Event structures and orthogonal term graph rewriting

J.R. Kennaway, J.W. Klop, M.R. Sleep, F.-J. de Vries

Computer Science/Department of Software Technology

Report CS-R9347 July 1993

Event Structures
and
Orthogonal Term Graph Rewriting

J.R. Kennaway [1], J.W. Klop [2], M.R. Sleep [3] & F.J. DE Vries [4]

(1) jrk@sys.uea.ac.uk, (2) jwk@cwi.nl, (3) mrs@sys.uea.ac.uk, (4) ferjan@cwi.nl

(1,3) *School of Information Systems, University of East Anglia,*
Norwich, NR4 7TJ, UK
(2,4) *CWI,*
P.O. Box 4079, 1009AB Amsterdam, The Netherlands

Abstract

For every normalisable term in an orthogonal term graph rewrite system, we construct an elementary event structure. Its events are the essentially different redexes which must be reduced to reach the normal form of the term. The associated state domain is the set of Lévy equivalence classes of needed reduction sequences starting from the term. Various properties of the event structure and state domain are related to properties of the term. The problems arising in extending this work to term rewrite systems and non-orthogonal rewrite systems are also discussed.

1991Mathematics Subject Classification: 68Q42

1991 CR Categories: F4.1, F4.2.

Keywords and Phrases: Concurrency, event structure, orthogonal term rewriting, graph rewriting, functional programming.

Note: All authors were partially sponsored by SEMAGRAPH, ESPRIT Basic Research Action 3074. The first author was also partially supported by a SERC Advanced Fellowship, and by SERC grant no. GR/F 91582. An earlier abstract was presented at the SemaGraph workshop, Nijmegen, December 1991.

Contents

Report CS-R9347
ISSN 0169-118X
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

1. Introduction

Several authors [Hue91, Sta89, and others] have hinted at a connection between transition systems such as are used to describe concurrency, and the reduction sequences that arise in term rewriting and lambda calculus. Here we make such a connection precise in the context of term graph rewriting. We construct for every normalisable term graph in an orthogonal term graph rewrite system, an elementary event structure. The events of this structure correspond to the different possible reduction steps that are required to reduce the term graph to normal form. The elements of the associated domain correspond to the possible needed reduction sequences which begin from the given term graph.

Similar connections have been remarked on for orthogonal term rewriting and lambda calculus, but in those contexts, the possibility of one reduction duplicating another redex makes it more complicated to derive any sort of event structure, and the resulting events are less closely related to physical computations.

In proving the results of this paper, we found the category-theoretic definition of graph rewriting which we introduced in [Ken91a] very useful in avoiding irrelevant technicalities. In particular, it casts further light on the physical meaning of Lévys equivalence relation on reduction sequences, which definitions in terms of tiling diagrams or permutation of reduction steps fail to do. Concrete definitions such as those of [Sta80,Bar87] would make these proofs much more complicated, and restrict them to one particular form of graph, while the more abstract definition may have application to other categories of graphs.

Proofs of theorems, where omitted here, will be found in the appendix.

2. Term graph rewriting

By term graph rewriting we mean, informally, one of the usual methods of implementing rewrite rules such as appear in functional languages such as ML or Miranda. The essential feature is that when a rewrite rule is applied whose right hand side contains multiple occurrences of a free variable, the corresponding subterm of the expression being evaluated is not duplicated; instead multiple pointers are created to the original copy. The expression is therefore no longer a string or a tree, but a graph of a particular sort: a term graph. It is technically convenient to represent these as hypergraphs that is, graphs in which each edge may have any positive number of vertexes.

2.1 DEFINITION. Given a set Σ of function symbols, each having some arity (a nonnegative integer), a *term graph* over Σ consists of a tuple (N,E,*str*), where N and E are sets, and *str* (structure) is a function from E to $\Sigma \times N \times N^*$. If *str*(e) = (F,n,s), then the *vertexes* of e are n and the members of s. n is the *principal* vertex of e. (N,E,*str*) is subject to the following conditions:

(i) If *str*(e) = (F,n,s) then the arity of F is the length of s.

(ii) Distinct hyperedges have distinct principal vertexes.

A node is *empty* if it is not the principal vertex of some hyperedge. A graph is *closed* if it contains no empty nodes. We use empty nodes to represent free variables. While a separate alphabet of variable symbols is a convenient means of writing term graphs in textual form, it is only a notational device and not a part of the underlying model.

A *rooted* term graph is a graph together with one of its nodes. It is *garbage-free* if every node in the graph is accessible from the root. Accessibility is defined thus: n' is accessible from n if either n=n', or *str*(n) = (F,n",s), and n' is accessible from n" or from some member of s.

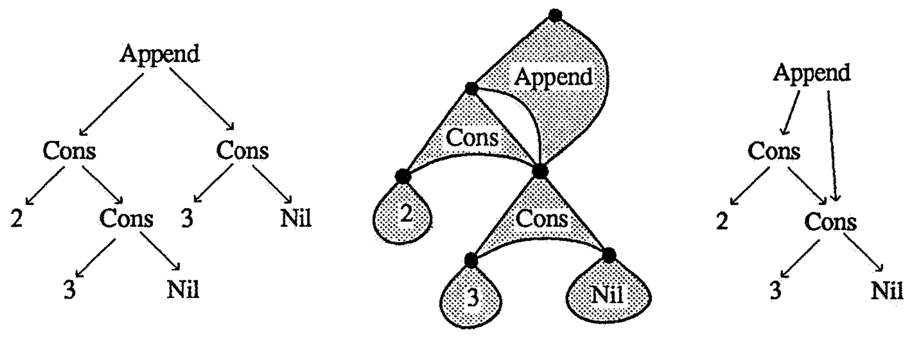
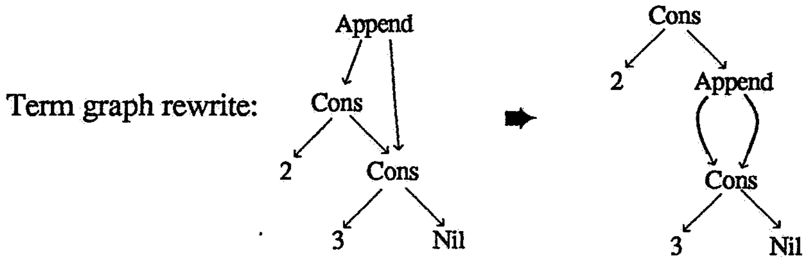


Figure 1: Terms and term graphs.

