $r: H_A \to H_B$ such that

$$\{h((p_l, \sigma(p))) \mid (p_l, \sigma(p)) \in X \in H_A\} \subseteq r(X) \in H_B,$$

i.e. if a set of arcs were incident on the same node to start with, then they would remain incident on some other same node after redirection.

We can argue that pushouts exist in $\mathscr{R}^e$ as follows. Let $(h_A, r_A): X \to A$ and $(h_B, r_B): X \to B$ be two arrows. Their pushout is then $(j_A, s_A): A \to Y$, $(j_B, s_B): B \to Y$, with $Y=\langle T_Y, H_Y\rangle$.

Here T_Y is the obvious pushout in (suitably labelled) $\mathscr{S}et$ as per our previous rewriting constructions and j_A and j_B are obvious. The partitions H_A and H_B induce partitions H_A^Y and H_B^Y of T_Y, and H_Y is defined as their join in the lattice of partitions of T_Y, i.e. the finest partition at least as coarse as H_A^Y and H_B^Y. The maps s_A and s_B are now also obvious.

From the existence of pushouts in $\mathscr{R}^e$ and the nature of the retraction functor, we deduce their existence in $\mathscr{R}^h$, and, since pushouts are only fixed up to isomorphism, we deduce the unsuitability of $\mathscr{R}^h$ as a vehicle for graph rewriting from our previous remarks. Note how the local universality of Section 4, unconstrained by the need to fit any particular categorical straightjacket, can be engineered to provide just the right

notion of isomorphism, and just the right notion of rewrite too. Readers may care to examine the universality theorem (Theorem 4.6) to see which parts of the hypotheses are responsible for each of these aspects of the conclusions. In the next section, we will see how a crafty compromise can be struck between $\mathscr{H}$-type behaviour and $\mathscr{R}^h$-type behaviour in a well-chosen opfibration.

6. Graph rewriting as opfibration

This section starts off much like Section 4. We define a slightly different rewriting model, the *P* rewriting model, and show that it has properties of the same sort as the *D* rewriting model. However, the minor differences in definition make a big difference in terms of universality properties, and the categorical version of the model, the $\mathscr{P}$ model, fits within an opfibration which we explore in the later parts of this section.

6.1. The P rewriting construction

The next few definitions and theorems parallel those of Section 4 quite closely. On the assumption that the reader has survived Section 4 unscathed, we mostly state them without comment.

Definition 6.1 (*P rule*). Let $\mathscr{P}_C$ be the category whose objects are patterns and whose arrows are rules (called P rules) depicted by pairs of functions $(i,r): L \to R$ satisfying the invariants (INJ), (IMP), (RED) and (HOM) below. (Note the slightly simpler form of (RED) due to the absence of mates. In fact, (RED)(b) is redundant because of (RED)(a) and the modified form of (IMP), but it is left in to aid comparison with the conditions of Definition 4.1.)

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

(IMP) y implicit $\Rightarrow \exists$ implicit x such that $y = i(x)$.

(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) (i,r) is a redirection couple, i.e. $[(p_k, c)$ an arc of $L \Leftrightarrow (i(p)_k, r(c))$ an arc of $R]$.

As previously, we can replace (RED)(c) with the stronger but more transparent

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

It is easy to check that componentwise composition of arrows makes $\mathscr{P}_C$ into a category, actually a subcategory of $\mathscr{R}^i$.

Definition 6.2 (*Rigid homomorphism*). Let L be a pattern and $g: L \to G$ a homomorphism. Then g is called rigid, or a rigid matching, iff [x explicit and y implicit $\Rightarrow$ $g(x) \neq g(y)$]. If $g: L \to G$ is a redex, it is called a rigid redex.

Definition 6.3 (*P rewriting construction*). Let $\delta \equiv (i, r): L \to R$ be an arrow of $\mathscr{P}_C$, and let $g: L \to G$ be a rigid homomorphism of L into a graph G. 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 $g(p) = x$ and $i(p) = n$.

(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 $h: R \to H$:

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

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

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

(3) $\alpha_H(\{\langle 1, x\rangle\})[l] = s(\alpha_G(x)[l])$,

$\alpha_H(\{\langle 2, n\rangle\})[l] = h(\alpha_R(n)[l])$,

$\alpha_H(\{\langle 1, x\rangle, \langle 2, n_1\rangle \ldots \langle 2, n_q\rangle\})[l] = s(\alpha_G(x)[l])$.

The reader will observe that this is a simplified form of Definition 4.3.

Lemma 6.4. *Definition* 6.3 *is consistent. Furthermore,*

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

(b) *(j, s) is a redirection couple, i.e.* [(x_l, y) *an arc of* $G \Leftrightarrow (j(x)_l, s(y))$ *an arc of* H],

(c) *h is a rigid homomorphism.*

6.2. Local universality of the P rewriting model

Theorem 6.5. *In the notation of Definition* 6.3 *and Lemma* 6.4, *$j \circ g = h \circ i$ and $s \circ g = h \circ r$, i.e. the squares of Fig.* 8 *(with Q replaced by R) commute.*

Since the P rewriting construction is a simplified form of D rewriting, it comes as no surprise to discover that it has universal properties by analogy with Theorem 4.6.

Theorem 6.6 (Local universality of P rewriting). *Using the notation of Definition* 6.3, *Lemma* 6.4 *and Theorem* 6.5, *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 node map,*
(2) *(j', s') is a redirection couple, i.e. $[(x_l, y)$ an arc of $G \Leftrightarrow (j'(x)_l, s'(y))$ an arc of $H']$.*
(3) *h' is a homomorphism,*
(4) *$j' \circ g = h' \circ i$ and $s' \circ g = h' \circ r$,*
(5) *$r(p) = i(a)$ and $r(q) = i(b)$ 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. 15 commutes.

Proof. We confirm that the factorising redirection couple (θ, ρ) is given by

$$\theta(j(x)) = j'(x),$$

$$\theta(h(n)) = h'(n),$$

$$\rho(s(x)) = s'(x),$$

$$\rho(h(n)) = h'(n),$$

$$\rho(j(x)) = \theta(j(x)) \quad \text{whenever } j(x) \notin s(G) \cup h(R).$$

Aside from this we have a slightly simpler version of the D construction's Theorem 4.6. □

Theorem 6.7. *Using the notation of Definition* 6.3, *Lemma* 6.4 *and Theorem* 6.5 *and* 6.6, *let $(j^*, s^*): G \to H^*$ and $h^*: R \to H^*$ satisfy conditions* (1)–(5) *of Theorem* 6.6. *Suppose, further, that for any $(j', s'): G \to H'$ and $h': R \to H'$ that also satisfy those conditions, there*

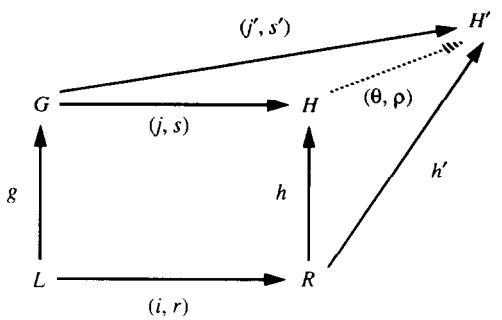


Fig. 15. Local universality of P rewriting.

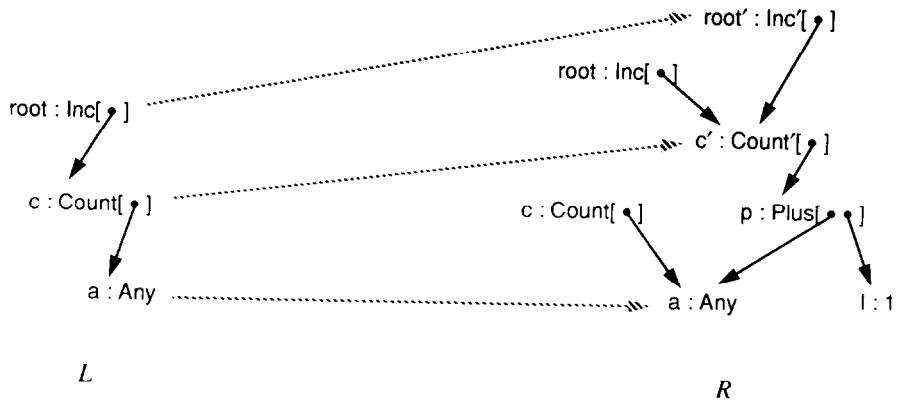


Fig. 16. The rule of Fig. 3 in P rule form.

is a unique $(\theta^*, \rho^*): H^* \to H'$ *satisfying Theorem* 6.6(a)–(d). *Then* H *and* H^* *are isomorphic.*

6.3. Examples

We do not need to look far to see the P rewriting construction in action. Example 3.10 furnishes most of what we need. We have a P rule for this rewrite in Fig. 16. The map i is the obvious injection and the map r is illustrated in Fig. 16. Figure 4 provides the graphs G and H, and the way in which they tie in to the P rewriting construction is clear.

A further example of a P rule is provided by the I combinator. Here the patterns L and R are both equal to the pattern P of Fig. 5. The injection i is obvious and the map r is given by $\{\mathsf{root} \to \mathsf{a}, \mathsf{a} \to \mathsf{a}\}$. The circular instance of this rule in the graph G of Fig. 5 is instructive, since the matching that defines this instance is clearly nonrigid. Nevertheless, the construction of Definition 6.3 goes through without difficulty, and the graph H of Fig. 5 is universal among ways of completing the square $G \leftarrow L \to R$ that satisfy Theorem 6.6(1)–(4). (As in Section 4, Theorem 6.6(5) is irrelevant.)

This state of affairs arises because the condition of rigidity is stronger than what is strictly necessary to assure the consistency of the Definition 6.3 construction, and we could have succeeded with the weaker invariant

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

Theorem 6.4 holds under these broader conditions, the only alteration to the proof being the addition of a third case to check in the well-definedness of s (in reality, two subcases corresponding to the disjunction in (W-RIG)). It is immediate that the circular I redex satisfies (W-RIG). The reason we did not use (W-RIG) before was that it only leads to a form of local universality, and we want global universality in this section.

6.4. Opfibrations and the Grothendieck construction

Suppose we have a (small) functor $W: \mathscr{B} \to \mathscr{C}at$. Then we can construct the category $\mathscr{G}(\mathscr{B}, W)$ as follows. The objects are pairs $\langle x, b\rangle$, where b is an object of $\mathscr{B}$ and x is an object of $W(b)$. The arrows are also pairs $\langle \phi, \beta\rangle : \langle x, b\rangle \to \langle x', b'\rangle$, with $\beta : b \to b'$ an arrow of $\mathscr{B}$ and $\phi : W(b)(x) \to x'$ an arrow of $W(b')$. The composition of $\langle \phi, \beta\rangle : \langle x, b\rangle \to \langle x', b'\rangle$ and $\langle \psi, \gamma\rangle : \langle x', b'\rangle \to \langle x'', b''\rangle$ is given by

$$\langle \psi, \gamma\rangle \circ \langle \phi, \beta\rangle = \langle \psi \circ W(\gamma)(\phi), \gamma \circ \beta\rangle.$$

This is the famous Grothendieck construction. There is an obvious projection functor $F: \mathscr{G}(\mathscr{B}, W) \to \mathscr{B}$ given by $F(\langle x, b\rangle) = b$ and $F(\langle \phi, \beta\rangle) = \beta$, and Grothendieck recognised that projection functors $F: \mathscr{X} \to \mathscr{B}$ correspond to functors $W: \mathscr{B} \to \mathscr{C}at$ with $\mathscr{X} \approx \mathscr{G}(\mathscr{B}, W)$, precisely when F is a split opfibration.

A split opfibration $F: \mathscr{X} \to \mathscr{B}$ is a functor with the following universal property. For every arrow $\beta : b \to b'$ of $\mathscr{B}$ and every object x in $\mathscr{X}$ with $F(x) = b$, there is an arrow $\kappa(\beta, x)$ of $\mathscr{X}$ such that if $F(\alpha : x \to z) = \delta \circ \beta$ then α factors as $\theta \circ \kappa(\beta, x)$ for some unique $\theta : \mathrm{cod}(\kappa(\beta, x)) \to z$; see Fig. 17. Furthermore,

$$\kappa(\mathrm{id}_{F(x)}, x) = \mathrm{id}_x$$

and

$$\kappa(\gamma, \mathrm{cod}(\kappa(\beta, x))) \circ \kappa(\beta, x) = \kappa(\gamma \circ \beta, x).$$

Arrows such as $\kappa(\beta, x)$ which satisfy the factorisation property are called opcartesian, and if they satisfy the extra two conditions, they form a split cleavage for F; see [16, 17], or [15] for the most comprehensive treatment in English, or [5] for a very accessible introduction.

Fibrations (one possible dual of opfibrations) find wide usage in category theory and mathematics in general. For example, Bénabou [6] advocates them as capturing

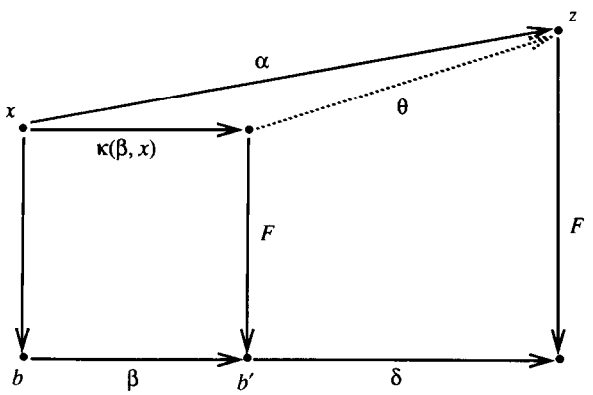


Fig. 17. The universal factorisation property of an opcartesian arrow.

the features needed in a categorical account of foundations. They are also increasingly used in categorical treatments of topics in computing science, e.g. [1, 8, 20, 26].

