Math. Struct. in Comp. Science (2002), *vol.* 12, *pp.* 423–448. © 2002 Cambridge University Press
DOI: 10.1017/S096012950100353X Printed in the United Kingdom

Algebraic properties of processes for Local Action Systems[†‡]

NICO VERLINDEN and DIRK JANSSENS

Department of Mathematics and Computer Science, U.I.A.
Universiteitsplein 1, B-2610 Antwerp, Belgium
Email: {Nico.Verlinden,Dirk.Janssens}@ua.ac.be

Received 19 July 2000; revised 15 December 2000

Graph rewriting has been used extensively to model the behaviour of concurrent systems and to provide a formal semantics for them. In this paper, we investigate processes for Local Action Systems (LAS); LAS generalize several types of graph rewriting based on node replacement and embedding. An important difference between processes for Local Action Systems and the process notions that have been introduced for other systems, for example, Petri nets, is the presence of a component describing the embedding mechanism. The aim of the paper is to develop a methodology for dealing with this embedding mechanism: we introduce a suitable representation (a dynamic structure) for it, and then investigate the algebraic properties of this representation. This leads to a simple characterization of the configurations of a process and to a number of equational laws for dynamic structures. We illustrate the use of these laws by providing an equational proof of one of the basic results for LAS processes, namely that the construction yielding the result graph of a process behaves well with respect to the sequential composition of processes.

1. Introduction

It has been recognized for some time that graph rewriting systems are a powerful tool for the description and analysis of concurrent systems (Rozenberg 1997; Ehrig *et al.* 2000), and a significant amount of research has led to a better understanding of the relationship between graph rewriting and other models of concurrency, in particular Petri nets (see, for example, Corradini (1996)). A notion that is of special interest in this context is the notion of a process (see, for example, Goltz and Reisig (1983)) for Petri nets, in which the causality relation is explicitly represented. Processes and related notions, such as Pomsets (Pratt 1987) and Event Structures (Winskel 1980), have been studied extensively in the literature, and have turned out to be a powerful tool for the study of concurrent semantics (see, for example, Sassone (2000) and Baldan *et al.* (1998) for a survey of some recent work on processes, and, in particular, on concatenable processes. For graph rewriting based on gluing, a process notion has been introduced in Corradini *et al.* (1996).

[†] This work was partially supported by the EC TMR Network GETGRATS (General Theory of Graph Transformation Systems) and Esprit Working Group APPLIGRAPH through Universitaire Instelling Antwerpen.
[‡] This paper is a complete and revised version of a lecture given at the workshop on Theory and Applications of Graph Transformations (GRATRA 2000), which was a satellite event of ETAPS 2000 in Berlin, March 2000.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

The various types of graph transformation systems that have been introduced in the literature can be roughly divided into two kinds: those in which an atomic step is based on the use of a gluing or pushout construction, and the others (Engelfriet and Rozenberg 1997), in which the general idea is that an atomic step consists of the replacement of a subgraph g of a graph h by a new graph g', which is then connected to the remaining part of h according to a so-called embedding mechanism. The aim of Local Action Systems (LAS), which were introduced in Janssens (1999), is to provide a model that generalizes a number of such embedding based approaches and allows a process notion for the description of concurrent runs. There are several important differences between the graph processes of Corradini *et al.* (1996) and those for LAS. First, the type of graph rewriting is very different: since LAS are based on replacement and embedding, their processes must contain information about the embedding mechanism, and the construction of configurations becomes more complicated. Second, LAS processes are defined on the level of the alphabets involved, independently of the choice of a particular set of productions. This has the advantage that it facilitates the definition of semantic equivalences. It may also be useful to note that, in contrast to what is the case in many other approaches to true concurrency, such as Pomsets (Pratt 1987) and Event Structures (Winskel 1980), actions or events are not represented by points, but by labelled edges of the DAG that represents the direct causality relation. This implies that Pomsets and Event Structures are more abstract than LAS processes, because they do not represent the system state, but just the events.