Figure 1 illustrates a term represented as a term graph. On the left is the term shown as a tree. In the middle is a visual representation of the hypergraph, where each of the shaded ares is a hyperedge. On the right is an equivalent representation, designed for its similarity to the tree picture. Instead of drawing the hyperedges explicitly, the function symbol of each hyperedge is attached to its principal vertex, from which arrows proceed to all the other vertexes of the hyperedge. In this example, the term graph has multiple references to the subgraph Cons(3,Nil) where the term has multiple occurences of that subterm. However, a hypergraph is allowed to contain distinct isomorphic subgraphs — we do not require "maximal sharing" as is done e.g. in [HofP88].



Term graph rewrite rule: Append(Cons(x, y), z) → Cons(x, Append(y, z))

Figure 2

Formal definitions of term graph rewriting have appeared in [Sta80, Bar87, Ken91a]. In this extended abstract we shall take the notion to be sufficiently intuitive to require no further explanation beyond Figures 1 and 2. However, the particular formalisation which we gave in [Ken91a] making use of category theory turns out to greatly simplify certain of the concepts and technical arguments which we shall later require, and we shall briefly describe this.

We do not require any advanced concepts of category theory, just the basic notions of category, functor, subobject, limit and colimit (in fact the only limits and colimits we need are pullbacks and pushouts). The hypergraphs we have defined form a category when a notion of homomorphism is given. This notion is the obvious one: a mapping of the nodes and edges of one graph to another which preserves function symbols and the connectivity of nodes and hyperedges. This category we call J (for *jungle*, a term coined in [HofP88]).

Morphisms preserve structure, but rewrites are intended to change the structure of a graph. We therefore represent rewrite rules not as morphisms of J, but as the morphisms of a derived category $\wp(J)$, the category of *partial morphisms* of J.

2.2 DEFINITION. In a category **C**, a *partial morphism* from A to B is a pair of morphisms A←X→B where X→A is a monomorphism. (We may indicate monomorphisms by $\hookrightarrow$ or $\hookleftarrow$.) More precisely, it is an equivalence class of such pairs. $A\hookleftarrow X\to B$ and $A\hookleftarrow Y\to B$ are equivalent if there is an isomorphism between X and Y making Figure 3 commute.

$$\begin{tikzcd} & X \arrow[dl, hook', "\supseteq"'] \arrow[dr] \arrow[dd, leftrightarrow, "\cong"] & \\ A & & B \\ & Y \arrow[ul, hook, "\supseteq"] \arrow[ur] & \end{tikzcd}$$

Figure 3: Equivalence of partial morphisms.

We write a partial morphism from A to B as A ⇸ B. It is *total* if $X\hookrightarrow A$ is an isomorphism. It is a *restriction* if X→B is an isomorphism.

Assume that **C** has the pullback of any pair of arrows of which at least one is a monomorphism. Then the composition of two partial morphisms $A \xleftarrow{\supseteq} X \rightarrow B$ and $B \xleftarrow{\supseteq} Y \rightarrow C$ is the partial morphism $A \xleftarrow{\supseteq} X \xleftarrow{\supseteq} Z \rightarrow Y \rightarrow C$ given by Figure 4, in which the square ZXYB is a pullback.

Composition of partial morphisms is associative, and the partial morphism $A \xleftarrow{id_A} A \xrightarrow{id_A} A$ is an identity for it. Thus the objects of **C** and partial morphisms form a category, which we denote by $\wp(\mathbf{C})$.

$$\begin{tikzcd}
Z \arrow[r, hook, "\subseteq"] \arrow[d] & X \arrow[r, hook, "\subseteq"] \arrow[d] & A \\
Y \arrow[r, hook, "\subseteq"] \arrow[d] & B & \\
C & &
\end{tikzcd}$$

Figure 4: Composition of partial morphisms.

2.3 DEFINITION. A *term graph rewrite rule* is a partial morphism $L \xleftarrow{\supseteq} X \rightarrow R$ of **J**, such that X is the subgraph of L obtained by omitting the root hyperedge (but retaining all the nodes), and such that every empty node of R is in the range of $X \rightarrow R$. A *redex* of this rule in a closed graph G is a total morphism from L to G. The hyperedges of G in the range of this morphism are *pattern-matched* by the redex. The *pre-reduct* of this redex is the graph H obtained as the pushout of $L \Rrightarrow R$ and $L \Rrightarrow G$. One may show that this graph always exists (although $\wp(\mathbf{J})$ does not have all pushouts).

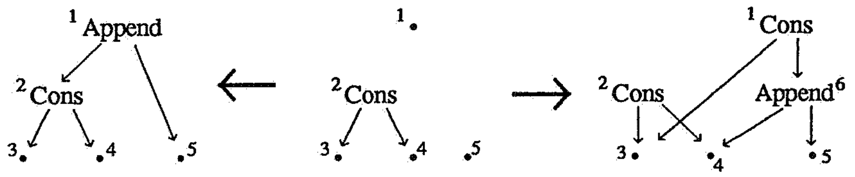


Figure 5: A term graph rewrite rule as a partial morphism

Figure 2 exhibited a term graph rewrite by the rule Append(Cons(x, y), z) $\rightarrow$ Cons(x, Append(y, z)). Figure 5 displays the formulation of this rule as a partial morphism of **J**. The attached numbers indicate the actions of the morphisms on nodes. Notice how empty nodes represent variables — we do not need a separate set of variable symbols.

Our definition of rewriting as a pushout of partial morphisms is not yet complete, hence the name pre-reduct. It omits the notion of garbage collection. To define this we must introduce the notion of a rooted graph.

2.4 DEFINITION. A *rooted* graph is a (total) morphism $\bullet \rightarrow G$, where $\bullet$ is the graph with one node and no edges. It is *garbage-free* if every node of G is accessible from the node which is the image of the morphism $\bullet \rightarrow G$. The result of *garbage-collecting* this rooted graph, GC($\bullet\!\to$G), is a garbage-free rooted graph $\bullet\!\to$G' such that there is a monomorphism G'$\to$G such that $\bullet\!\to$G'$\to$G = $\bullet\!\to$G, and such that G' is the largest subgraph of G for which this is so. (Category theorists may note that this amounts to an adjoint to the inclusion of the category of garbage-free rooted graphs in the category of rooted graphs, the latter being the comma category $\bullet\!\downarrow$J.)

Note that when G is closed, GC($\bullet\!\to$G) = $\bullet\!\to$G' where G' is the *unique* closed subgraph of G for which $\bullet\!\to$G' is garbage-free.

In J, $\bullet\!\to$G factors through G'. In $\wp$(J), we can also factor $\bullet\!\to$G' through $\bullet\!\to$G as $\bullet\!\to$G$\Rightarrow$G', where G$\Rightarrow$G' is the restriction morphism G$\leftarrow$G'$\cong$G'.