For us, opfibrations solve the problem of casting a diagram like Fig. 15 into a categorical framework. Note how the vertical and horizontal arrows of Fig. 17 are different kinds of entities as we require, and that if we set δ to an identity, Figs. 15 and 17 look quite alike (no matter that the vertical arrows go in opposite directions in the two figures). The fact that in an opfibration δ is not restricted to identities is a precise statement of global universality.

6.5. *The $\mathscr{P}$ rewriting construction and global universality*

We continue to develop the properties of the P rewriting construction.

Definition 6.8. For each object P of $\mathscr{P}_C$, we construct a category $\mathscr{G}_C^P$. The objects of $\mathscr{G}_C^P$ are pairs $\langle G, g\rangle$. Here G is a graph and $g: P \to G$ is a rigid matching, so an object of $\mathscr{G}_C^P$ is a redex for any rule with LHS P. The arrows $\phi: \langle G, g\rangle \to \langle G', g'\rangle$ of $\mathscr{G}_C^P$ are graph homomorphisms $\phi: G \to G'$ such that

$$g' = \phi \circ g \quad \text{and} \quad \phi^{-1}(g'(P)) = g(P).$$

We call such arrows rigid redex morphisms. In the sequel, the context will always make clear which notion of rigidity is at issue: the former for redexes and the latter for morphisms of redexes.

Lemma 6.9. *Let $\delta \equiv (i, r): L \to R$ be an arrow of $\mathscr{P}_C$ and let $g: L \to G$ be a rigid matching. Let $\phi: G \to G'$ be a rigid redex morphism. Let H and H' be the graphs obtained by rewriting the rigid redexes $g: L \to G$ and $g': L \to G'$ using δ, and let $h: R \to H$ and $h': R \to H'$ be the rigid matchings produced by these rewrites. Then there is a rigid redex morphism $\psi: H \to H'$ in $\mathscr{G}_C^R$ such that Fig. 18 commutes.*

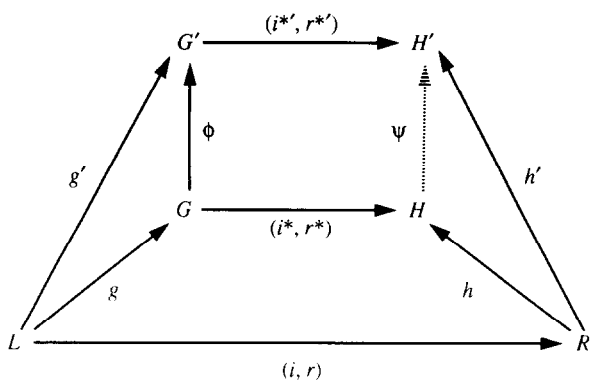


Fig. 18. Preservation of rigid redex morphism property by P rewriting.

Proof. Referring to Fig. 18, set $j' = i^{*\prime} \circ \phi$, $s' = r^{*\prime} \circ \phi$. A short diagram chase establishes that $(j', s') : G \to H'$ and $h' : R \to H'$ satisfy the hypotheses of the local universality theorem (Theorem 6.6). Thence, we get a unique $(\theta, \rho) : H \to H'$ satisfying Theorem 6.6(a)–(d).

For this to be useful, we have to verify that θ and ρ define the same homomorphism on H. This is a short diagram chase. Whereupon, we can set $\psi = \theta = \rho$. Checking that ψ is a rigid redex morphism is now easy. □

The reader will notice how the two vertical sides of Fig. 18 are symmetrical. This allows the extension of the rigid redex morphism preservation property to a functor on $\mathscr{G}_C^L$.

Theorem 6.10. *Let $\delta \equiv (i, r) : L \to R$ be an arrow of $\mathscr{P}_C$. Then there is a functor $Rew_C^\delta : \mathscr{G}_C^L \to \mathscr{G}_C^R$ such that*

$$Rew_C^\delta(\langle G, g \rangle) = \langle H, h \rangle,$$

$$Rew_C^\delta(\phi : \langle G, g \rangle \to \langle G', g' \rangle) = \psi : \langle H, h \rangle \to \langle H', h' \rangle.$$

Proof. Routine. □

Remark. We permit ourselves to write $Rew_C^\delta(G)$ when we are principally interested in the effect of Rew_C^δ on graphs, and the redex preservation is to be understood from the context.

Section 6.4 will have prepared the reader for the next step: the gluing together of the individual functors Rew_C^δ to make a combined functor $Rew_C : \mathscr{P}_C \to \mathscr{C}at$. However, before doing this, we need to address a minor technical detail.

Doing one rewrite and then following it with another is not the same thing as doing a single rewrite using a suitably more elaborate rule; the results are merely isomorphic as graphs. This is because concrete models in set theory of the disjoint union operation are not strictly associative. They are only associative up to isomorphism. This blocks the construction of the functor that we require, since composition fails to work "on the nose". Rather than explore the more convoluted "Australian" version of the resulting theory, we resort to taking representatives of isomorphism classes of graphs and patterns, so-called abstract graphs and patterns, and working only with them. We get categories $\mathscr{P}, \mathscr{G}^L$, etc., skeletons of $\mathscr{P}_C, \mathscr{G}_C^L$, and so on. They will be uniformly denoted by dropping the subscript "C" from our previous categories of concrete graphs and patterns. For precision, we will call the rewriting construction that automatically chooses the right representative of the isomorphism class of the result of the rewrite the $\mathscr{P}$ rewriting construction, distinct from the P rewriting construction. Of course, all the results established previously in this section remain true for $\mathscr{P}$ rewriting, although "up to isomorphism" results like those of Theorem 6.7 become a little vacuous. Now we proceed to the main theorem.

Theorem 6.11. *There is a functor* $Rew: \mathscr{P} \to \mathscr{C}at$ *such that*

$$Rew(P) \to \mathscr{G}^P,$$

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

Proof. See the appendix. □

The existence of $Rew: \mathscr{P} \to \mathscr{C}at$ leads immediately to the construction of the Grothendieck category $\mathscr{G}(\mathscr{P}, Rew)$. The objects of $\mathscr{G}(\mathscr{P}, Rew)$ are pairs $\langle \langle G, g \rangle, L \rangle$, where L is an object of $\mathscr{P}$ 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 $\mathscr{G}(\mathscr{P}, Rew)$ are pairs $\langle \phi, \delta \rangle: \langle g: L \to G \rangle \to \langle h: R \to H \rangle$, where $\delta \equiv (i, r): L \to R$ is an arrow of $\mathscr{P}$ and $\phi: Rew^\delta(G) \to H$ is an arrow of $\mathscr{G}^R$.

In slightly less combinatorial terms, an arrow $\langle \phi, \delta \rangle$ of $\mathscr{G}(\mathscr{P}, Rew)$ can be viewed as a $\mathscr{P}$ rewrite of a redex $\langle g: L \to G \rangle$ by a rule $\delta \equiv (i, r): L \to R$ 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, (j, s) is a redirection couple. Such a pair (j, s) is, strictly speaking, a different thing from $\langle \phi, \delta \rangle$, but we will not be fussy about this below.

Note that, compared to the $\mathscr{P}$ rewriting construction, the arrows of $\mathscr{G}(\mathscr{P}, Rew)$ have an extra homomorphism "tacked onto the end". The significance of these extra homomorphisms will be discussed in Section 7 when we talk about real systems. The arrows of $\mathscr{G}(\mathscr{P}, Rew)$ will be called $\mathscr{G}$ rewrites, and the (Grothendieck) construction that associates one or another arrow $\langle \phi, \delta \rangle$ to each rule δ and redex $\langle g: L \to G \rangle$ will be called the $\mathscr{G}$ rewriting construction. Of course, from a categorical viewpoint, we might justifiably prefer $\mathscr{G}$ rewrites to be called corewrites or oprewrites.

The 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 given by

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

as per the general theory above, and the situation is illustrated in Fig. 19.

One can now read off various properties of the $\mathscr{G}$ rewriting construction from standard results in the theory of opfibrations. For example, the projection functor $F: \mathscr{G}(\mathscr{P}, Rew) \to \mathscr{P}$ is obvious, as are the fibres $F^{-1}(P) = \mathscr{G}^P$ for P in $\mathscr{P}$. Of more interest is the fact that the split opcartesian arrows $\kappa((i, r): L \to R, \langle g: L \to G \rangle)$ can be precisely identified with $\mathscr{P}$ rewrites of $\langle g: L \to G \rangle$ by $(i, r): L \to R$.

Note the pleasing way in which the Grothendieck construction has separated syntax and semantics while nevertheless maintaining the required level of interplay 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 but related categories, rather than one, gives us just the technical elbow room we need. This segregation of syntax and semantics was the main reason why we chose to work with $\mathscr{G}^P$ fibres rather

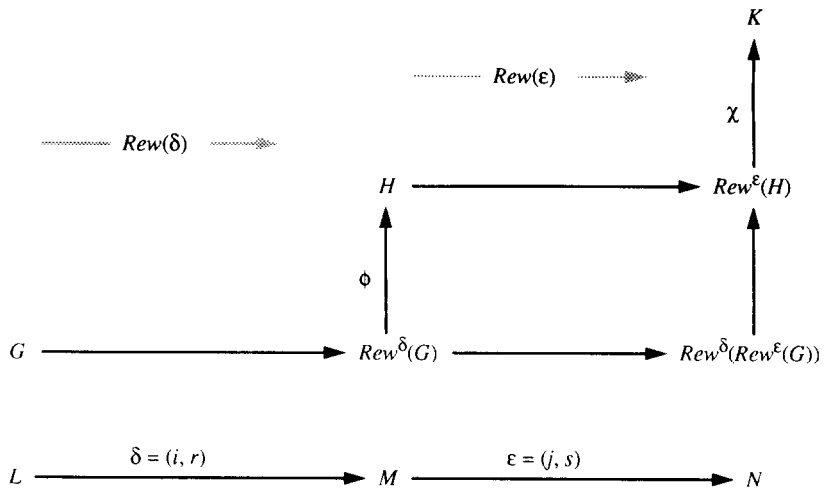


Fig. 19. Composition of arrows $\langle \phi, \delta \rangle : \langle g : L \to G \rangle \to \langle h : M \to H \rangle$ and $\langle \chi, \varepsilon \rangle : \langle h : M \to H \rangle \to \langle k : N \to K \rangle$ in $\mathscr{G}(\mathscr{P}, Rew)$.

than $\mathscr{P}^P$ fibres (where $\mathscr{P}^P$ is the category of images of P within patterns). The latter, although slightly more general in a purely technical sense, perhaps offer the temptation of not keeping the distinction clean enough. This separation also fits in well with the conventional attitude in rewriting theory, where the objects from which rules are built are allowed to make use of the appropriate notion of variable – in our context, implicit nodes – whereas the objects that the rules manipulate generally do not do so – in our context, graphs rather than patterns.

6.6. *Operational aspects of $\mathscr{P}$ rewriting*

The results of the later part of Section 4, relating DACTL and D, have analogues in the framework of $\mathscr{P}$ rewriting for a suitable subset of DACTL rules.

Definition 6.12. We define DACTL$^{\mathscr{P}}$ as the subset of DACTL rules $\langle incl : L \to P, root, Red \rangle$ such that

(RED-3$^{\mathscr{P}}$) x, y explicit and $\exists$ a homomorphism $h : L \to Z$ such that $h(x) = h(y) \Rightarrow [\langle x, t \rangle \in Red \Leftrightarrow \langle y, t \rangle \in Red]$.

The condition (RED-3$^{\mathscr{P}}$) precludes the possibility that $g(x) = g(y)$ in some redex, but a nontrivial redirection of x overrides the unstated identity redirection of y. Assured of this, no explicit nodes of a full pattern need acquire mates in the translation to $\mathscr{P}$ rewriting. Rigidity of redexes ((W-RIG)-idity is enough) likewise makes mates for implicit nodes unnecessary. We give the following results on translation without proofs.

Procedure 6.13.
Input: A DACTL$^{\mathscr{P}}$ rule $\langle incl: L \to P, root, Red \rangle$.
Output: A $\mathscr{P}$ rule $(i,r): L \to R$ (up to isomorphism).
(1) Apply the DACTL$^{\mathscr{P}}$ rule to the redex $incl: L \to L$ to obtain a pattern R.
(2) Define $(i,r): L \to R$ in the obvious way.

Lemma 6.14. *The construction of Procedure* 6.13 *is consistent.*

Theorem 6.15. *On rigid redexes, the* DACTL$^{\mathscr{P}}$ *and* $\mathscr{P}$ *rewriting constructions give isomorphic results.*

The fact that $\mathscr{P}$ rewrites can be composed along arrows of $\mathscr{P}$ gives us an alternative way of getting Theorem 6.15. A DACTL$^{\mathscr{P}}$ rewrite may be decomposed into two DACTL$^{\mathscr{P}}$ rewrites, one having nontrivial contractum building but trivial redirections, and the other having trivial contractum building but nontrivial redirections. Mapping each of these to a corresponding $\mathscr{P}$ rewrite is easy. Using the functoriality of *Rew* then gives Theorem 6.15.

As previously, the function defined by Procedure 6.13, which maps any DACTL$^{\mathscr{P}}$ rule to an equivalent $\mathscr{P}$ rule, has no inverse.

Also from an implementation point of view, we could follow [D-1]–[D-4] of Section 4 exactly, except that there are no mate nodes to worry about.

7. Describing real systems

The deliberations of the previous section belong to the world of "all conceivable $\mathscr{G}$ rewrites". Real systems typically make use of a small selection from the class of all possibilities. In this section we show how to construct categories to describe such situations. This will demonstrate the conceptual usefulness of the global universality of the Grothendieck construction.