The focus of this paper is on the analysis of individual processes, and on the way one can deal with the additional complexity of incorporating an embedding mechanism. The algebraic properties of its representation by dynamic structures, which was introduced in Janssens (1999), are investigated: some simple operations on dynamic structures are defined, and then used to express the configurations of a process. A number of equations for expressions involving dynamic structures are derived, and their use is demonstrated by providing an equational proof of one of the central properties of LAS processes: the coherence of the construction for configurations, that is, the fact that the construction behaves well with respect to the sequential composition of processes.

The paper is organized as follows. In Section 2 some basic notions and terminology are introduced, and in Section 3 the basic ideas of local actions and LAS processes are developed. In Section 4 a mechanism is given for extracting from a process the configurations (graphs) that occur in its execution. In Section 5 productions and systems are introduced, and the process semantics of a LAS system is defined. In the LAS approach the processes of a system are obtained by gluing together production occurrences, which are in themselves simple processes. In Section 6 the algebraic properties of dynamic structures are considered.

2. Preliminaries

1 For a set V, the powerset of V is denoted by $\mathscr{P}(V)$. The set $\{(x,x) \mid x \in V\}$ is denoted by $diag(V)$. For a relation $R \subseteq V \times V$, we use R^* to denote the transitive and reflexive closure of R. For a set $A \subseteq V$, we use $R(A)$ to denote the set $\{v \in V \mid$

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

there exists an $a \in A$ such that $(a, v) \in R\}$. The free commutative monoid on a set X is denoted by $FCM(X)$. Its elements may be viewed as functions from X into the set of natural numbers, and the neutral element is the constant zero function.

2 Functions are assumed to be total. For a function $f : X \to Y$ and $A \subseteq X$, the restriction of f to A, denoted $f|_A$, is the function on A where for each $x \in A$, $f|_A(x) = f(x)$. $f^{(2)} : X \times X \to Y \times Y$ is the function defined by: $f^{(2)}((x_1, x_2)) = (f(x_1), f(x_2))$, for each $x_1, x_2 \in X$. For functions f_1, f_2, the notation $\langle f_1 * f_2 \rangle$ is used to denote function composition: first f_1, then f_2. Hence $\langle f_1 * f_2 \rangle(x) = f_2(f_1(x))$.

3 Let $f : X \to Y$ be a function, let B be a set such that $X \subseteq B$ and let $(Y, +)$ be a monoid with neutral element $\mathbf{n}$. Then $d(f) = \{x \in X \mid f(x) \neq \mathbf{n}\}$, and the extension of f to B, denoted $f|^B$, is the function on B where for each $x \in X$, $f|^B(x) = f(x)$, and for each $x \in B \setminus X$, $f|^B(x) = \mathbf{n}$. For functions $f_1 : X \to Y$ and $f_2 : X \to Y$, $f_1 + f_2$ is the function $h : X \to Y$ where $h(x) = f_1(x) + f_2(x)$ for each $x \in X$.

4 Let V be a set and let $R \subseteq V \times V$. A *path in* R is a sequence $p = x_0, x_1, \ldots, x_n$ of elements of V such that $n \geqslant 0$ and, for each $0 \leqslant i \leqslant n - 1$, $(x_i, x_{i+1}) \in R$. n is the *length* of the path p, and p is a path *from* x_0 *into* x_n. The fact that p is a path from x into y is denoted by $p : x \to y$. The concatenation of paths is denoted by $\odot$: if $p = x_0, x_1, \ldots, x_n$ and $q = x_n, x_{n+1}, \ldots, x_{n+k}$, then $p \odot q$ is the path $x_0, x_1, \ldots, x_{n+k}$. The relation R is *acyclic* if there are no non-trivial cycles; that is, if $p = x_0, x_1, \ldots, x_n$ is a path in R such that $x_0 = x_n$, then $n = 0$. If V is finite and the relation R is acyclic, $depth(R)$ denotes the maximal length of a path in R.