2.5 DEFINITION. Given a rooted graph $\bullet\!\to$G, the result of reducing a redex L $\Rightarrow$ G of a rule L $\Rightarrow$ R is depicted in Figure 6. The square LRGH is a pushout, performing a pre-reduction as above. $\bullet\!\to$G$\Rightarrow$H is in fact total, since the domain of G$\Rightarrow$H includes all the nodes of G. We can therefore apply garbage-collection to it and obtain a rooted graph $\bullet\!\to$H$\Rightarrow$H'. This is the *reduct* of the redex.

$$\begin{tikzcd}
 & L \arrow[r, Rightarrow] \arrow[d, Rightarrow] & R \arrow[d, Rightarrow] & \\
\bullet \arrow[r] & G \arrow[r, Rightarrow] & H \arrow[r, Rightarrow] & H'
\end{tikzcd}$$

Figure 6: Reduction step.

This defines a single reduction step. A reduction sequence can be constructed by stringing successive reductions together as in Figure 7. One important feature of this definition of rewriting is that it gives additional information besides the final graph: it also gives a partial morphism from the initial graph to the final graph which has a concrete and intuitive meaning. Let the morphism be $G_0$$\leftarrow$X$\to$$G_n$. Consider X as a subgraph of G_0. Then the nodes and hyperedges of G_0 outside X are those which the sequence erases. Hyperedges in X are preserved by the reduction. Nodes which are empty in X but nonempty in G are changed. Other nodes of X are preserved. The nodes and hyperedges of G_n outside the range of X$\to$$G_n$ are created by the reduction.

$$\begin{tikzcd}
 & L_0 \arrow[r, Rightarrow] \arrow[d, Rightarrow] & R_0 \arrow[d, Rightarrow] & L_0 \arrow[r, Rightarrow] \arrow[d, Rightarrow] & R_0 \arrow[d, Rightarrow] & & \\
\bullet \arrow[r] & G_0 \arrow[r, Rightarrow] & G'_0 \arrow[r, Rightarrow] & G_1 \arrow[r, Rightarrow] & G'_1 \arrow[r, Rightarrow] & G_2 \ \ldots \arrow[r, Rightarrow] & G_n
\end{tikzcd}$$

Figure 7: Reduction sequence

2.6 DEFINITION. Let there be given two distinct redexes $L_1 \Rightarrow G$ and $L_2 \Rightarrow G$ of rules $L_1 \Rightarrow R_1$ and $L_2 \Rightarrow R_2$. They are *disjoint* if there is no hyperedge of G which is erased by reduction of one redex but pattern-matched by the other.

A rule system is *orthogonal* if no graph contains non-disjoint redexes.

In the remainder of the paper we restrict attention to orthogonal systems.

3. Reduction graphs

Besides the graphs with whose rewriting we are concerned, we deal with another sort of graph.

A *reduction graph* of a term graph t is a rooted directed graph labelled as follows. Each node is labelled with a term graph. Each arc is labelled with a redex r of the term labelling its source, such that reduction of the redex yields the term labelling its target. Distinct out-arcs of a node bear distinct labels. The root of the graph is labelled with t, and all nodes are acessible from the root.

Note that different nodes may be labelled with the same term. We can consider several different reduction graphs of a term graph t. Firstly, there is the *reduction tree* of t, denoted RT(t). As its name implies, it is a tree. The out-arcs of each node are in 1-1 correspondence with the set of all the redexes of the term graph labelling that node. These two properties uniquely identify RT(t). Its nodes are in 1-1 correspondence with the set of finite reduction sequences starting from t.

The *minimal reduction graph* of t, MG(t), is obtained from RT(t) by identifying together all nodes bearing the same label, and identifying together their corresponding out-arcs. With each node of MG(t), labelled with t', we can associate the set of all paths in the graph from the root to that node. Each such path can be read as a reduction sequence from t to t'. Thus MG(t) can be represented by an equivalence relation on reduction sequences starting from t: s ≅ s' iff s and s' have the same final term.

A third reduction graph concerns us here: the *Lévy graph* of t, or LG(t). This stands mid-way between RT(t) and MG(t). Like MG(t), it identifies certain nodes of RT(t) together, but to a lesser extent, in general, than MG(t). It arises from an equivalence relation on reduction sequences finer than that associated with MG(t): the relation of *Lévy-equivalence.*

4. Lévy-equivalence

Consider the rewrite rule A(B) → C and the graph D(A(x:B),A(x)). The graph contains two redexes. There is an obvious sense in which we can reduce them both, in either order, and it is clear that the result is the same: D(C,C). See Figure 8. This notion is formalised as Lévy-equivalence. Lévy originally defined this for lambda calculus [Lév78, Lév80], but it applies to orthogonal rewriting in general [HueL91]. For term graph rewriting, it is rather simpler than for lambda calculus or term rewriting, and our categorical formulation of rewriting makes it trivial to define.

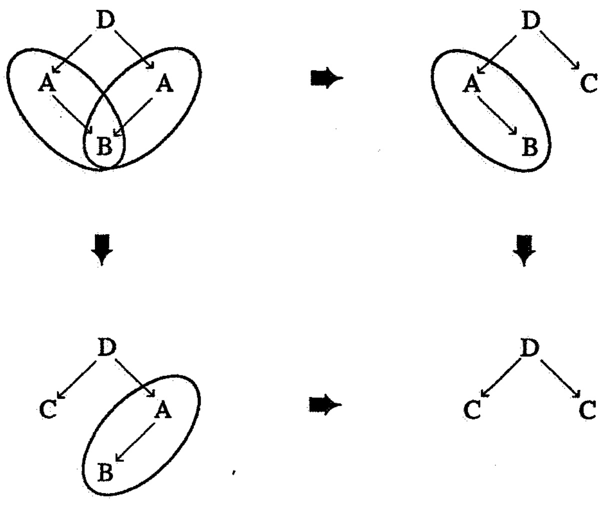


Figure 8.

4.1 DEFINITION. On finite reduction sequences, *Lévy equivalence* is the equivalence relation $\equiv_L$ generated by the following axioms:

(1) $r \cdot (r'/r) \equiv_L r' \cdot (r/r')$

(2) $s \equiv_L s' \Rightarrow s \cdot s'' \equiv_L s' \cdot s'' \wedge s'' \cdot s \equiv_L s'' \cdot s'$. □

4.2 THEOREM. [Lév78,Lév80] The above definition is equivalent to: $s \equiv s'$ iff s/s' and s'/s are both the empty sequence. □

4.3 THEOREM. Lévy-equivalent sequences determine the same partial morphism. □

Note that the converse does not hold. By the above theorem, Lévy-equivalent sequences do the same thing to each node and hyperedge of their common initial graph, but in addition, they also do the same thing to each node or hyperedge they create. Sequences determining the same partial morphism need not do the latter.