7.1. On the role of ϕ in a $\mathscr{G}$ rewrite $\langle \phi, \delta \rangle$

Arrows in $\mathscr{G}(\mathscr{P}, Rew)$ consist of pairs $\langle \phi, \delta \rangle$. Here δ is an arrow of $\mathscr{P}$ and ϕ is a rigid redex morphism. The role of such ϕ's may appear controversial. The first thing to note is that the ϕ's may be dispensed with altogether. All we need to do is use the discrete subcategory $\mathscr{G}^P_{\mathrm{id}}$ of $\mathscr{G}^P$, as fibre over P, for all P in $\mathscr{P}$. In $\mathscr{G}^P_{\mathrm{id}}$, which is just a set of objects, all arrows are identity homomorphisms, obviously rigid. The Grothendieck construction then restricts itself to its simpler variant given by a functor $Rew_{\mathrm{id}}: \mathscr{P} \to \mathscr{S}et$. The arrows of the associated Grothendieck category $\mathscr{G}(\mathscr{P}, Rew_{\mathrm{id}})$ are now just $\langle \mathrm{id}, \delta \rangle$ or, in the alternative notation, $(i^*, r^*): G \to H$, where (i^*, r^*) represents the effect of a $\mathscr{P}$ rewrite of the appropriate redex of G according to δ.

Nevertheless, the ϕ's are useful. In [18, 19], and also in [21], the desirability of identifying isomorphic subgraphs of a graph is explained. The motives include the applicability of nonlinear rewrite rules and the desire to avoid duplicating work wherever possible by sharing identical subgraphs.

Some of this can easily be accommodated within $\mathscr{G}$ rewriting. We merely need to follow each $\mathscr{P}$ rewrite according to a rule δ say, with a suitable homomorphism ϕ, and the $\mathscr{G}$ arrow $\langle \phi, \delta \rangle$ will do the job we need. Of course, the fibres $\mathscr{G}^P$ are too general to do just this since their arrows are arbitrary rigid redex morphisms, but if we use instead the subcategories of $\mathscr{G}^P$, $\mathscr{G}^P_{\text{onto}}$ or $\mathscr{G}^P_{\text{maxonto}}$ as fibres, we get what we require. Here $\mathscr{G}^P_{\text{onto}}, \mathscr{G}^P_{\text{maxonto}}$ have the same objects as $\mathscr{G}^P$ but, apart from identities, have only onto rigid redex morphisms and maximal onto rigid redex morphisms, respectively, as arrows. A maximal onto rigid redex morphism is one whose codomain can serve as the domain for only the identity onto rigid redex morphism (up to isomorphism).

By these means we obtain the Grothendieck categories $\mathscr{G}(\mathscr{P}, Rew_{\text{onto}})$ and $\mathscr{G}(\mathscr{P}, Rew_{\text{maxonto}})$, where Rew_{onto} and Rew_{maxonto} are the obvious functors. Thus, the Grothendieck construction not only shows us how rigid redex morphisms fit well with rewriting ideas, but gives us merge operations that identify (some, although not all) identical subgraphs for free. Previously, these have been treated as special-purpose extensions of the normal rewriting mechanism. Figure 20 shows a small example where a maximal onto rigid redex morphism identifies two subgraphs. Note that the third child of F cannot be identified with the other two because of rigidity.

7.2. *Other categories for rewriting*

Let us write Rew_* for any of $Rew, Rew_{\text{id}}, Rew_{\text{onto}}, Rew_{\text{maxonto}}$, thus using $*$ as a metavariable. We can easily move from the category $\mathscr{G}(\mathscr{P}, Rew_*)$ to categories that more conveniently describe rewriting. Firstly, we define the category $\mathscr{G}_*$, a category of graphs and rewrite sequences, by moving information around in the Grothendieck category. Its objects are graphs G. Its arrows are generated by arrows

$$\langle \phi_*, (i, r) : \langle g : L \to G \rangle \to \langle h : M \to H \rangle \rangle : G \to H,$$

which for notational purposes can be abbreviated as

$$\langle \phi_*, (i, r) : L \to M, g, h \rangle : G \to H.$$

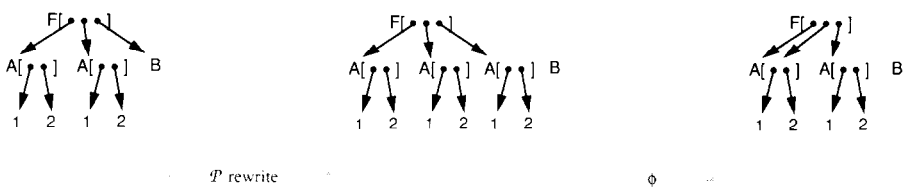


Fig. 20. A $\mathscr{G}$ rewrite $\langle \phi, \delta \rangle$ with nontrivial maximal onto ϕ.

Here $\delta \equiv (i,r): L \to M$ is a rule of $\mathscr{P}$, $g: L \to G$, $h: M \to H$ are rigid redexes, and H is given by $\phi_*: Rew^{\delta}(G) \to H$, where ϕ_* is an arrow in the fibre $\mathscr{G}_*^M$. Such an arrow of $\mathscr{G}_*$ is thus given by the ingredients of a $\mathscr{G}$ rewrite. To make $\mathscr{G}_*$ into a category we have to define composition of arrows. We therefore formally compose

$$\langle \delta \rangle \equiv \langle \phi_*, (i,r): L \to M_1, g, h_1 \rangle : G \to H$$

with

$$\langle \varepsilon \rangle \equiv \langle \psi_*, (j,s): M_2 \to N, h_2, k \rangle : H \to K$$

according to the formula

$$\langle \varepsilon \rangle \circ \langle \delta \rangle = \begin{cases} \langle \varepsilon \circ \delta \rangle \equiv \langle \psi_* \circ Rew^{\varepsilon}(\phi_*), (j \circ i, s \circ r): L \to N, g, k \rangle : G \to K \\ \quad \text{if } M_1 = M_2 \text{ and } h_1 = h_2, \\ \langle \varepsilon; \delta \rangle \equiv \langle \psi_*, (j,s): M_2 \to N, h_2, k; \phi_*, (i,r): L \to M_1, g, h_1 \rangle : G \to K \\ \quad \text{otherwise.} \end{cases}$$

We call the composition $\varepsilon \circ \delta$ internal in the first case above. More generally, if $\langle \delta; \ldots \rangle : B \to H$ and $\langle \ldots; \varepsilon \rangle : H \to C$, with δ, ε given as above are arrows of $\mathscr{G}_*$, their composition is defined by

$$\langle \ldots; \varepsilon \rangle \circ \langle \delta; \ldots \rangle : B \to C = \begin{cases} \langle \ldots; \varepsilon \circ \delta; \ldots \rangle : B \to C \text{ if } M_1 = M_2 \text{ and } h_1 = h_2, \\ \langle \ldots; \varepsilon; \delta; \ldots \rangle : B \to C \quad \text{otherwise.} \end{cases}$$

Thus, the arrows of $\mathscr{G}_*$ are identities and all possible rewriting sequences between two graphs, where rewrites directly composable in $\mathscr{G}(\mathscr{P}, Rew)$ are combined internally. The choice of composing ε and δ internally, rather than by formal concatenation whenever this is possible, leads to a functor $T_*: \mathscr{G}(\mathscr{P}, Rew_*) \to \mathscr{G}_*$ given by

$$T_*(\langle g: L \to G \rangle) = G,$$

$$T_*(\langle \phi_*, (i,r): L \to R \rangle : \langle g: L \to G \rangle \to \langle h: R \to H \rangle)$$

$$= \langle \phi_*, (i,r): L \to R, g, h \rangle : G \to H,$$

which would not exist if composition of δ and ε were done by formal concatenation in every case.

By forgetting various subsets of the information contained in the arrows of $\mathscr{G}_*$, we obtain categories that describe graph rewriting at various levels of abstraction.

Firstly, we can construct the category $\mathscr{G}'_*$ whose objects are graphs and which has an arrow

$$\langle \ldots; \phi_*, (i,r): L \to R, g; \ldots \rangle : B \to C$$

whenever $\mathscr{G}_*$ has an arrow

$$\langle \ldots; \phi_*, (i,r): L \to R, g, h; \ldots \rangle : B \to C.$$

Since the homomorphism h in an arrow of $\mathscr{G}_*$ can always be reconstructed from $\phi_*, (i,r): L \to R$ and g, the obvious forgetful functor $\mathscr{U}'_*: \mathscr{G}_* \to \mathscr{G}'_*$ is in fact an isomorphism.

Further, we can construct categories $\mathscr{G}''_*, \mathscr{G}'''_*, \mathscr{G}''''_*, \mathscr{G}^0_*$ whose objects are graphs and whose arrows are, respectively,

$$\langle \ldots; \phi_*, (i,r): L \to R; \ldots \rangle : B \to C,$$

$$\langle \ldots; (i,r): L \to R; \ldots \rangle : B \to C,$$

$$\langle \ldots; \phi_*; \ldots \rangle : B \to C,$$

$$\langle\ \rangle : B \to C,$$

whenever there is an arrow

$$\langle \ldots; \phi_*, (i,r): L \to R, g, h; \ldots \rangle : B \to C$$

in $\mathscr{G}_*$. The obvious forgetful functors $\mathscr{U}''_*: \mathscr{G}_* \to \mathscr{G}''_*, \ldots, \mathscr{U}^0_*: \mathscr{G}_* \to \mathscr{G}^0_*$ are full.

The variously primed $\mathscr{G}_*$ categories are functorial images of $\mathscr{G}(\mathscr{P}, Rew_*)$ only by virtue of the fact that composition of arrows of $\mathscr{G}_*$ is done internally wherever possible. For $\mathscr{G}^0_*$, a preorder whose arrows record simply that a rewriting sequence exists between the source and the target, a functor $T^0_*: \mathscr{G}(\mathscr{P}, Rew_*) \to \mathscr{G}^0_*$ exists regardless of internal composition. The functor is given by

$$T^0_*(g: L \to G\rangle) = G,$$

$$T^0_*(\langle \phi_*, (i,r): L \to R\rangle : \langle g: L \to G\rangle \to \langle h: R \to H\rangle) = \langle\ \rangle : G \to H.$$

If we do not care about rewriting categories being functorial images of $\mathscr{G}(\mathscr{P}, Rew_*)$, we can construct alternative $\mathscr{G}'^{\cdots'}_*$ categories (call them $\bar{\mathscr{G}}'^{\cdots'}_*$), whose composition of arrows is always concatenation. However, the composition actually used turns out to be fairly natural when we consider specific systems rather than the world of "all conceivable rewrites" which we are still concerned with.

7.3. Categories for specific rewrite systems

A specific system is given by a set of rules $\mathscr{R}$ which is a subset of the arrows of $\mathscr{P}$. Consider $\mathscr{G}^{\mathscr{R}}_*$, the largest subcategory of $\mathscr{G}_*$ for which each member of the sequence $\langle \ldots; \phi_*, (i,r): L \to R, g, h; \ldots \rangle$ that constitutes an arrow of $\mathscr{G}_*$ has $(i,r): L \to R$ in $\mathscr{R}$. This is well defined provided no two rules of $\mathscr{R}$ can be composed in $\mathscr{P}$. For convenience, we will assume that no two rules of $\mathscr{R}$ can be composed in $\mathscr{P}$ until further notice.

When considering a specific system $\mathscr{R}$, one is normally not interested in all rewritings that can be generated using $\mathscr{R}$, but in those that can be generated from a particular starting point. For example, by convention, DACTL systems are assumed to start rewriting from the so-called initial graph whose only node is labelled with the symbol Initial with empty arity. Obviously, to make progress, a system $\mathscr{R}$ must possess a rule whose left (sub)pattern matches Initial.

We will take a similar position with $\mathscr{G}$ rewriting. Let I, therefore, be some graph we decree to be initial. Let $\mathscr{G}_*^{\mathscr{R}I}$ be the largest subcategory of $\mathscr{G}_*^{\mathscr{R}}$ such that, for every object G, there is some arrow $\langle \ldots \rangle : I \to G$. Thus, $\mathscr{G}_*^{\mathscr{R}I}$ describes all rewritings generated from $\mathscr{R}$ starting with I, and I is by definition a weak initial object for $\mathscr{G}_*^{\mathscr{R}I}$.

The categories $\mathscr{G}_*^{\mathscr{R}}$ and $\mathscr{G}_*^{\mathscr{R}I}$ generate variously primed images under the suitably restricted forgetful functors $\mathscr{U}_*^{\prime\cdots\prime}$, and these describe rewriting according to a specific system at various levels of abstraction.

For practical purposes, the categories $\mathscr{G}_*^{\mathscr{R}I}$ are of more interest than the $\mathscr{G}_*^{\mathscr{R}}$ because, in the world of general graphs, it is all too easy to construct counterexamples to properties which we might consider useful or desirable. Thus, $\mathscr{G}_*^{\mathscr{R}}$ will possess few interesting properties. On the other hand, in $\mathscr{G}_*^{\mathscr{R}I}$, any property that holds for I, and is preserved by all members of $\{\phi_* \circ Rew^{\delta} \mid \delta \equiv (i,r) : L \to R \in \mathscr{R},\ \phi_* \in \mathscr{G}_*^{R}\}$, will be possessed by every object of $\mathscr{G}_*^{\mathscr{R}I}$ by induction.

7.4. Good granularity

We assumed above that, for a real system $\mathscr{R}$, no two rules for $\mathscr{R}$ are composable in $\mathscr{P}$. We return to this here, but first we need some concepts about roots of graphs, which will prove useful now and subsequently.

Definition 7.1. A root of a pattern or graph is always one of its nodes. If, in addition, it is an orphan node, we call it an o-root. If, furthermore, it is the unique o-root from which all other nodes are accessible, we call it a u-root.

Two arrows of $\mathscr{P}$, δ and ε are composable iff the right pattern of δ is the left pattern of ε. Usually, we are interested in systems such that the left pattern of each rule is u-rooted, although $\mathscr{P}$ rewriting happily describes much more general situations. This is because DACTL rules impose such a condition, and practice shows it to be convenient (particularly in allowing the redexes in a system to be coded up by just referring to their u-roots).

So if δ and ε are to be composable, then the right pattern of δ must be u-rooted. Usually this is not true for rules that we are interested in, since we frequently wish to redirect the root of the left pattern to a fresh value, and this value and (the copy in the right pattern of) the left root are mutually inaccessible. Thus, for collections of such rules, all the right patterns are disjoint from all the left patterns. This is not to say that one cannot construct rules which introduce new values and arrange the redirections in such a way that the right pattern is u-rooted, but such rules are rather bizzare and of no great interest for practical computations.

This leaves the class of rules for which no new nodes are introduced in the right pattern. We call these generalised selector rules since their sole effect is to rearrange the accessibility relation in a graph via redirection (such rules are also called collapsing rules in the literature). If the root redirection is the only nontrivial redirection of

the rule, the left and right patterns will actually be identical, and we call such rules true selector rules. In this case we have the following important property.

Proposition 7.2. *Application of a true selector rule to a redex is idempotent.*

Proof. If the source and the target of the redirection in a given redex are different, then the first application of the rule leaves the root of the redex without in-arcs, and so subsequent applications will be null. On the other hand, if the source and target are the same, then the first as well as all other applications will be null. □

A moment's thought shows that this idempotence holds even under purely locally universal rewriting.

True selector rules are important computationally, the I and K combinators and obvious rules for IF being important examples; so it is good to see that they are well behaved under composition. More general selector rules will have different left and right patterns since nonroot redirection will redirect arcs internal to the redex and this will show up as nonisomorphism between left and right patterns. Such rules are generally of much less interest computationally.

The likelihood that systems of real computational interest will have a composable pair of distinct rules is thus remote, most particularly since rules for different purposes will normally have left patterns whose roots are labelled with distinct symbols signalling their function. In the unlikely event of two or more rules being composable, it is worth considering the effects of "compiling" the rule set into a form where this no longer holds, by replacing composable sets of rules with their compositions. It may be that the resulting system exhibits different behaviour from the original system though. In any event we make the following definition.

Definition 7.3. A system of rules $\mathscr{R}$ has good granularity iff no two distinct rules of $\mathscr{R}$ are composable in $\mathscr{P}$, and all rules composable with themselves are true selector rules.

Our thesis is that useful systems have good granularity. The name refers to the fact that, for such systems, an automaton rewriting a graph according to the rules of the system in units of a single rewrite agrees with the functor $T_*: \mathscr{G}(\mathscr{P}, Rew_*) \to \mathscr{G}_*$ as to what the indivisible steps of the computation are, where, for $\mathscr{G}_*$, an indivisible step in a rewriting sequence is an element of the sequence that makes up an arrow. The only possible exceptions to this agreement are for identity rewrites, which we can live with, and rewrites using a selector rule where several applications of the same rule to the same redex would show up as distinct individual steps in the automaton's view of the rewriting history, but which would all be absorbed into a single element of an arrow of $\mathscr{G}_*$ because of the internal composition of composable steps in arrows of $\mathscr{G}_*$. However, it is a fairly safe assumption that a sensible automaton will "know" which of the rules of the system are selector rules, and thus will not repeatedly use the same rule

on the same redex if the result is guaranteed to be a null action. So, in practice, we expect this exception not to arise.

8. Abstract garbage collection

8.1. Notions of garbage

We have noted before that our various rewriting constructions feature garbage retention, but we have never said explicitly what we mean by garbage. There is a little more to this than meets the eye. Firstly, notions of garbage are intimately related to reduction strategies, since if we wish to discard a value we had better be confident that the reduction strategy will never subsequently cause us to access it again. Secondly, the graph rewriting paradigms of this paper can serve as the basis for different classes of graph rewriting, each with its own notions of appropriate reduction strategy, each thus leading to a notion of garbage natural within its own context. So there is no unique notion of garbage that fits all occasions. Let us immediately mention two classes of graph rewriting that require distinct notions.

(1) The first is the class of graph rewriting that is intended to model term rewriting. Here, each graph has an explicit root and every reduction strategy only ever selects redexes which are accessible from the root. Thus, a notion of garbage that declares all nodes accessible from the root as live and all other nodes as garbage is natural.

(2) The full DACTL language decorates the nodes and arcs of computational graphs as discussed in this paper, with node and arc markings designed to encode explicitly reduction strategies. Some of the term-oriented reduction strategies of class (1) are easily coded up using these markings, others less so. In addition, the full language is general enough to model many other programming styles, e.g. logic and concurrent logic programming, imperative programming, and communicating processes. For these paradigms, the kind of reduction strategies that are most natural can normally be coded via the markings very conveniently. Also (particularly when modelling communicating processes), it is often the case that having a single explicit root in the graph is not the most sensible way of characterising liveness and garbage. Under these circumstances, the most natural notions of liveness and garbage involve the markings, and perhaps some additional criteria.

When we apply a notion of garbage to a graph G, we obtain a graph with garbage $\langle G, \mathrm{Gar}(G)\rangle$, where $\mathrm{Gar}(G)$ is a subset of the nodes of G, the garbage set of G. The live set of G, denoted $\mathrm{Live}(G)$, contains all nodes of G not in $\mathrm{Gar}(G)$. In practice of course, $\mathrm{Live}(G)$ is got by applying some recursive definition of liveness to G, and $\mathrm{Gar}(G)$ arises as its complement.

From the vantage point of $\mathscr{G}$ rewriting, notions of garbage fall into two classes: deterministic notions and nondeterministic notions. A deterministic notion of garbage unambiguously maps each graph G to $\langle G, \mathrm{Gar}(G)\rangle$, while a non-deterministic notion may permit several different $\langle G, \mathrm{Gar}(G)\rangle$ objects to correspond to a given G. Deter-

ministic notions of liveness thus typically use only the data inherent in a graph G: the symbols labelling nodes, and topological notions such as accessibility.

Here are some examples. Let Root and Leaf be particular symbols, and let Acc be the accessibility relation, i.e. x Acc y iff there is a path (of length $\geqslant 0$) from x to y.

(DG-A) Every node labelled with Root is live. If x is live and x Acc y then y is live.

(DG-B) Every node labelled with Leaf is live. If y is live and x Acc y then x is live.

(DG-C) Every node labelled with Root or Leaf is live. If x is labelled with Root and x Acc y then y is live. If y is labelled with Leaf and x Acc y then x is live.

(DG-D) Every node labelled with Root is live. If x is labelled with Root and x Acc y then y is live. If x is labelled with Root, y is labelled with Leaf, x Acc y, z Acc y, then (y is live and) z is live.

(DG-A)–(DG-D) give four different deterministic notions of liveness. These are just proof systems. Given a graph G and a particular one of these notions, any node that can be proved live is in Live(G), and Gar(G) arises as Live(G)'s complement.

Nondeterministic notions of garbage arise when some element of choice is permitted, e.g.:

(NG-A) Choose a subset of the nodes and let it be Live(G).

(NG-B) Choose a subset of the nodes and let them be live. If x is live and x Acc y then y is live.

(NG-C) Choose a single node and let it be live. If x is live and x Acc y then y is live.

From the vantage point of $\mathscr{G}$ rewriting, nondeterministic notions of garbage usually arise through the suppression of extra detail. For instance, when term rewriting is modelled using $\mathscr{G}$ rewrites, the appropriate notion of garbage is (NG-C). This is because, in a term rewriting derivation, there is an implicit assumption that the root of the initial term of the derivation defines the root of the live subgraph of the term (which will be the whole of the initial term since terms are normally taken to be free of garbage). Furthermore, the structure of a derivation carries with it a root management strategy which determines what the root of the new graph will be, given the root of the old graph. Since these considerations are invisible to the $\mathscr{G}$ rewriting mechanism, the choice of the root of a graph that models some term in such a derivation seems arbitrary. Nevertheless, the choice can hardly be regarded as such, and it is often possible to model at least the deterministic parts of, say, a term-oriented root management strategy, within the graph paradigm itself. If one chooses the modelling carefully, it may be possible to reduce the notion of garbage (NG-C) to a notion like (DG-A). We will return to this later.

8.2. Notions of garbage and rewriting

Notions of garbage must also fit smoothly with rewriting; the root management strategy mentioned above is but one specific example.

Let Θ be a notion of garbage. Let P be a pattern. Let $\mathscr{G}^P_{*\Theta}$ have as objects all $\langle g : P \to \langle G, \mathrm{Gar}(G)\rangle\rangle$, where g is a homomorphism from P to a graph G and $\langle G, \mathrm{Gar}(G)\rangle$ is a graph with garbage permitted by Θ. Arrows of $\mathscr{G}^P_{*\Theta}$ are all possible

$\phi_*:\langle g_1:P\to\langle G_1,\mathrm{Gar}(G_1)\rangle\rangle\to\langle g_2:P\to\langle G_2,\mathrm{Gar}(G_2)\rangle\rangle$ whenever there is an arrow $\phi_*:\langle g_1:P\to G_1\rangle\to\langle g_2:P\to G_2\rangle$ in the corresponding category $\mathscr{G}^P_*$. Thus, $\mathscr{G}^P_{*\Theta}$ contains all possible ways of decorating objects of $\mathscr{G}^P_*$ with permissible garbage and all arrows arising thereby. Proceeding to construct $\mathscr{G}^P_{*\Theta}$ for all P in $\mathscr{P}$, we end up with the Grothendieck category $\mathscr{G}(\mathscr{P},Rew_*)_{\{\Theta\}}$ which contains all rewrites of $\mathscr{G}(\mathscr{P},Rew_*)$ decorated with garbage in all possible ways. Now some of these rewrites will be sensible in the context of the notion Θ, and others will not. We define the subcategory $\mathscr{G}(\mathscr{P},Rew_*)_\Theta$ of $\mathscr{G}(\mathscr{P},Rew_*)_{\{\Theta\}}$ by giving it all objects but only the valid arrows of $\mathscr{G}(\mathscr{P},Rew_*)_{\{\Theta\}}$.

Definition 8.1. Let

$$\delta\equiv\langle\phi_*,(i,r):L\to R\rangle:\langle g:L\to\langle G,\mathrm{Gar}(G)\rangle\rangle\to\langle h:R\to\langle H,\mathrm{Gar}(H)\rangle\rangle$$

be an arrow of $\mathscr{G}(\mathscr{P},Rew_*)_{\{\Theta\}}$. Each such arrow gives a pair of functions $(j,s):G\to H$, where $j=\phi_*\circ i^*$, $s=\phi_*\circ r^*$ and $(i^*,r^*):G\to Rew^\delta(G)$ is the concrete representation in terms of node maps of the opcartesian arrow for $(i,r):L\to R$ and $\langle g:L\to\langle G,\mathrm{Gar}(G)\rangle\rangle$. The arrow δ is valid with respect to Θ iff (j,s) satisfy (GAR) below.

(GAR) For all $x\in\mathrm{Gar}(G)$,
- (a) $j(x)\in\mathrm{Gar}(H)$,
- (b) j is 1–1 on $\mathrm{Gar}(G)$,
- (c) $s(x)=j(x)$,
- (d) $y\in G$, and $s(x)=s(y)\Rightarrow x=y$,
- (e) $(p_l,j(x))$ an arc of $H\Leftrightarrow\exists\,(t_l,x)$ an arc of G such that $j(t)=p$.

The implications of (GAR) are fairly obvious. Garbage must be persistent ((a)) and must not be manipulated by rewrites in a nontrivial way. Thus, (b) guarantees that garbage is preserved 1–1; (c), (d) and (e), respectively, guarantee that garbage nodes may not be sources or targets of redirections, and that they may not acquire additional parents. It is clear that (GAR) is such that $\mathscr{G}(\mathscr{P},Rew_*)_\Theta$ is indeed a category.

Definition 8.2. Let $\delta\equiv(i,r):L\to R$ be an arrow of $\mathscr{P}$. The SL set (surely live set) of δ, written $\mathrm{SL}(\delta)$, consists of nodes of L such that (SL) below holds.

(SL) For all $x\in\mathrm{SL}(\delta)$, either
- (a) $r(x)\neq i(x)$, or
- (b) there is an $x\neq y\in L$ such that $r(x)=r(y)$, or
- (c) there is an arc $(p_l,i(x))$ in R such that there is no arc (t_l,x) in L with $i(t)=p$.

Theorem 8.3. *Let*

$$\langle\phi_*,\delta\rangle:\langle g:L\to\langle G,\mathrm{Gar}(G)\rangle\rangle\to\langle h:R\to\langle H,\mathrm{Gar}(H)\rangle\rangle$$

be an arrow of $\mathscr{G}(\mathscr{P},Rew_*)_\Theta$, *with* $\delta\equiv(i,r):L\to R$. *Then* $g(\mathrm{SL}(\delta))\subseteq\mathrm{Live}(G)$.