5. Event structures

We now define event structures. These were invented by Winskel [Nie81,Win80] to give a semantics for Petri nets.

There are several types of event structure. We will only require the simplest of them.

5.1 DEFINITION. An *elementary event structure* is a finite or countable set E and a partial ordering $\leq$ of E. $\leq$ is called the *causality relation.*

A *left closed* subset of E is a subset X such that $e \leq e' \wedge e' \in X \Rightarrow e \in X$.

Ev(E) denotes the set of left closed subsets of E, ordered by inclusion.

5.2 THEOREM . [Win80] L(E) is a prime algebraic complete lattice. □

The intuition behind these definitions is that E is the set of events that can happen in the course of a computation. $\leq$ is a relation of dependency or causality: when $e < e'$, then e' cannot happen unless e has already happened. The members of L(E) thus represent possible computational states: a state is the set of events which have happened so far.

6. Event structures for orthogonal term graph rewriting

The intuition underlying the following construction is that given a redex r of a graph G, and a reduction sequence $s:G \to G'$, if r/s is nonempty then it is in some sense the same piece of work as r, deferred to a later time.

6.1 DEFINITION. A *pre-event* of a term graph G is a pair (s,r), where $s:G \to H$ is a reduction sequence and r is a redex of H. Pre(G) is the set of all pre-events of G. For any reduction sequence s, the pre-events of s, denoted Pre(s), are the events (s',r) such that $s' \cdot r$ is an initial segment of s.

6.2 DEFINITION. Two pre-events are equivalent if they can be proved so by the following axioms:

(i) $(s,r) \cong (s',r)$ if $s \equiv_L s'$.

(ii) $(s,r) \cong (s \cdot s', r/s')$ if r/s' exists.

We now arrive at a theorem which is fundamental to the interpretation of term graphs as event structures.

6.3 THEOREM. No two distinct pre-events of a reduction sequence are equivalent. $\square$

Informally, this theorem means that it is not possible for a reduction sequence to do the same piece of work twice. This theorem also shows the distinction between term graph rewriting and term rewriting. *Mutatis mutandis*, it is false for the latter, because of the possibility that reduction of one redex can make multiple copies of another, which may all be later reduced. By the definitions we have given, the reductions of each of these copies, considered as pre-events, would be equivalent. Reducing more than one of them would give a reduction sequence containing two or more equivalent pre-events, defeating the construction of an event structure, in which an event is something which can only happen once.

6.4 DEFINITION. An *event* of G is an equivalence class of pre-events of G. Ev(G) is the set of events of G. Ev(s) is the set of events which are represented by the members of Pre(s).

The pre-events of a graph have an immediate computational interpretation as the possible steps which may be executed by a reduction machine evaluating the graph. The last theorem implies that the more abstract events may be interpreted in the same way. Furthermore, if we consider a machine capable of executing distinct redexes concurrently, without necessarily any definite total ordering of its reductions beyond that implied by causality, then events precisely correspond to the steps which may be made by such a machine.

Ev(s) does not quite describe "the work done by s", since it is possible for some steps of s to erase parts of the graph in which some previous steps were performed, making those steps unnecessary.

6.5 DEFINITION. An event e of a graph G is *needed* if $e \in Ev(s)$ for every reduction s of G to normal form. $Ev^0(G)$ is the set of needed events of G. A needed reduction sequence is one, all of whose events are needed. $LG^0(G)$ is the subgraph of LG(G) obtained by restricting to needed reduction steps.

$Ev^0(s)$ is the description we seek, as shown by the next theorem.

6.6 THEOREM. $s \equiv_L s'$ iff $Ev^0(s) = Ev^0(s')$. □

We now show that the notion of one redex creating another gives rise to a partial ordering of Ev(t).

6.7 DEFINITION. For e and e' in Ev(G), define $e \leq e'$ if for every reduction sequence s starting from G, if $e' \in Ev(s)$ then $e \in Ev(s)$. (It is immediate that this is indeed a partial order.) Define $e \prec e'$ if $e<e'$ and there is no e" such that $e<e''<e'$. □

The partial order has a concrete meaning.

6.8 DEFINITION. In a sequence s·r·s'·r', the pre-event (s,r) *contributes to* the pre-event (s·r·s',r') if there is a node n which is either the root of the contractum of r or is created by r, which is preserved by s', and such that n/s' is matched by r'.

(s,r) is *needed for* (s·r·s',r') if it contributes to (s·r·s',r') or if it contributes to a later step of s·r·s' which is needed for (s·r·s',r').

6.9 THEOREM. $e_0 < e_1$ iff there is a sequence of the form $s_0 \cdot r_0 \cdot s_1 \cdot r_1$ such that (s_0,r_0) is needed for $(s_0 \cdot r_0 \cdot s_1, r_1)$, and these two pre-events represent e_0 and e_1 respectively.

Finally, we have the required event structure and its associated configuration domain.

6.10 THEOREM. $Ev^0(G)$ with the partial ordering inherited from Ev(G) (of which it is a lower section) is an elementary event structure. Its associated domain of configurations is isomorphic to $LG^0(G)$. The resulting partial ordering of $LG^0(G)$ is identical to the ordering defined by Huet and Lévy [HueL91]: $s \leq s'$ iff s/s' is empty. □

Thus a Lévy-equivalence class of needed reductions starting from G is equivalent to a lower section of the set of needed events of G.

7. Related work and further developments.

In [Sta89], Stark defines a notion of 'concurrent transition system'. This takes as basic a notion of an abstract residual operation on abstract transitions. However, his primary concern is the study of process networks. His paper only mentions in passing the possibility of constructing event structures from concurrent transition systems, and that orthogonal term rewriting and lambda calculus reduction can give rise to examples of such structures. However, finding such structures in these contexts requires taking the basic transitions to be not ordinary reductions, but complete developments, which amounts to considering term graph reduction without the name. The construction — which is the purpose of the present paper — is still non-trivial.

We expect that the construction of event structures can also be applied in the presence of infinite graphs and transfinite rewriting, as set out in [Ken91b]. The set $Ev^0(t)$ is generalised to the set of *Böhm-needed* redexes of t — those redexes which must be reduced in any reduction of t which obtains every part of its Böhm tree (a concept borrowed from lambda-calculus).