Proof. Since $\langle \phi_*, \delta \rangle$ is an arrow of $\mathscr{G}(\mathscr{P}, Rew_*)_\Theta$, it is certainly valid as an arrow of $\mathscr{G}(\mathscr{P}, Rew_*)_{\{\Theta\}}$. Thus, its concrete representation $(j,s): G \to H$ satisfies (GAR). It is now easy to check that $g(\mathrm{SL}(\delta)) \cap \mathrm{Gar}(G) = \emptyset$. □

Lemma 8.4. *Let $\delta \equiv (i,r): L \to M$ and $\varepsilon \equiv (j,s): M \to N$ be two arrows of $\mathscr{P}$. Then*

$$\mathrm{SL}(\varepsilon \circ \delta) = (\mathrm{SL}(\delta) \cup i^{-1}(\mathrm{SL}(\varepsilon))) - X_{\varepsilon \circ \delta},$$

where

$$X_{\varepsilon \circ \delta} = \{ x \in L \mid j \circ i(x) = s \circ r(x), \text{ and there is no } x \neq y \in L \text{ such that } s \circ r(x) = s \circ r(y), \text{ and } [(p_l, j \circ i(x)) \text{ an arc of } N \Leftrightarrow \exists\, (t_l, x) \text{ an arc of } L \text{ such that } j \circ i(t) = p] \}.$$

Proof. Call the complement of an SL set a PG set (potentially garbage set). The PG set of δ contains all nodes which satisfy the conjunction of the negations of (SL)(a)–(c). Clearly, $\mathrm{PG}(\delta) \cap i^{-1}(\mathrm{PG}(\varepsilon))$ also satisfies the conjunction of the negations of (SL)(a)–(c) for $\varepsilon \circ \delta$, so $\mathrm{SL}(\delta) \cup i^{-1}(\mathrm{SL}(\varepsilon))$ is an upper bound for $\mathrm{SL}(\varepsilon \circ \delta)$. We must remove from this set all nodes for which ε conspires precisely to undo the effect of δ in terms of qualification for membership of $\mathrm{SL}(\varepsilon \circ \delta)$. A simple comparison of the three clauses in (SL) with the three clauses defining $X_{\varepsilon \circ \delta}$ shows that $X_{\varepsilon \circ \delta}$ is the correct set to remove. □

Corollary 8.5. *The surely live concept is categorical.*

Proof. This comes down to showing that, if $\delta_n \equiv (i_n, r_n): M_n \to M_{n+1}$, with $n = 1, 2, 3$, are three composable arrows of $\mathscr{P}$, then

$$\mathrm{SL}(\delta_3 \circ (\delta_2 \circ \delta_1)) = \mathrm{SL}((\delta_3 \circ \delta_2) \circ \delta_1)$$
$$= (\mathrm{SL}(\delta_1) \cup i_1^{-1}(\mathrm{SL}(\delta_2)) \cup (i_2 \circ i_1)^{-1}(\mathrm{SL}(\delta_3))) - X,$$

where

$$X = \{ x \in L \mid i_3 \circ i_2 \circ i_1(x) = r_3 \circ r_2 \circ r_1(x), \text{ and there is no } x \neq y \in L \text{ such that } r_3 \circ r_2 \circ r_1(y) = r_3 \circ r_2 \circ r_1(x), \text{ and } [(p_l, i_3 \circ i_2 \circ i_1(x)) \text{ an arc of } M_4 \Leftrightarrow \exists\, (t_l, x) \text{ an arc of } L \text{ such that } i_3 \circ i_2 \circ i_1(t) = p] \},$$

which is straightforward if tedious. □

We can therefore embellish the data for an arrow $\langle \phi_*, \delta \rangle$ of $\mathscr{G}(\mathscr{P}, Rew_*)_\Theta$ to include the SL set of δ. (We do not bother to define a whole new category to express this, although strictly speaking we should.) Thus,

$$\langle \phi_*, (i, r, \mathrm{SL}(\delta)): L \to R \rangle : \langle g: L \to \langle G, \mathrm{Gar}(G) \rangle \rangle \to \langle h: R \to \langle H, \mathrm{Gar}(H) \rangle \rangle.$$

Theorem 8.3 and the remarks that follow it (which make a fairly loose connection between garbage in semantic objects (graphs) and sure liveness (alternatively, potential garbage) in syntactic arrows (rules between patterns)) constitute all that can be said for garbage in the general case. In particular, note that the concept of garbage set cannot be pulled down into the base category $\mathscr{P}$. This is most clearly illustrated by the fact that an SL set, despite being a subcomponent of an object, is nevertheless a property of an arrow of $\mathscr{P}$. This mismatch between garbage in the Grothendieck category being a property of objects, and sure liveness in the base being a property of arrows, is another case where local universality does not extend to global universality and prevents the obvious projection $F:\mathscr{G}(\mathscr{P}, Rew_*)_\Theta \to \mathscr{P}$ from being an opfibration for an arbitrary notion of garbage Θ. The situation is directly comparable to those discussed in Sections 5.1 and 6.4. Nevertheless, in benign circumstances, the problems caused by this mismatch can be overcome, and an opfibration can be recovered. Section 9, on term rewriting, illustrates just such a case.

In general, therefore, there is nothing to force $Rew^\delta(G)$ to be opcartesian for δ and G in $\mathscr{G}(\mathscr{P}, Rew_*)_\Theta$, although it is opcartesian in $\mathscr{G}(\mathscr{P}, Rew_*)$. This is connected with the fact that the liveness of a node is a property that depends on the global structure of the graph rather than on just the structure of a redex. In other graph rewriting paradigms this is not necessarily the case, and a more precise story on garbage can be told (see e.g. [7], where the opposite view to ours prevails).

With these considerations in place, we can mimic the earlier development of categories that describe rewriting. Firstly, we define $\mathscr{G}_{*\Theta}$ whose objects are graphs with garbage $\langle G, \mathrm{Gar}(G)\rangle$ and whose arrows are generated by arrows such as

$$\langle \phi_*, (i, r, \mathrm{SL}(\delta)) : \langle g : L \to G\rangle \to \langle h : R \to H\rangle\rangle : \langle G, \mathrm{Gar}(G)\rangle \to \langle H, \mathrm{Gar}(H)\rangle.$$

In general, arrows of $\mathscr{G}_{*\Theta}$ are sequences, as was the case with $\mathscr{G}_*$. If composition is done internally whenever appropriate as before, then we get a functor $T_{*\Theta}:\mathscr{G}(\mathscr{P}, Rew_*)_\Theta \to \mathscr{G}_{*\Theta}$ given by

$$T_{*\Theta}(\langle g : L \to \langle G, \mathrm{Gar}(G)\rangle\rangle) = \langle G, \mathrm{Gar}(G)\rangle,$$

$$T_{*\Theta}(\langle \phi_*, (i, r, \mathrm{SL}(\delta)) : L \to R\rangle : \langle g : L \to \langle G, \mathrm{Gar}(G)\rangle\rangle \to \langle h : R \to \langle H, \mathrm{Gar}(H)\rangle\rangle)$$
$$= \langle \phi_*, (i, r, \mathrm{SL}(\delta)) : L \to R, g, h\rangle : \langle G, \mathrm{Gar}(G)\rangle \to \langle H, \mathrm{Gar}(H)\rangle.$$

As before, we can construct a wealth of variously primed counterparts of $\mathscr{G}_{*\Theta}$. In addition, the reader will have no trouble imagining what $\mathscr{G}^{\mathscr{R}}_{*\Theta}$ and $\mathscr{G}^{\mathscr{R}!}_{*\Theta}$ stand for.

8.3. Factoring out garbage; reduction strategies

As well as being able to construct the usual selection of forgetful functors on $\mathscr{G}(\mathscr{P}, Rew_*)_\Theta$, we have another more interesting possibility, that of identifying graphs modulo garbage. Indeed, this corresponds to the normal way of viewing computations, since, if one were to regard computations as carrying all their garbage with them, the total number of nodes in the computational graph would be always

nondecreasing, and recursively enumerable sets would all be recursive. Noting that for some notions of garbage, e.g. (NG-A) or (DG-B), there may be garbage nodes accessible from some live nodes, we need the following definitions.

Definition 8.6. Let $\langle G, \mathrm{Gar}(G)\rangle$ be a graph with garbage. The live subgraph of $\langle G, \mathrm{Gar}(G)\rangle$, $\mathrm{LSG}(\langle G, \mathrm{Gar}(G)\rangle)$, consists of all nodes in $\mathrm{Live}(G)$ and all arcs for which parent and child nodes are both in $\mathrm{Live}(G)$.

Definition 8.7. A homomorphism of live subgraphs $h: \mathrm{LSG}(\langle G, \mathrm{Gar}(G)\rangle) \to \mathrm{LSG}(\langle H, \mathrm{Gar}(H)\rangle)$ is a map on the nodes of $\mathrm{Live}(G)$ such that

(a) for all $x \in \mathrm{Live}(G)$,

(i) $h(x) \in \mathrm{Live}(H)$,

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

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

(b) for all (x_l, y), an arc of the live subgraph of G, $(h(x)_l, h(y))$ is an arc of the live subgraph of H.

Once we have a notion of homomorphism of live subgraphs, we can define isomorphism classes of live subgraphs, and then select an abstract representative from each class. Assume in future that the notation $\mathrm{LSG}(\langle G, \mathrm{Gar}(G)\rangle)$ refers to this object.

We define the category $\mathscr{G}_{*[\Theta]}$. Its objects are (the abstract representatives of) the live subgraphs of the graphs with garbage which are the objects of $\mathscr{G}_{*\Theta}$. Its arrows are generated in the by now familiar way from arrows such as

$$\langle \phi_*, (i, r, \mathrm{SL}(\delta)) : \langle g : L \to \langle G, \mathrm{Gar}(G)\rangle\rangle \to \langle h : R \to \langle H, \mathrm{Gar}(H)\rangle\rangle\rangle :$$
$$\mathrm{LSG}(\langle G, \mathrm{Gar}(G)\rangle) \to \mathrm{LSG}(\langle H, \mathrm{Gar}(H)\rangle).$$

We see that the graphs G and H are now an indispensable part of the arrow data since they are no longer implicit in the source or target data. As usual, we define composition internally wherever possible, and we need to use the G and H in the arrow data to decide this since

$$\langle \delta \equiv \langle \ldots \to \langle h_1 : M_1 \to \langle H_1, \mathrm{Gar}(H_1)\rangle\rangle\rangle; \ldots\rangle : B \to X$$

and

$$\langle \ldots; \varepsilon \equiv \langle \ldots \to \langle h_2 : M_2 \to \langle H_2, \mathrm{Gar}(H_2)\rangle\rangle\rangle\rangle : X \to C$$

are not composable internally unless $M_1 = M_2$, $\langle H_1, \mathrm{Gar}(H_1)\rangle = \langle H_2, \mathrm{Gar}(H_2)\rangle$ and $h_1 = h_2$. This follows since it is possible that

$$\mathrm{LSG}(\langle H_1, \mathrm{Gar}(H_1)\rangle) = X = \mathrm{LSG}(\langle H_2, \mathrm{Gar}(H_2)\rangle)$$

without having $H_1 = H_2$. There is an obvious forgetful functor $\mathscr{U}_{[\Theta]} : \mathscr{G}_{*\Theta} \to \mathscr{G}_{*[\Theta]}$ given by sending each graph with garbage to its live subgraph.

The category $\mathscr{G}_{*[\Theta]}$ describes rewriting where one collects the garbage at the computational graph level as one goes. Obviously, we can write down the usual large collection of primed counterparts of $\mathscr{G}_{*[\Theta]}$: likewise $\mathscr{G}^{\mathscr{R}}_{*[\Theta]}$ and $\mathscr{G}^{\mathscr{R}I}_{*[\Theta]}$.

This is also an appropriate point to bring up the matter of rewriting strategies. We take a rewriting strategy to be a relation v such that, for each object X, $X\, v\, Y \Leftrightarrow [X$ can be rewritten (in zero or more steps) to $Y]$. We let $\mathscr{G}_{*[\Theta]v}$ be the subcategory of $\mathscr{G}_{*[\Theta]}$ generated by the arrows given by v. This describes rewriting under the strategy v; likewise the categories $\mathscr{G}^{\mathscr{R}}_{*[\Theta]v}$ and $\mathscr{G}^{\mathscr{R}I}_{*[\Theta]v}$, and their variously primed derivatives.

8.4. Example: rooted rewriting

Of the rewriting constructions we have discussed, only the DACTL construction mentions a root. Since the DACTL model is equivalent to the D model, the use of roots is not strictly needed in the semantics of DACTL. Nevertheless, rooted systems form an important class of systems. In general, we need only require that the LHS of a rule has a u-root, and that the notion of liveness in force includes accessibility from a root of the computational graph, to be within the sphere of rooted systems, and this is the basis on which most of this section is built. There is an important subclass of this class, wherein we require also that the LHS u-root is the one and only redirection of the rule, that all patterns of rules are acyclic, that the RHS pattern contains two o-roots, one of which is the image of the LHS u-root, and that the redirection redirects the LHS u-root to the other o-root of the RHS. This class models term rewriting systems within term graph rewriting, and was the impetus for the development of term graph rewriting in the first place.

During a rewrite, the roots of the computational graph and the roots of the patterns in the rule must tie together as follows. If X is the graph being rewritten, with root t, and the rule is $\langle P, root, Red\rangle$, then if Y is the result of the rewrite the root of Y depends on whether the image of *root* in the matching of the left subpattern of P was t or not. If it was, then the root of Y is the node that t got redirected to; if not, it is the injective image of t in Y.

There are various ways of modelling the above using $\mathscr{G}$ rewriting. The key issue is to fix on a suitable notion of garbage. Possible candidates are (DG-A) and (NG-C). However, we will adhere to the notion (ROOT), a notion related to (DG-A), for reasons that will become clear soon. In this respect we assume that Root is a symbol that does not occur in any system of interest to us and always occurs with arity $\{1\}$. Root is thus a "special symbol".

(ROOT) The unique child of any node labelled with Root is live. If x is live and x Acc y then y is live.

With this single choice the model of rewriting we require follows immediately. Consider the category $\mathscr{P}$. It contains a subcategory $\mathscr{P}_{\mathsf{Root}}$, the largest subcategory of $\mathscr{P}$ for which no object contains an occurrence of the Root symbol. For P in $\mathscr{P}_{\mathsf{Root}}$ consider the fibre $\mathscr{G}^{P}_{\mathrm{id}}$. It consists of all graphs containing an instance of P. Now

$\mathscr{G}_{\mathrm{id}}^{P}$ splits into equivalence classes of graphs according to how many Root-labelled nodes a graph has. We focus on the class whose graphs have precisely one such node and on the subclass of that class for which the Root-labelled node is an orphan. Selecting all arrows whose source and target are both in this subclass, we obtain a subcategory $\mathscr{G}_{\mathrm{id1Root}}^{P}$. We regard its objects as graphs with garbage in the usual way. Now since the criteria that govern $\mathscr{G}$ rewriting are quite independent of those involving Root, we can make a Grothendieck construction with base $\mathscr{P}_{\mathsf{Root}}$ and fibres $\mathscr{G}_{\mathrm{id1Root}}^{P}$. We obtain the Grothendieck category $\mathscr{G}(\mathscr{P}_{\mathsf{Root}}, Rew_{\mathrm{id1}})_{\{\mathrm{ROOT}\}}$ (where Rew_{id1} is the obvious functor). From this we generate the category $\mathscr{G}(\mathscr{P}_{\mathsf{Root}}, Rew_{\mathrm{id1}})_{\mathrm{ROOT}}$, which has only valid rewrites.

We claim that the arrows of $\mathscr{G}(\mathscr{P}_{\mathsf{Root}}, Rew_{\mathrm{id1}})_{\mathrm{ROOT}}$ do root management in exactly the way we need. We note that every rewrite cannot but preserve the Root-labelled node of a graph injectively since the Root-labelled node is never part of a redex and the root of the live subgraph is its child. If we express rewrites in concrete terms by $(j, s): G \to H$ then the root of the live subgraph of H is the value of s at the root of the live subgraph of G.

We can now follow the standard construction and manufacture $\mathscr{G}_{\mathrm{id1ROOT}}$ whose objects are graphs with garbage and whose arrows are rewriting sequences, and then we can forget the garbage and get $\mathscr{G}_{\mathrm{id1[ROOT]}}$. Restricting ourselves to a specific system $\mathscr{R}$ and initial graph I, we get, in turn, $\mathscr{G}_{\mathrm{id1[ROOT]}}^{\mathscr{R}}$ and $\mathscr{G}_{\mathrm{id1[ROOT]}}^{\mathscr{R}I}$, etc.

On the basis of the preceding considerations, it is clear that the notion (ROOT) captures DACTL's root management strategy particularly well when used in the context of rooted rewriting, regardless of whether or not we are in the special case for term rewriting, showing, incidentally, that the invariants that describe this situation are independent of the problems of rootedness or garbage.

The above treatment used (ROOT), a deterministic notion of garbage. We point out briefly how a treatment using (NG-C) would go. The main difference between deterministic and non-deterministic notions of garbage is that the former are given by a function from graphs to graphs with garbage. This function can often be enriched to a functor for various categories of graphs and this can give us extra structure for free. Suppose then we were using (NG-C) to model the previous situation. The fibre above a pattern P would consist of pointed graphs, i.e. graphs with a distinguished node, the root. We could accomplish what we achieved before, but at every stage we would have had to enhance every functor or construction by specifying how it acted on the distinguished node. All of these enhancements would have been fairly obvious, but all would have meant a slight stepping outside of the framework already constructed. Using the deterministic notion (ROOT) avoided all this.

It has to be said that, in many ways, the (ROOT) notion is a more honest way of doing things. In a genuine implementation of rewriting, the automaton doing the rewriting would indeed have a fixed anchor point such as a Root-labelled node, from which the live part of the computational graph was accessible.

Counterexample 8.8. This is a good place to substantiate the statement made earlier that, in general, the projection $F: \mathscr{G}(\mathscr{P}, Rew_{*})_{\Theta} \to \mathscr{P}$ is not an opfibration, even for

rooted rewriting. Consider Example 3.10 in the $\mathscr{G}$ formulation as illustrated in Fig. 16, using the notion (ROOT). This rule performs two nontrivial redirections, the root redirection root→root′ and the subroot redirection c→c′. Consider factoring this rule into two in $\mathscr{P}$. The first factor δ performs only the root redirection and its RHS pattern is the pattern P of Fig. 3. The second factor ε performs only the subroot redirection, its LHS being the pattern P just mentioned and its RHS pattern being R of Fig. 16. Clearly, $\varepsilon \circ \delta$ is the original rule.

If we rewrite a redex $\langle g : L \to G \rangle$ such that every path from the root of G to $g(\mathsf{c})$ passes through $g(\mathsf{root})$, then the image in the right-hand redex of c will be garbage because of the redirection root→root′. If we do both redirections simultaneously in the composite form of the rule, no harm is done. However, if we use the factored form of the rewrite, then the rewrite using ε will not be valid since the subroot redirection redirects a garbage node. Since the rewrite using ε is precisely the unique factorisation arrow of the composite rewrite needed for $F : \mathscr{G}(\mathscr{P}, Rew_*)_\Theta \to \mathscr{P}$ to be an opfibration, we have our claim.

Readers should prove, however, that when we restrict ourselves to the subcase for describing term rewriting via term graph rewriting, the functor $F : \mathscr{G}(\mathscr{P}, Rew_*)_\Theta \to \mathscr{P}$ does indeed turn out to be an opfibration. This is to be expected on the basis of the contents of Section 9, given the results that relate term rewriting and term graph rewriting in e.g. [4, 11, 24].

9. Term rewriting using opfibrations

The previous sections developed general term graph rewriting in an opfibration framework in fair detail. Since the style of graph rewriting we are most concerned with in this paper is inspired by and is a generalisation of term rewriting, it is instructive to examine the development of term rewriting in a similar framework. We do this briefly below.

9.1. Term rewriting with garbage retention

Suppose we have an alphabet of symbols which is the disjoint union of function symbols (with nonempty arity), constant symbols and variable symbols (the latter two with empty arity). A term is a tree-shaped graph, each of whose nodes is labelled with a symbol from the alphabet. A term is closed if it contains no leaves labelled by variables. A term is open if it contains zero or more leaves labelled by variables (note the disparity with conventional usage). A term homomorphism or matching is a graph homomorphism between terms, such that if a node is labelled with a constant or function then its image must be labelled by the same symbol (with the same arity). However, nodes labelled with a variable symbol may match anything, with the proviso that the subterms rooted at the images of distinct nodes labelled with the same

variable must be isomorphic. The concept of term homomorphism extends naturally to sets and multisets of terms and we will use the extended notion below without further comment. A term homomorphism is strict iff the image of a node labelled with a variable is labelled with the same variable.

The base category for the rewriting enterprise is $\mathcal{MO}$ whose objects are multisets of open terms and whose arrows are $(i, r): L \to R$. Here (i, r) satisfy (TR-INJ), (TR-RED) and (TR-SUB) below.

(TR-INJ) $i: L \to R$ is a multiset of strict term homomorphisms and maps roots of terms of L to roots of terms of R.

(TR-RED) $r: L \to R$ is defined only on roots of terms of L and maps them injectively into roots of terms of R.

(TR-SUB) Every variable symbol of R occurs also in L.

These arrows are generalised term rewrite rules as we will presently see. The fibre above some P in $\mathcal{MO}$ is a category $\mathcal{MT}^P$ of multisets of closed terms which are homomorphic instances of P. More explicitly, each object of $\mathcal{MT}^P$ is a pair $\langle S, s\rangle$ such that $s: P \to S$ is a term homomorphism. An arrow $\phi: \langle S, s\rangle \to \langle S', s'\rangle$ of $\mathcal{MT}^P$ is a rigid redex morphism, i.e. it is a homomorphism $\phi: S \to S'$, such that $s' = \phi \circ s$ and $\phi^{-1}(s'(P)) = s(P)$.

A rule $\delta \equiv (i, r): L \to R$ of $\mathcal{MO}$ induces a functor $TRew^\delta: \mathcal{MT}^L \to \mathcal{MT}^R$ whose object map is a form of parallel term rewriting patterned after (an operational version of) $\mathcal{P}$ rewriting. Thus, let $s: L \to S$ be a redex. Add fresh copies of all terms of $R - i(L)$ to S giving a multiset of (potentially open) terms S'. Redirect each arc of S' targeted at $s(root(l))$, so that it points instead at the copy of $r(root(l))$ in S', where l is an open term of L, $root(l)$ is its root and s is the obvious extension of $s: L \to S$ to $s: L \to S'$. Call the resulting multiset S''. Replace each leaf of an open term of S'' labelled with a variable symbol V by a fresh copy of the subterm rooted at $s(a)$, where a is any leaf of an open term of L labelled with V. This gives a multiset of (closed) terms T. Then T is in $\mathcal{MT}^R$ (because r is injective) with obvious homomorphism $t: R \to T$. Clearly, $TRew^\delta$ extends to rigid homomorphisms of $\mathcal{MT}^L$ in the obvious way.

The reader will not doubt that the above construction can be rephrased in a locally universal manner. We call this (operational) rewriting construction, and also its locally universal counterpart (which has exactly the same expressive power), the $\mathcal{MT}$ rewriting construction. The local universality extends to a global universality, i.e. the functors $TRew^\delta$ merge together to give a functor $TRew: \mathcal{MO} \to \mathcal{C}at$ and we have another opfibration. Call the resulting Grothendieck category $\mathcal{MT}(\mathcal{MO}, TRew_*)$, where we can restrict the rigid homomorphisms in the $\mathcal{MT}^P$ categories in various ways as in the previous section, and parametrise the construction using $*$ as before. When necessary, we can refer to this generalisation of $\mathcal{MT}$ rewriting as the $\mathcal{MT}_*$ rewriting construction. In any of these variations on the same theme, a rewrite can be written in concrete notation as $(j, f): S \to T$, where S and T are multisets of closed terms, j is the obvious symbol/arity-preserving node function, f is a redirection function, and so (j, f) form a redirection couple, i.e. [(x_l, y) an arc of $S \Leftrightarrow (j(x)_l, f(y))$ an arc of T].

9.2. Notions of garbage for term rewriting

Let us now consider garbage, in particular the notion (ROOT) that we used previously. As before, we can identify subcategories of the fibres whose objects have exactly one orphan Root-labelled node, and a subcategory of $\mathscr{MC}$, none of whose objects contain any Root-labelled nodes. A rewrite of a redex $s: L \to S$ using $(i, r): L \to R$, yielding $(j, f): S \to T$ and $t: R \to T$, is valid iff it satisfies (TR-GAR) below.

(TR-GAR) (a) $j(\mathrm{Gar}(S)) \subseteq \mathrm{Gar}(T)$,
(b) j is 1–1 on $\mathrm{Gar}(S)$,
(c) $f = j$ on $\mathrm{Gar}(S)$,
(d) $x \in \mathrm{Gar}(S), y \in S, f(x) = f(y) \Rightarrow x = y$,
(e) $\mathrm{VarS}(t^{-1}(\mathrm{Live}(T))) \subseteq \mathrm{VarS}(s^{-1}(\mathrm{Live}(S)))$,

where $\mathrm{VarS}(X)$ is the set of variable symbols labelling leaves of X. In fact, a little thought shows that (TR-GAR)(a) implies a dual property to (TR-GAR)(e), namely

(TR-GAR) (a′) $\mathrm{VarS}(s^{-1}(\mathrm{Gar}(S))) \subseteq \mathrm{VarS}(t^{-1}(\mathrm{Gar}(T)))$.

Amongst other things, these properties say that garbage is persistent, and that any variable labelling any node of R matched to a live node of T also labels some node of L matched to a live node of S (and dually). This latter property is the analogy in the present context of the idea that "rewrites may not manipulate garbage nodes in a nontrivial manner". Note the contrast between (TR-GAR)(e), which uses the structure of the rule governing the rewrite in an essential way, and the last three clauses of (GAR) in the previous section which make no reference to the rule used. This highlights the fact that term rewriting semantics uses the isomorphism of subterms rooted at images of similarly labelled variable nodes, while graph rewriting semantics has no such "nonlocal" property. This nonlocality obscures the meaning of the phrase "rewrites may not manipulate garbage nodes in a nontrivial manner", particularly where the left pattern of a rule is not linear and some variable has both live and garbage instances in some redex. Presumably, instantiating a live variable node of S'' with a subterm obtainable only by inspecting a garbage subterm of S counts as nontrivial manipulation and is therefore prohibited by (TR-GAR)(e). But we could go further and replace (TR-GAR)(e) by the stronger (TR-GAR)(e′), or the still stronger (TR-GAR)(e″) below if we wished, to get alternative notions of validity. Again we can view the whole construction as being parametrised by the choice of one or other of these three possibilities.

(TR-GAR)(e′)

$$\mathrm{VarS}(s^{-1}(\mathrm{Gar}(S))) \cap \mathrm{VarS}(t^{-1}(\mathrm{Live}(T))) = \emptyset,$$

(TR-GAR)(e″)

$$\mathrm{VarS}(s^{-1}(\mathrm{Live}(S))) \cap \mathrm{VarS}(s^{-1}(\mathrm{Gar}(S))) = \mathrm{VarS}(t^{-1}(\mathrm{Live}(T))) \cap \mathrm{VarS}(t^{-1}(\mathrm{Gar}(T))) = \emptyset.$$

All of the properties we have written down are composable and associative in the expected manner. Therefore, we can emulate the treatment of the previous section and build a large range of derived categories describing our version of term rewriting at different levels of abstraction.

Instead of doing this, however, we set off in a different direction and exploit the additional topological properties of tree-structured systems. This will ultimately bring us to a fairly conventional picture of term rewriting.

9.3. Towards conventional term rewriting

Let us fix one or other notion of validity. Let us fix all fibres to be discrete (and drop the suffix id on *TRew* when naming Grothendieck categories), and let us consider the category $\mathscr{MT}(\mathscr{MO}, TRew)_{\text{ROOT}}$ consisting of valid rewrites. There is a projection functor $\mathscr{M}F: \mathscr{MT}(\mathscr{MO}, TRew)_{\text{ROOT}} \to \mathscr{MO}$, although it is not an opfibration. Consider now the subcategory $\mathscr{MT}(\mathscr{MO}, TRew)^{1}_{\text{ROOT}}$ whose objects are those objects of $\mathscr{MT}(\mathscr{MO}, TRew)_{\text{ROOT}}$ which satisfy

(TR-ONE) $\langle s: L \to \langle S, \text{Gar}(S)\rangle\rangle$ is an object of $\mathscr{MT}(\mathscr{MO}, TRew)^{1}_{\text{ROOT}}$ iff precisely one open term of L is mapped by s into Live(S).

and whose arrows are all arrows of $\mathscr{MT}(\mathscr{MO}, TRew)_{\text{ROOT}}$ with source and target within $\mathscr{MT}(\mathscr{MO}, TRew)^{1}_{\text{ROOT}}$. It is easy to see that this is a subcategory.