To extend this work to non-orthogonal systems, we would have to deal with the possibility of conflicts among events. While event structures with a notion of conflict are well-known, all such structures in the literature depend on a conflict relation which is symmetric: if event e_1 conflicts with e_2, then e_2 conflicts with e_1. This is in general not the case for conflicts among redexes. Consider the rules $F(A) \rightarrow B$, $A \rightarrow C$. There is a conflict between these rules. The graph F(A) may be reduced either to B or to F(C). In this case, the conflict between the two redexes is symmetric. If one reduces either redex, the other no longer exists. However, consider the graph D(x:F(y:A),y). This again contains two conflicting redexes. Reducing the redex at y breaks the redex at x. But reducing the redex at x gives the graph D(x:B,y:A), in which the redex at y is still present. A type of event structure based on an asymmetric conflict relation is therefore required.

8. Conclusion.

For any normalisable term graph t in an orthogonal term graph rewrite system, the essentially different pieces of work which are required in the evaluation of t to normal form form an elementary event structure. The partial ordering embodies the relation of one redex contributing to another. Redexes can be reduced in any order compatible with the dependency relation.

The associated state domain is the set of Lévy-equivalence classes of needed reduction sequences starting from t. The top element corresponds to the reduction of t to normal form. The height of the partial ordering of $Ev^0(t)$ implies a lower bound on the time required to reach the normal form by reduction, and the width implies an upper bound on the amount of useful parallelism that can be employed.

Technical appendix

A.1. Lévy-equivalence and partial morphisms.

4.3 THEOREM. Lévy-equivalent sequences determine the same partial morphism.

PROOF. It is sufficient to establish this for the case where the two sequences have the form of those in the first axiom of Definition 2.2.1. That is, they are the two possible complete developments of a pair of redexes. The following lemmas do this.

A.1.1 LEMMA. In Figure 9, let the squares $L_1R_1GH_1$ and $L_2R_2GH_2$ be pre-reductions of disjoint redexes of G. Then H_1 and H_2 both pre-reduce to the same graph H, which is obtained as the pushout of $G \Rrightarrow H_1$ and $G \Rrightarrow H_2$. □

$$\begin{tikzcd}
 & L_1 \arrow[r, Rrightarrow] \arrow[d, Rrightarrow] & R_1 \arrow[d, Rrightarrow] \\
L_2 \arrow[r, Rrightarrow] \arrow[d, Rrightarrow] & G \arrow[r, Rrightarrow] \arrow[d, Rrightarrow] & H_1 \arrow[d, Rrightarrow] \\
R_2 \arrow[r, Rrightarrow] & H_2 \arrow[r, Rrightarrow] & H
\end{tikzcd}$$

Figure 9.

PROOF. This is immediate from the fact that if $L_1R_1GH_1$ and GH_1H_2H are pushouts, so is $L_1R_1H_2H$ (and similarly interchanging 1 and 2). Disjointness of the given redexes ensures that the partial morphisms $L_1 \Rrightarrow G \Rrightarrow H_2$ and $L_2 \Rrightarrow G \Rrightarrow H_1$ are also redexes. □

A.1.2 LEMMA. Let $\bullet \rightarrow G$ pre-reduce to $\bullet \rightarrow H$. Then $GC(\bullet \rightarrow G)$ reduces to $GC(\bullet \rightarrow H)$.

PROOF. It is enough to show this for a single step pre-reduction. Let $GC(\bullet \rightarrow G) = \bullet \rightarrow G \Rrightarrow G'$ and $GC(\bullet \rightarrow H) = \bullet \rightarrow G \Rrightarrow H'$. If the root of L is not in the domain of $L \Rrightarrow G \Rrightarrow G'$, then the node changed by the reduction of $L \Rrightarrow G$, and all nodes added by that reduction, are garbage in H. Thus G=H, and $GC(\bullet \rightarrow G)$ reduces to $GC(\bullet \rightarrow H)$ by the empty reduction.

Otherwise, take the pushout H" of $G \Rrightarrow H$ and $G \Rrightarrow G'$ (see Figure 10). H" is also the pushout of $L \Rrightarrow R$ and $L \Rrightarrow G \Rrightarrow G'$. Since $G \Rrightarrow G'$ is a restriction morphism, so is $H \Rrightarrow H''$. Therefore $H \Rrightarrow H'$ factors through $H \Rrightarrow H''$. Therefore $GC(\bullet \rightarrow G \Rrightarrow G' \Rrightarrow H'') = GC(\bullet \rightarrow G \Rrightarrow H)$. □

$$\begin{tikzcd}
 & L \arrow[r, Rightarrow] \arrow[d, Rightarrow] & R \arrow[d, Rightarrow] & \\
\bullet \arrow[r] & G \arrow[r, Rightarrow] \arrow[d, Rightarrow] & H \arrow[r, Rightarrow] \arrow[d, Rightarrow] & H' \\
 & G' \arrow[r, Rightarrow] & H'' \arrow[ur, Rightarrow] &
\end{tikzcd}$$

Figure 10.

A.1.3 LEMMA. In Figure 11, let G reduce to H_1 and H_2 by reduction of the redexes $r_1 = L_1 \Rightarrow G$ and $r_2 = L_2 \Rightarrow G$. Then H_1 and H_2 both reduce to the same graph H, which is obtained by garbage-collecting the pushout H' of $G \Rightarrow G_1$ and $G \Rightarrow G_2$.

PROOF. From the preceding lemmas. □

$$\begin{tikzcd}
\bullet \arrow[dr, Rightarrow] & L_1 \arrow[r, Rightarrow] \arrow[d, Rightarrow] & R_1 \arrow[d, Rightarrow] & \\
L_2 \arrow[r, Rightarrow] \arrow[d, Rightarrow] & G \arrow[r, Rightarrow] \arrow[d, Rightarrow] & G_1 \arrow[r, Rightarrow] \arrow[d, Rightarrow] & H_1 \\
R_2 \arrow[r, Rightarrow] & G_2 \arrow[r, Rightarrow] \arrow[d, Rightarrow] & G' \arrow[dr, Rightarrow] & \\
 & H_2 & & H
\end{tikzcd}$$

Figure 11.

This establishes the theorem. □

A.2. Non-equivalence of distinct steps of a sequence.

This section proves Theorem 6.3. We proceed by establishing properties of pre-events which are of minimal length in their equivalence class. Equivalent minimal pre-events are found to satisfy a much stronger equivalence relation. From this Theorem 6.3 will then follow.

A.2.1 DEFINITION. A pre-event (s,r) is *minimal* if there is no $(s',r') \equiv (s,r)$ with $|s'| < |s|$.

A pre-event (s,r) is *irredundant* if every pre-event of s contributes to a later pre-event of s·r. (Equivalently, if every pre-event of s is needed for the pre-event (s,r).) □

A.2.2 LEMMA. Every minimal pre-event is irredundant.