When projected down by $\mathscr{M}F$ to $\mathscr{MO}$, the images of the arrows of $\mathscr{MT}(\mathscr{MO}, TRew)^{1}_{\text{ROOT}}$ are arrows $(i, r): L \to R$ such that for at most one term l in L is it true that $r(root(l)) \neq i(root(l))$. Such arrows of $\mathscr{MO}$ do not form a subcategory. Nevertheless, the projection $\mathscr{M}F: \mathscr{MT}(\mathscr{MO}, TRew)^{1}_{\text{ROOT}} \to \mathscr{MO}$ enjoys the desirable properties of a split opfibration, i.e. rewrites are opcartesian for their source and image in $\mathscr{MO}$, and they compose like a splitting, as the reader can check easily enough.

Note how unlike the general graph rewriting case this is. All of this good behaviour is entirely attributable to the strong topological properties of terms under the simple invariant (TR-ONE).

Whenever a functor $P: \mathscr{X} \to \mathscr{B}$ is such that, for every $x \in \mathscr{X}$ and arrow $f: x \to y \in \mathscr{X}$, (1) there is an assigned opcartesian arrow $\bar{f}: x \to \bar{y}$ with $P(f) = P(\bar{f})$, (2) these arrows are consistently assigned in that $P(f: x \to y) = P(f': x \to y') \Rightarrow \bar{f} = \bar{f}'$, (3) they satisfy $P(\text{Arr}(y, -)) = P(\text{Arr}(\bar{y}, -))$, where $\text{Arr}(y, -)$ is the set of arrows with domain y, and (4) they form a splitting, we have a weak opfibration (see [3]). We can then construct a category $\mathscr{M}$, a functor $F_{\mathscr{M}}: \mathscr{M} \to \mathscr{C}at$ such that $\mathscr{X} \approx G(\mathscr{M}, F_{\mathscr{M}})$, the category produced by the Grothendieck construction from $\mathscr{M}$ and $F_{\mathscr{M}}$, and a projection $K: \mathscr{M} \to \mathscr{B}$, such that P is the composition of a split opfibration $P_{\mathscr{M}}: \mathscr{X} \to \mathscr{M}$ with K. When the category $\mathscr{X}$ arises as a suitable preimage under some functor $H: \mathscr{X} \to \mathscr{Y}$, of a Grothendieck category $Y = G(\mathscr{B}, F_{\mathscr{B}})$ over $\mathscr{B}$, we call the category $\mathscr{M}$ the split opfibration complement of the functor $H: \mathscr{X} \to \mathscr{Y}$ and the split opfibration $P_{\mathscr{B}}: \mathscr{Y} \to \mathscr{B}$, and we have a morphism of split opfibrations $\langle P_{\mathscr{B}}: \mathscr{Y} \to \mathscr{B}\rangle \to \langle P_{\mathscr{M}}: \mathscr{X} \to \mathscr{M}\rangle$ given by H and K.

Readers can check that the projection $\mathcal{M}F:\mathcal{M}\mathcal{T}(\mathcal{M}\mathcal{O}, TRew)^1_{\text{ROOT}}\to\mathcal{M}\mathcal{O}$ satisfies the required criteria. The split opfibration complement whose existence is asserted above is isomorphic to the category $\mathcal{M}\mathcal{O}^1_{\text{ROOT}}$. The objects of $\mathcal{M}\mathcal{O}^1_{\text{ROOT}}$ are pointed multisets (i.e. multisets with a distinguished element) of open terms, and their arrows are $(i, r):L^*\to R^*$, such that, for all nondistinguished terms l of L^*, $i(root(l))=r(root(l))$, and r maps the distinguished term of L^* to the distinguished term of R^*. Arrows δ and ε are composable, giving $\varepsilon\circ\delta$ iff the distinguished term of the target of δ is the distinguished term of the source of ε.

On the basis of the preceding considerations, it is easy to see that the projection

$$\mathcal{M}F^1_{\text{ROOT}}:\mathcal{M}\mathcal{T}(\mathcal{M}\mathcal{O}, TRew)^1_{\text{ROOT}}\to\mathcal{M}\mathcal{O}^1_{\text{ROOT}}$$

that takes $\langle s:L\to\langle S, \text{Gar}(S)\rangle\rangle$ to L^* (where L^* is just L with, as the distinguished term, that term of L which is mapped by s into Live(S)), and which takes an arrow

$$\langle(i, r):L\to R\rangle:\langle s:L\to\langle S, \text{Gar}(S)\rangle\rangle\to\langle t:R\to\langle T, \text{Gar}(T)\rangle\rangle$$

to the corresponding $(i, r):L^*\to R^*$, is an opfibration. It is clear that the objects of $\mathcal{M}\mathcal{O}^1_{\text{ROOT}}$ are prelabelled versions of objects in $\mathcal{M}\mathcal{O}$ (the prelabelling being the classification of terms as (preimages of) live or garbage terms for rewriting) and thus that there is an obvious projection $P:\mathcal{M}\mathcal{O}^1_{\text{ROOT}}\to\mathcal{M}\mathcal{O}$, which just forgets which of the terms of an object is distinguished. Given the simple topological structure of objects of $\mathcal{M}\mathcal{O}$ and of fibres $\mathcal{M}\mathcal{T}^L$, this prelabelling is a sensible thing to do.

9.4. *Conventional term rewriting*

Consider conventional term rewriting, which is always done without any mention of garbage. The base category for this is $\mathcal{O}$, whose objects are individual open terms L, and whose arrows are $L\to R$ where $\text{VarS}(R)\subseteq\text{VarS}(L)$. Above each L in $\mathcal{O}$, place a fibre $\mathcal{T}^L$, whose objects are pairs $\langle S, s\rangle$, with S an individual closed term and $s:L\to S$ a matching of L into S, and whose arrows are just identities. If $\delta\equiv L\to R$ is an arrow of $\mathcal{O}$, let $TR^\delta:\mathcal{T}^L\to\mathcal{T}^R$ be the functor which takes a redex $s:L\to S$ to $t:R\to T$, where T is the result of applying the term rewrite rule δ to $s:L\to S$ (i.e. replace the instance of L in S by an instance of R with variables suitably instantiated via s). Predictably, the functors TR^δ glue together to give a functor $TR:\mathcal{O}\to\mathcal{C}at$ with $TR(L)=\mathcal{T}^L$ and $TR(\delta:L\to R)=TR^\delta:\mathcal{T}^L\to\mathcal{T}^R$. Conventional term rewriting thus consists of the arrows of the Grothendieck category $\mathcal{T}(\mathcal{O}, TR)$, with split opfibration $F:\mathcal{T}(\mathcal{O}, TR)\to\mathcal{O}$.

9.5. *Relating the different styles of term rewriting*

The pièce de résistance of this section is the morphism of split opfibrations

$$\text{Live}:\langle\mathcal{M}F^1_{\text{ROOT}}:\mathcal{M}\mathcal{T}(\mathcal{M}\mathcal{O}, TRew)^1_{\text{ROOT}}\to\mathcal{M}\mathcal{O}^1_{\text{ROOT}}\rangle\to\langle F:\mathcal{T}(\mathcal{O}, TR)\to\mathcal{O}\rangle.$$

This consists of two suitably related functors: Live$G$, which acts on the Grothendieck categories, and Live$B$, which acts on the bases. Live$B:\mathcal{M}\mathcal{O}^1_{\text{ROOT}}\to\mathcal{O}$ takes each

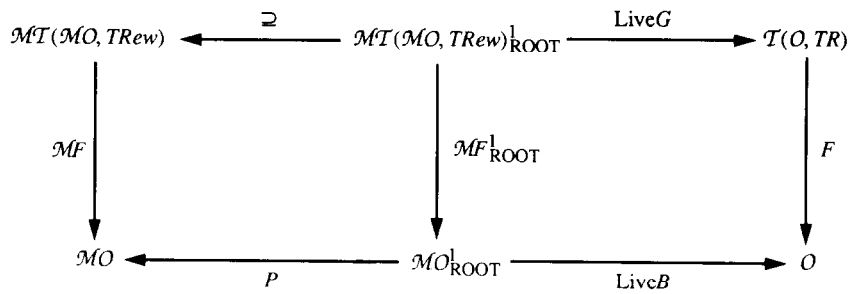


Fig. 21. Morphisms of split opfibrations.

pointed multiset L^* to its distinguished term L', and each rule $(i, r): L^* \to R^*$ to $L' \to R'$, where in the latter case no explicit function on roots is necessary.

The functor $\mathrm{Live}G: \mathcal{MT}(\mathcal{MO}, TRew)^{1}_{\mathrm{ROOT}} \to \mathcal{T}(\mathcal{O}, TR)$ takes each object $\langle s: L \to S \rangle$ to $\langle s': L' \to S' \rangle$, the instance of L' in the subterm rooted at the unique child of the unique Root-rooted term of S. Thus, $S' = \mathrm{Live}(S)$. On arrows Live$G$ takes

$$\langle \mathrm{id}, (i, r): L^* \to R^* \rangle : \langle s: L \to \langle S, \mathrm{Gar}(S) \rangle \rangle \to \langle t, R \to \langle T, \mathrm{Gar}(T) \rangle \rangle$$

to

$$\langle \mathrm{id}, L' \to R' \rangle : \langle s': L' \to S' \rangle \to \langle t': R' \to T' \rangle.$$

The situation is illustrated in Fig. 21, which also includes the original projection $\mathcal{M}F: \mathcal{MT}(\mathcal{MO}, TRew) \to \mathcal{MO}$.

9.6. *Discussion*

The above makes plausible the claim at the end of Section 8 that the special case of rooted rewriting indeed gives rise to an opfibration. We get a similar figure to Fig. 21, but with a different left-hand square. The two left-hand squares are related by functors constructible on the basis of results in works such as those by Barendregt et al. [4], Kennaway et al. [24] and Farmer and Watro [11].

At this point, we might reasonably ask whether an enterprise similar to Fig. 21 might not succeed for graph rewriting in general. On the whole, it will not, since the category of valid rewrites need not have all the necessary split opcartesian arrows. Nevertheless, there may well be circumstances other than those directly concerning terms, where the category of valid rewrites is a weak opfibration over the rules. This explains why graph rewriting in the DACTL style is best done with garbage included, while term rewriting is best done in a "garbage-free" way.

Finally, we hope readers will be amused by the structural similarity of the problem of finding the split opfibration complement in Fig. 21 and the problem of finding the pushout complement in algebraic graph rewriting. Both are connected with garbage, but describe different phenomena, and at vastly different levels of abstraction.

10. Conclusions

In the preceding sections, we have set up a number of notions of graph rewriting, starting with the purely operational DACTL model, moving through its locally universal generalisation, the D model, and finishing with the globally universal $\mathscr{P}$ and $\mathscr{G}$ models. In the $\mathscr{G}$ model, rewriting is described using an opfibration, the base consisting of patterns and rules, and the fibres being instances. The treatment of term rewriting from the same viewpoint was also instructive. The more categorical models show how garbage collection and real systems can be described in a natural manner, although some of the latter aspects of this exercise were admittedly a little taxonomic in nature.

It is gratifying to note that most of this was accomplished without real strain; the only place where we might be accused of having "fait du bricolage", as the French so charmingly put it, is in the construction of the D model, where some "handiwork" was needed to accommodate the full spectrum of DACTL redirection ambiguity.

Occupying an intermediate position between these extremes is the circular I rewrite. We showed that it can be described using the mechanics of the globally universal construction, but possesses only local universality, because of the different invariants involved. Other approaches to this rewrite have been proposed, and are discussed in [24] as well as in this paper. The author feels that the further clarification of universal properties possessed by the D model will illuminate this rewrite, and that the issue of good granularity may well play a part.

In rewriting theory, there are a large number of concepts that are categorical to the extent that they involve an associative law of composition, but it is not always clear how these fit together to form an overall description. The present treatment fares quite well in this respect as a consequence of the richness of the categorical data structures used in its constructions. To paraphrase MacLane, "there's lots of room in a Grothendieck construction".

Graph rewriting is a particularly good place to explore these issues. Unlike term rewriting, where the topological properties of the objects are so strong that many things are true "by accident", in graph rewriting nothing is true unless for a very good reason. As a consequence, while restricting truths that hold for graph rewriting to the term rewriting case is normally successful and shows how the simpler structure of terms strengthens such truths, the opposite approach, of trying to generalise arbitrary truths that hold for term rewriting to the graph rewriting case, normally ends in grief.

We leave the reader with the following thought. The various flavours of rewriting treated above can all be viewed as instances of some very general notion of update, or destructive assignment, or substitution. All operational models of computation possess such a component since they proceed (at some level of abstraction) by discrete stages that modify some computational structure. Therefore, the update or replacement feature of any model of computation can be modelled using a suitably universal construction. If the invariants of the model are such that properties of objects are

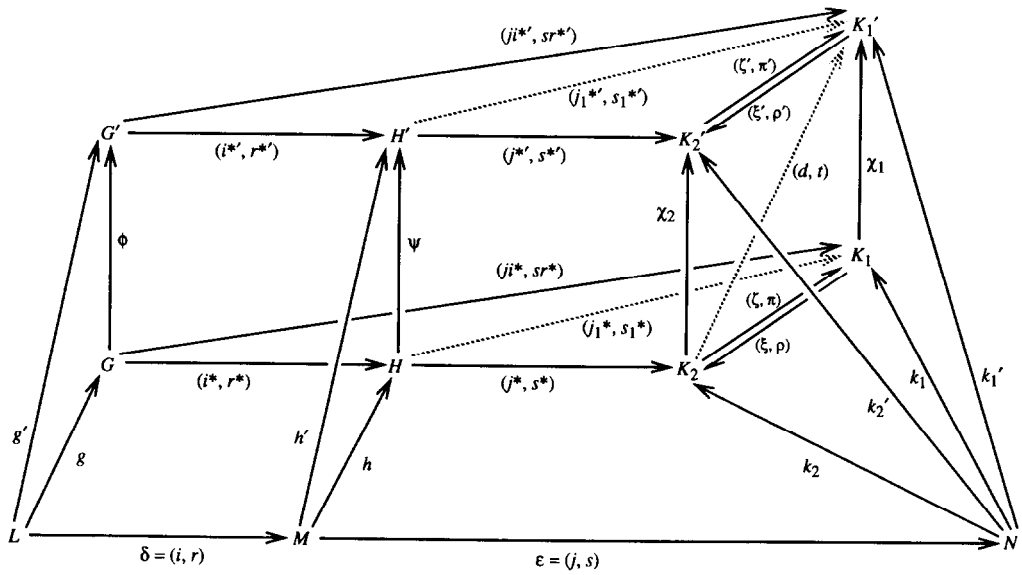


Fig. 22.

independent of properties of rules and of instances of rules, then we can expect to make a Grothendieck construction. The more nondeterministic a model of computation is, the more convincing the description via the opfibration is, in terms of having captured the essence of computation within the model. Conversely, the more deterministic the model, the less the opfibration captures, since more of the significant aspects of the model depend on strategy, which is a higher-level notion as far as our approach is concerned. One could make a manifesto of such a generalisation.