PROOF. Let (s,r) be a counterexample of minimal length. Let r_0 and r_1 be the first two steps of s·r. By minimality of the counterexample, r_0 does not contribute to any later step. Therefore r_0 does not create r_1. Let $r_1 = r_2/r_0$. If $r_0 = s$ and $r_1 = r$, then $(\langle\rangle, r_2) \equiv (r_0, r_1)$, contradicting minimality. Otherwise, $s = r_0 \cdot r_1 \cdot s' \equiv_L r_2 \cdot (r_0/r_1) \cdot s'$. If r_0/r_1 is empty, then $(r_2 \cdot s', r) \equiv (s,r)$ and $r_2 \cdot s'$ is shorter than s, contradicting minimality. Finally, if r_0/r_1 is nonempty, it does not contribute to any step of s'·r, and so $((r_0/r_1) \cdot s', r)$ is a shorter counterexample. □

We can elaborate the above proof into an algorithm for transforming any pre-event (s,r) into an equivalent irredundant pre-event.

A.2.3. THE MINIMISATION ALGORITHM. Firstly, note that if s is empty, (s,r) is irredundant.

For nonempty s, we will deal with each step of s, from the last backwards. At each stage, we will have transformed (s,r) into an equivalent pre-event $(s_0{\cdot}r_0{\cdot}s_1,r_1)$, where (s_1,r_1) is minimal. Initially, $s_0{\cdot}r_0 = s$ and s_1 is empty (making (s_1,r_1) irredundant).

If $(\langle\rangle,r_0)$ contributes to some later step of $r_0{\cdot}s_1{\cdot}r_1$, then $(r_0{\cdot}s_1,r_1)$ is irredundant. Otherwise, we need the following lemma.

A.2.4 LEMMA. If $(\langle\rangle,r)$ does not contribute to any later pre-event of a sequence r·s, then there is a sequence s' such that s = s'/r and $|s| = |s'|$.

PROOF. If s is empty this is trivial. Otherwise s = r'·s', where r does not create r'. Then r' = r"/r for some r", and in the sequence (r/r")·s', if r/r" is nonempty, it does not contribute to any later pre-event of the sequence. s' is shorter than s, so by induction we may assume that there is an s" such that s' = s"/(r/r") and $|s'| = |s''|$. Then s = (r"/r)·(s"/(r/r")) = (r"·s")/r and $|s| = |r''{\cdot}s''|$ □

Applying this lemma to the situation where $(\langle\rangle,r_0)$ does not contribute to any later step of $r_0{\cdot}s_1{\cdot}r_1$, we find that there is an $s_2{\cdot}r_2$ such that $s_1{\cdot}r_1 = (s_2{\cdot}r_2)/r_0$ and $|s_2{\cdot}r_2| = |s_1{\cdot}r_1|$. This implies that projection over r_0 does not erase any step of $s_2{\cdot}r_2$, and that in particular $r_1 = r_2/(r_0/s_2)$. Therefore $(r_0{\cdot}s_1,r_1) \equiv (s_2,r_2)$. Since (s_1,r_1) is irredundant, by Theorem A.3.2 (s_2,r_2) is also. Furthermore $(s_0{\cdot}r_0{\cdot}s_1,r_1) \equiv (s_0{\cdot}s_2,r_2)$.

By continuing in this way, we process each member of s, obtaining in the end an equivalent irredundant pre-event (s',r'). □

Notice that the above construction also provides us with a reduction sequence s" such that s'·r'·s"/s·r is empty. s" consists, roughly speaking, of the parts of s that were not needed for (s,r).

A.2.5 DEFINITION. The number of steps of s which are needed for (s,r) is the *needed length* of (s,r).

A.2.6 LEMMA. If the minimisation algorithm transforms (s,r) into (s',r'), then $|s'|$ is the needed length of (s,r).

PROOF. Clear from the construction. □

A.2.7 DEFINITION. Two pre-events (s,r) and (s',r') are *strongly equivalent* if $s \equiv_L s'$ and r = r'. □

A.2.8 LEMMA. If (s,r) $\cong$ (s',r') then the needed lengths of the two pre-events are equal. Furthermore, the respective results of applying the minimisation algorithm to both are strongly equivalent.

PROOF. It is sufficient to prove this when (s,r) and (s',r') are related by either of the axioms of Definition 6.2. Since the second part of the theorem implies the first, it is sufficient to prove only the second part.

For the second axiom, it is clear that the minimisation algorithm will produce identical results given either (s,r) or (s·s',r/s').

For the first axiom, it is sufficient to take the case where r = r' and s = $s_0 \cdot r_0 \cdot (r_1/r_0) \cdot s_1 \equiv_L s_0 \cdot r_1 \cdot (r_0/r_1) \cdot s_1$ = s'. For s to be distinct from s', at least one of r_1/r_0 and r_0/r_1 must be nonempty. Assume r_1/r_0 is nonempty.

Applying the minimisation algorithm, we may assume without loss of generality that (s_1,r) is minimal. We must show that the results of applying the algorithm on the one hand to r_1/r_0 and then r_0, and on the other hand to r_0/r_1 (if nonempty) and r_1, have the same length, and that this equality of length is preserved when we apply the algorithm to s_0. We shall prove these by induction on the length of s_1.

For the base case, let s_1 be empty. Then the results of minimising $r_0 \cdot (r_1/r_0)$ and $r_1 \cdot (r_0/r_1)$ will be either both r_0 (if r_0 erases r_1), both r_1 (if r_1 erases r_0), or the same two sequences (which are Lévy equivalent).

Let $s_1 = r_2 \cdot s_2$. If r_2 is not created by $r_0 \cdot (r_1/r_0)$, then there is an r_2' such that $r_2 = r_2'/(r_0 \cdot (r_1/r_0)) = r_2'/(r_1 \cdot (r_0/r_1))$. The minimisation algorithm applied to $r_0 \cdot (r_1/r_0) \cdot s_1$ will first move r_1/r_0 past r_2, and then into the rest of s_1, and then do the same with r_0. The result will be equal to the result of applying the algorithm to $(r_0/r_2') \cdot ((r_1/r_2')/(r_0/r_2')) \cdot s_2$, and prefixing r_2'. Minimising $r_1 \cdot (r_0/r_1) \cdot s_1$ will produce the same result as minimising $(r_1/r_2') \cdot ((r_0/r_2')/(r_1/r_2')) \cdot s_2$, and prefixing r_2'. Since s_2 is smaller than s_1, the result follows by induction.

If r_1/r_0 contributes to r_2 but r_0 does not, then the minimisation algorithm applied to $r_0 \cdot (r_1/r_0) \cdot s_1$ will leave r_1/r_0 where it is, and move r_0 past r_2, giving $r_1 \cdot r_2' \cdot (r_0/(r_1 \cdot r_2')) \cdot s_2$, where $r_2 = r_2'/(r_0/r_1)$. It will then move the residual of r_0 into s_2, but we do not need to follow its subsequent fate. Applied to $r_1 \cdot (r_0/r_1) \cdot s_1$, the algorithm will move r_0/r_1 past r_2 to give $r_1 \cdot r_2' \cdot (r_0/(r_1 \cdot r_2')) \cdot s_2$, as before.

If r_0 contributes to r_2 but r_1/r_0 does not, then r_0/r_1 contributes to r_2 and r_1 does not, which is equivalent to the preceding case.

If r_0 and r_1/r_0 both contribute to r_2, then so do r_1 and r_0/r_1. The results of minimising $r_0 \cdot (r_1/r_0) \cdot s_1$ and $r_1 \cdot (r_0/r_1) \cdot s_1$ are therefore the respective results of minimising $r_0 \cdot (r_1/r_0)$ and $r_1 \cdot (r_0/r_1)$, each suffixed by s_1, from which the theorem follows. □

A.2.9 LEMMA. (i) An irredundant pre-event is minimal.

(ii) Distinct redexes of the same term, considered as pre-events with empty history, are not equivalent.

PROOF. (i) If $(s,r) \cong (s',r')$ and (s,r) is irredundant, then from Lemmas A.2.6 and A.2.8, $|s{\cdot}r| \le |s'{\cdot}r'|$, i.e. (s,r) is minimal.

(ii) Distinct redexes are minimal pre-events, yet not strongly equivalent. Hence (ii) follows from Lemma A.2.8. □

A.2.10 LEMMA. If $(s_0,r_0) \cong (s_1,r_1)$ and both $(s_0,r_0)/s$ and $(s_1,r_1)/s$ exist, then they are equivalent.

PROOF. It is sufficient to show this for the two cases of the definition of equivalence of pre-events. Each of these in turn follows from the Cube Lemma; see Figure 12. □

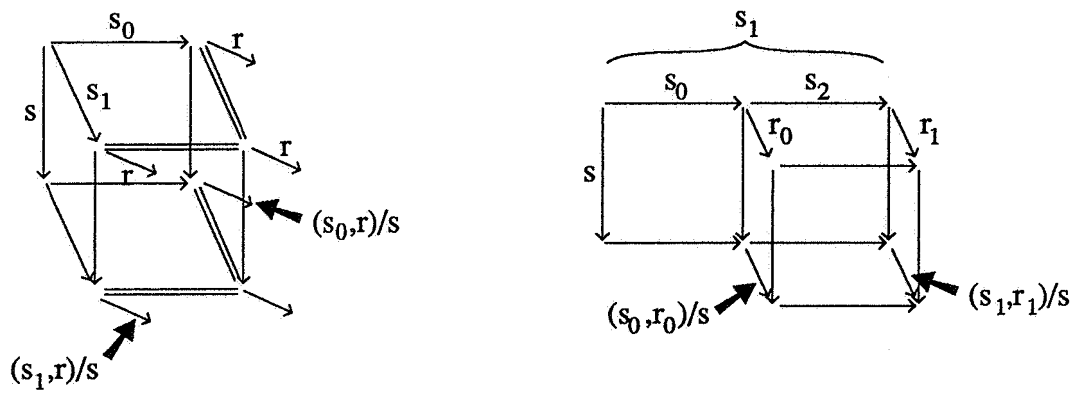


Figure 12

6.3 THEOREM. No two distinct pre-events of a reduction sequence are equivalent.

PROOF. $(s_0,r_0) \cong (s_0{\cdot}r_0{\cdot}s_1,r_1)$

$\Rightarrow$ $(s_0,r_0)/s_0 \cong (s_0{\cdot}r_0{\cdot}s_1,r_1)/s_0$ (Lemma A.2.10)

$\Rightarrow$ $(\langle\rangle,r_0) \cong (r_0{\cdot}s_1,r_1)$

Apply the minimisation algorithm to $(r_0{\cdot}s_1,r_1)$. This must yield a pre-event of the form $(\langle\rangle,r_2)$, such that $r_1 = r_2/(r_0{\cdot}s_1)$. But $(\langle\rangle,r_0)$ and $(\langle\rangle,r_2)$ must be strongly equivalent, hence $r_0 = r_2$, and $r_2/(r_0{\cdot}s_1)$ cannot exist. □

A.3. Lévy-equivalent reductions have the same needed events.

6.6 THEOREM. $s_1 \equiv_L s_2$ iff $Ev^0(s_1) = Ev^0(s_2)$. □

PROOF. The forwards implication is immediate from the definition of Ev^0.

For the converse, suppose $s_1 \not\equiv s_2$. Choose sequences s'_1 and s'_2 Lévy-equivalent to s_1 and s_2 respectively and of minimal length. At least one of s'_1/s'_2 and

s'_2/s'_1 must be nonempty. Supposing it is the first, consider the sequence $s'_2 \cdot (s'_1/s'_2)$. By Theorem 6.3, no step in the second segment can be equivalent to any step of s'_2. But every step of the second segment is equivalent to a step of s'_1. Therefore $Ev(s'_1) \neq Ev(s'_2)$. But $Ev^0(s_1) = Ev^0(s'_1) = Ev(s_1)$, and similarly for s_2, hence the theorem. □

A.4. The event structure of needed events.

6.9 THEOREM. $e_0 < e_1$ iff there is a sequence of the form $s_0 \cdot r_0 \cdot s_1 \cdot r_1$ such that (s_0,r_0) is needed for $(s_0 \cdot r_0 \cdot s_1,r_1)$, and these two pre-events represent e_0 and e_1 respectively.

PROOF. Let $e_0 < e_1$. By the definition of the ordering, there must be a sequence $s_0 \cdot r_0 \cdot s_1 \cdot r_1$ where (s_0,r_0) and $(s_0 \cdot r_0 \cdot s_1,r_1)$ represent e_0 and e_1 respectively. If (s_0,r_0) were not needed for $(s_0 \cdot r_0 \cdot s_1,r_1)$, then applying the minimisation algorithm to $(s_0 \cdot r_0 \cdot s_1,r_1)$ would begin by transforming it to a form $(s_0 \cdot r_0 \cdot s'_1,r'_1)$ with (s'_1,r'_1) a minimal pre-event, and then eliminate r_0. The final result would be a pre-event (s_2,r_2) equivalent to $(s_0 \cdot r_0 \cdot s_1,r_1)$, and in which $Ev(s_2)$ is a subset of $Ev(s_0 \cdot r_0 \cdot s_1)$ not containing e_0 (since by Theorem 6.3, no other pre-event of $s_0 \cdot r_0 \cdot s_1$ can be equivalent to (s_0,r_0)). But this contradicts $e_0 < e_1$.