Appendix. Proof of Theorem 6.11

We give this result in fair detail because of its importance. We have to show that $Rew(\mathrm{id}_P) = \mathrm{Id}_{\mathscr{G}^P}$ and $Rew(\varepsilon \circ \delta) = Rew^{\varepsilon \circ \delta} = Rew^{\varepsilon} \circ Rew^{\delta} = Rew(\varepsilon) \circ Rew(\delta)$. The way to do this is to show that Fig. 22, in the world of concrete graphs, commutes. In particular, we need to show that K_1 and K_2 are isomorphic as graphs, and that χ_1 and χ_2 are isomorphic as arrows.

We start with K_1 and K_2. K_1 is obtained by rewriting in one step, while K_2 is obtained in two steps. To show that they are isomorphic it is enough to show that there are unique redirection couples $(\xi, \rho): K_1 \to K_2$ and $(\zeta, \pi): K_2 \to K_1$, since their compositions must then be identities.

Now K_1 is universal as a completion of the square $G \leftarrow L \rightarrow N$ via $(ji^*, sr^*): G \to K_1$ and $k_1 : N \to K_1$ because these are a rewrite. (Note that ji^* is a name of a function, not some sort of composition.) We claim that K_2 completes the same square via

$(j^* \circ i^*, s^* \circ r^*): H \to K_2$ and $k_2 : N \to K_2$. To see this, note that the squares $LGHM$ and MHK_2N can be juxtaposed to give a commuting square LGK_2N because the arrows of these squares are built out of functions, and such squares in $\mathscr{S}et$ can be juxtaposed. This easily gives us Theorem 6.6(1)–(4). We get Theorem 6.6(5) by noting that s^* satisfies it already because it is the redirection map of a rewrite, so we have our claim. Theorem 6.6 now gives us a unique redirection couple $(\xi, \rho): K_1 \to K_2$.

To get a unique couple $(\zeta, \pi): K_2 \to K_1$, we proceed in two stages. Firstly, we define $(j_1^*, s_1^*): H \to K_1$ as follows, and show that it is well defined:

$$j_1^*(i^*(x)) = ji^*(x),$$

$$j_1^*(h(m)) = k_1(j(m)),$$

$$s_1^*(r^*(x)) = sr^*(x),$$

$$s_1^*(h(m)) = k_1(s(m)),$$

$$s_1^*(i^*(x)) = j_1^*(i^*(x)) \quad \text{for } i^*(x) \in H - (r^*(G) \cup h(M)).$$

Secondly, we show that $(j_1^*, s_1^*): H \to K_1$ satisfies the hypotheses of Theorem 6.6, whence our desired unique redirection couple exists.

Here are the details. The first and fifth clauses are unambiguous since i^* is injective. The middle three might be ambiguous since h and r^* might not be injective. If there are distinct preimages under h or r^* of some node of H, then there are distinct l_1 and l_2 in L, the preimages of these preimages under i, g, respectively. Chasing l_1 and l_2 round the rewrite LGK_1N shows that the second and third clauses are unambiguous. For the fourth clause, we need the invariant (RED)(c) for unambiguity. Finally, to establish the well-definedness of (j_1^*, s_1^*), we examine the cases where the clauses overlap. These are the first and second, and also the third and fourth. It is a routine matter to chase a suitable l in L round the rewrite LGK_1N to establish consistency in both cases.

Now we must check that the hypotheses of Theorem 6.6 for $(j_1^*, s_1^*): H \to K_1$ and $k_1 : N \to K_1$ are satisfied. Condition (1) is obvious since all maps used in the definition of j_1^* are symbol/arity-preserving. Condition (2) follows by noting first that every arc of H is either the (i^*, r^*) image of an arc of G or the h image of an arc of M. In the former case, the (ji^*, sr^*) image of the G arc gives (2); in the latter case, the $(k_1 \circ j, k_1 \circ s)$ image of the M arc gives (2). Agreement on the overlap follows as usual by noting that, for arcs in the overlap, there is an L arc which we can chase round the rewrite LGK_1N to give consistency. Condition (3) holds by assumption and condition (4) holds by construction (see the second and fourth clauses for (j_1^*, s_1^*)). To check condition (5), suppose $s(p) = j(a)$, $s(q) = j(b)$ and $h(a) = h(b)$. We must show that $s_1^*(h(p)) = s_1^*(h(q))$. Since $h(a) = h(b)$ there must be l_1 and l_2 in L such that $a = i(l_1)$, $b = i(l_2)$ and $g(l_1) = g(l_2)$. Since $g(l_1) = g(l_2)$, $sr^*(g(l_1)) = sr^*(g(l_2))$. Now $s_1^*(h(p)) = k_1(s(p)) = k_1(j(a)) = k_1(j \circ i(l_1)) = sr^*(g(l_1)) = sr^*(g(l_2)) = k_1(j \circ i(l_2)) = k_1(j(b)) = k_1(s(q)) = s_1^*(h(q))$, giving (5), and completing the proof for objects.

We now turn to the arrows. Our discussion of objects showed that the lower plane of Fig. 22 commutes. Taking the obvious interpretation of the primed objects in the upper plane, we assume that it commutes too. Given the rigid redex morphism $\phi: G \to G'$, let $\chi_1: K_1 \to K_1'$ be obtained in one step, while $\chi_2: K_2 \to K_2'$ is obtained in two steps, via $\psi: H \to H_1'$. We must show that χ_1 and χ_2 are isomorphic as rigid redex morphisms, i.e. that $\chi_2 = \xi' \circ \chi_1 \circ \zeta$, which is enough since ξ' and ζ are isomorphisms of objects. We do this by showing that there is a unique $(d, t): K_2 \to K_1'$. For, given such a (d, t), the commutativity of Fig. 22, which we will have proved, allows us to write

$$(\zeta' \circ \chi_2, \pi' \circ \chi_2) = (d, t) = (\chi_1 \circ \zeta, \chi_1 \circ \pi).$$

Noting that $\pi = \zeta$ and $\pi' = \zeta'$ we find $d = t$, whence $\chi_2 = \xi' \circ \chi_1 \circ \zeta$ immediately.

It remains to show that (d, t) exists as claimed. This is not too difficult given the foregoing. K_2 is universal among completions of $H \leftarrow M \to N$ as noted previously since it is a rewrite. We claim that $(j_1^* \circ \psi, s_1^* \circ \psi): H \to K_1'$ and $k_1': N \to K_1'$ complete the same square in a manner that allows the use of Theorem 6.6. To check this we must establish the hypotheses of Theorem 6.6(1)–(5) again. Conditions (1)–(3) are obvious, and condition (4) is clear once we note that $\psi \circ h = h'$ and that $MH'K_1'N$ is a rewrite. We are left with (5), which amounts to showing that, if $s(p) = j(a)$, $s(q) = j(b)$ and $h(a) = h(b)$, then $s_1^{*\prime} \circ \psi(h(p)) = s_1^{*\prime} \circ \psi(h(q))$. But writing this equality in another way as $s_1^{*\prime}(h'(p)) = s_1^{*\prime}(h'(q))$, we spot the upper plane analogue of condition (5) for s_1^* which we showed above. So Theorem 6.6 gives us our unique $(d, t): K_2 \to K_1'$ and we are done. □

Acknowledgment

It is a pleasure to thank an anonymous referee, who suggested a number of improvements to the paper.

References

[1] A. Asperti and S. Martini, Categorical models of polymorphism, *Inform. and Comput.* **99** (1992) 1–79.

[2] R.H. Banach, Term graph rewriting and garbage collection à la Grothendieck, a previous draft of this paper, 1992.

[3] R.H. Banach, Weak fibrations, *J. Pure Appl. Algebra* **86** (1993) 7–22.

[4] 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, in: J.W. de Bakker and A.J. Nijman, eds., *Proc. PARLE-87*, Lecture Notes in Computer Science, Vol. 259 (Springer, Berlin, 1987) 141–158.

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

[6] J. Bénabou, Fibered categories and the foundations of naive category theory, *J. Symbolic Logic* **50** (1985) 10–37.

[7] P.M. van den Broek, Algebraic graph rewriting using a single pushout, in: S. Abramsky and T.S.E. Maibaum, eds., *Proc. CAAP-91*, Lecture Notes in Computer Science, Vol. 493 (Springer, Berlin, 1991) 90–102.

[8] T. Coquand, C. Gunter and G. Winskel, Domain theoretic models of polymorphism, *Inform. and Comput.* **81** (1989) 123–167.
[9] H. Ehrig, Introduction to the algebraic theory of graph grammars (a survey), Lecture Notes in Computer Science, Vol. 73 (Springer, Berlin, 1979) 1–69.
[10] H. Ehrig, Tutorial introduction to the algebraic approach of graph grammars, in: *Proc. 3rd Int. Workshop on Graph Grammars*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1986) 3–14.
[11] W.M. Farmer and R.J. Watro, Redex capturing in term graph rewriting, *Internat. J. Found. Comput. Sci.* **1** (1990) 369–386.
[12] J.R.W. Glauert, K. Hammond, J.R. Kennaway, G.A. Papadapoulos and M.R. Sleep, DACTL: some introductory papers, Report SYS-C88-08, School of Information Systems, Univ. of East Anglia, Norwich, UK, 1988.
[13] J.R.W. Glauert, J.R. Kennaway and M.R. Sleep, Final specification of DACTL, Report SYS-C88-11, School of Information Systems, Univ. of East Anglia, Norwich, UK, 1988.
[14] J.R.W. Glauert, J.R. Kennaway and M.R. Sleep, DACTL: an experimental graph rewriting language, 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, 1990) 378–395.
[15] J.W. Gray, Fibered and cofibered categories, in: S. Eilenberg et al., eds., *Proc. La Jolla Conf. on Categorical Algebra* (Springer, Berlin, 1966) 21–83.
[16] A. Grothendieck, Technique de Descente et Théoremés d'Existence en Géometrie Algébrique, I. Generalites, Descente par Morphismes Fidélement Plats, Paris, Seminaire Bourbaki 190, 1960.
[17] A. Grothendieck, Cathégories Fibrées et Descente, Seminaire de Géometrie Algébrique de l'Institut des Hautes Etudes Scientifiques Paris; also in Lecture Notes in Mathematics, Vol. 224 (Springer, Berlin, 1961) 145–194.
[18] 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) 92–112.
[19] B. Hoffman and D. Plump, Jungle evaluation for efficient term rewriting, in: *Proc. Internat. Workshop on Algebraic and Logic Programming*, Mathematical Research, Vol. 49 (Akademie-Verlag, Berlin, 1988).
[20] J.M.E. Hyland and A. Pitts, The theory of constructions: categorical semantics and Topos-theoretic models, in: J.W. Gray and A. Scedrov, eds., *Categories in Computer Science and Logic*, Contemporary Mathematics, Vol. 92 (Amer. Mathematical Soc., Providence, RI, 1989) 137–199.
[21] J.R. Kennaway, On "On graph rewritings", *Theoret. Comput. Sci.* **52** (1987) 37–58.
[22] J.R. Kennaway, Implementing term rewrite languages in DACTL, *Theoret. Comput. Sci.* **72** (1990) 225–250.
[23] J.R. Kennaway, Graph rewriting in some categories of partial morphisms, in: H. Ehrig, H. Kreowski and G. Rozenberg, eds., Lecture Notes in Computer Science, Vol. 532 (Springer, Berlin, 1991) 490–504.
[24] J.R. Kennaway, J.W. Klop, M.R. Sleep and F.-J. De Vries, Transfinite reductions in orthogonal term rewriting systems, CWI Report CS-9041, 1990; *Inform. and Comput.*, to appear.
[25] M. Lowe and H. Ehrig, Algebraic approach to graph transformation based on single pushout derivations, in: R.H. Möhring, ed., *Proc. Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, Vol. 484 (Springer, Berlin, 1991) 338–353.
[26] E. Moggi, A category-theoretic account of program modules, *Math. Struct. Comput. Sci.* **1** (1991) 103–139.
[27] F. Parisi-Presicce, H. Ehrig and U. Montanari, Graph rewriting with unification and composition, in: *Proc. 3rd Internat. Workshop on Graph Grammars*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1986) 496–514.
[28] J.C. Raoult, On graph rewritings, *Theoret. Comput. Sci.* **32** (1984) 1–24.
[29] J. Staples, Computation on graph-like expressions, *Theoret. Comput. Sci.* **10** (1980) 171–185.
[30] J. Staples, Optimal evaluations of graph-like expressions, *Theoret. Comput. Sci.* **10** (1980) 287–316.
[31] J. Staples, Speeding up subtree replacement systems, *Theoret. Comput. Sci.* **11** (1980) 39–47.
[32] C.P. Wadsworth, Semantics and pragmatics of the lambda calculus, Ph.D. Thesis, Oxford Univ., 1971.