For the converse, suppose we have a sequence $s_0 \cdot r_0 \cdot s_1 \cdot r_1$ of the stated form. To establish the ordering of the events, we must show that for any pre-event (s_2,r_2) equivalent to $(s_0 \cdot r_0 \cdot s_1,r_1)$, s_2 must contain a pre-event equivalent to (s_0,r_0). It is sufficient to do this for the cases where (s_2,r_2) is related to $(s_0 \cdot r_0 \cdot s_1,r_1)$ by one of the axioms for equivalence.

Axiom (i): $s_2 \equiv_L s_0 \cdot r_0 \cdot s_1$. We may assume that s_2 and $s_0 \cdot r_0 \cdot s_1$ are related by an application of part (1) of definition 2.2.1 to a part of $s_0 \cdot r_0 \cdot s_1$. If this part does not include r_0, then s_2 will have a pre-event (s'_0,r_0) equivalent to (s_0,r_0). Otherwise, there are two cases.

(a) There exist s_3 and $r_3 \neq r_0$ such that $s_2 = s_0 \cdot r_3 \cdot (r_0/r_3) \cdot s_3$, $s_1 = (r_0/r_3) \cdot s_3$, and $r_2 =$. If r_0/r_3 is nonempty, then $(s_0 \cdot r_3,(r_0/r_3)) \equiv (s_0,r_0)$. Otherwise, r_3 erases r_0. But this implies that r_3/r_0 erases every node which r_0 changes or creates. Therefore r_0 cannot contribute to any step of $s_1 \cdot r_1$ later than r_3/r_0. It cannot contribute to r_3/r_0 either, since this is a residual of a redex existing before r_0. This contradicts the hypothesis that (s_0,r_0) is needed for $(s_0 \cdot r_0 \cdot s_1,r_1)$.

(b) There exist s_3, r_3 and r_4 such that $s_0 = s_3 \cdot r_3$, $r_0 = r_4/r_3$, and $s_2 = s_3 \cdot r_4 \cdot (r_3/r_4) \cdot s_1$. Then $(s_3,r_4) \equiv (s_0,r_0)$.

Axiom (ii): There are three subcases.

(a) There exists r_3 such that $s_2 = s_0 \cdot r_0 \cdot s_1 \cdot r_3$ and $r_2 = r_1/r_3$. Then s_2 contains the pre-event (s_0,r_0).

(b) There exist s_3 and r_3 such that $s_1 = s_3 \cdot r_3$, $r_1 = r_2/r_3$, and $s_2 = s_0 \cdot r_0 \cdot s_3$. Again, s_2 contains the pre-event (s_0, r_0).

(c) s_1 is empty, $s_2 = s_0$, and there exists r_2 such that $r_1 = r_2/r_0$. But this implies that $(s_0, r_0) \equiv (s_0 \cdot r_0 \cdot s_1, r_1)$, contradicting Theorem 6.3.

6.10 THEOREM. $Ev^0(G)$ with the partial ordering inherited from Ev(G) (of which it is a lower section) is an elementary event structure. Its associated domain of configurations is isomorphic to $LG^0(G)$. The resulting partial ordering of $LG^0(G)$ is identical to the ordering defined by Huet and Lévy [HueL91]: $s \leq s'$ iff s/s' is empty.

PROOF. It is immediate that $Ev^0(G)$ is a lower section of Ev(G). Nodes of $LG^0(G)$ are in 1–1-correspondence with Lévy-equivalence classes of needed reduction sequences, which by Theorem 6.6 are in 1–1-correspondence with $Ev^0(G)$.

The configuration domain of $Ev^0(G)$ is the set of lower sections, ordered by the subset relation. Let s and s' be needed reduction sequences. If s/s' is empty, then $s' \equiv_L s \cdot (s'/s)$, therefore $Ev^0(s) \subseteq Ev^0(s')$. Conversely, suppose $Ev^0(s) \subseteq Ev^0(s')$. In the sequence $s' \cdot (s/s')$, every step in the s/s' segment is equivalent to some step of s, hence by hypothesis to some step of s'. But by Theorem 6.3 there can be no such step. Therefore s/s' is empty, and $s \leq s'$. □

References

[Bar87] Barendregt, H.P., van Eekelen, M.C.J.D., Glauert, J.R.W., Kennaway, J.R., Plasmeijer, M.J., and Sleep, M.R., Term graph rewriting, Proc. PARLE'87 Conference II, LNCS **259**, 141-158, (Springer-Verlag, 1987).

[HofP88] Hoffmann, B. and Plump, D., Jungle evaluation for efficient term rewriting, Proc. Int. Workshop on Algebraic and Logic Programming, eds. J. Grabowski et al, *Mathematical Research*, **49**, 191-203, (Akademie-Verlag, 1988).

[Hue91] Huet, G. and Lévy, J.-J., Computations in orthogonal rewriting systems: I and II, in J.-L. Lassez and G.D. Plotkin, eds., *Computational Logic: Essays in Honor of Alan Robinson*, 394–443 (MIT Press, 1991).

[Ken91a] Kennaway, J.R., Graph rewriting in some categories of partial morphisms, in *Proc. Int. Workshop on Graph Grammars and their Application to Computer Science*, Bremen, LNCS **532**, 490–504, (Springer-Verlag, 1991).

[Ken91b] Kennaway, J.R., Klop, J.W., Sleep, M.R., and de Vries, F.J., Transfinite reductions in orthogonal term rewriting systems, *Proc. Conference on Rewriting Techniques and Applications*, LNCS **488**, 1–12, (Springer-Verlag, 1991).

[Lév78] Lévy, J.-J., *Reductions correctes et optimales dans le lambda-calcul*, Thèse de doctorat d'état, Université Paris VII, 1978.

[Lév80] Lévy, J.-J., Optimal reductions in the lambda calculus, pp.159-191 in *To H.B. Curry: essays on combinatory logic, lambda calculus and formalism*, eds. J.P. Seldin, and J.R. Hindley, (1980).

[Nie81] Nielsen, M., Plotkin, G.D., and Winskel, G., Petri nets, event structures, and domains, Part 1, *Th. Comp. Sci.*, **13**, 1981.

[Pey87] Peyton Jones, S.L., *The Implementation of Functional Languages*, (Prentice-Hall, 1987).

[Sta80] Staples, J., Computation on graph-like expressions, *Th. Comp. Sci.*, **10**, 171-185, (1980).

[Sta89] Stark, E.W., Concurrent transition systems, *Th. Comp. Sci.*, **64**, 221-270, (1989).

[Win80] Winskel, G., *Events in Computation*, D.Phil thesis, Dept. of Computer Science, University of Edinburgh, (1980).