5 Let V be a set and let R be a partial order on V. A *cut of* R is a maximal subset K of V such that, for each $x, y \in K$, $(x, y) \notin R$ and $(y, x) \notin R$. For a subset X of V, the sets $Pre(X)$ and $Post(X)$ are defined by

$$Pre(X) = X \cup \{v \in V \mid \text{there exists an } x \in X \text{ such that } (v, x) \in R\},$$

$$Post(X) = X \cup \{v \in V \mid \text{there exists an } x \in X \text{ such that } (x, v) \in R\}.$$

3. Local actions and processes

In this section we will introduce the basic notions concerning local actions and LAS processes. In LAS graph rewriting, it is assumed that each rewriting step has three aspects: testing, creation and transformation. In each step, a set A of nodes is replaced by a set B of new nodes. In doing so, first one has to *test* whether a certain graph structure is present on A – this graph is an occurrence of the left-hand side of a production. Then, a graph structure on B is *created* – this structure corresponds to the right-hand side of a production. Finally, the graph structure on A is *transformed* into a graph structure on B, and combined with the structure created.

Arguably, the most basic idea in the LAS approach to graph rewriting is that the transformation of graph structures is based on local actions: each of these transforms information about a node x into information about a node y. This implies that y is causally dependent on x. Information about a node x can be of three kinds: the node label of x; labels of outgoing edges of x; and labels of incoming edges of x. Thus it is assumed that each local action t consists of three components t^v, t^s and t^t acting on

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

those three kinds of information. The effect of a local action is illustrated in Figure 1: the action $t = (t^v, t^s, t^t)$ transforms the graph g into the graph h. The labels α, a and β are transformed into $t^t(\alpha)$, $t^v(a)$ and $t^s(\beta)$, respectively. Obviously, it is possible that two local

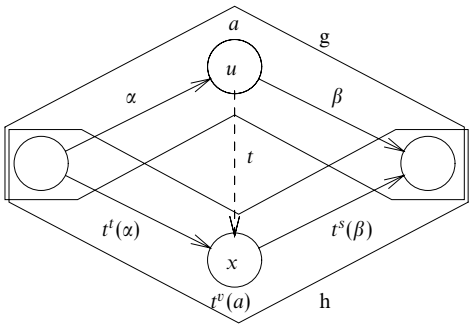


Fig. 1. The effect of a local action.

actions may be involved in the transformation of a single edge, as illustrated by Figure 2 (a). In this case t_1 and t_2 are not causally related. Since it is a fundamental assumption in the theory of processes that the order of events that are not causally related should not have an effect on their combined effect, it is assumed that, for each pair of local actions t_1 and t_2, their second and third components commute: $\langle t_1^s * t_2^t \rangle = \langle t_2^t * t_1^s \rangle$.

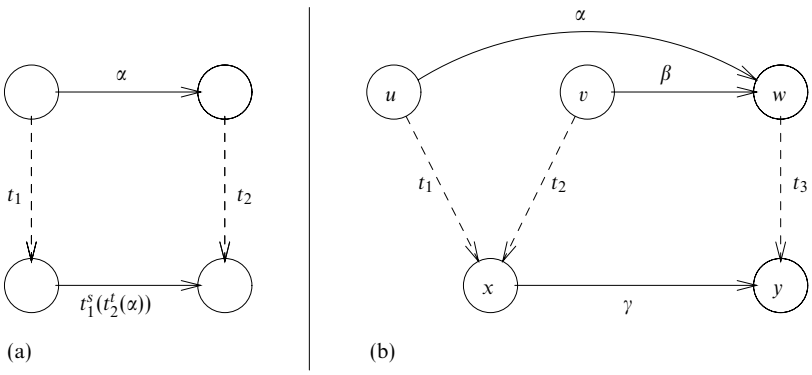


Fig. 2. The transformation of edges.

Since information about a node can be the result of several actions (for example, in Figure 2 (b), t_1 and t_2 yield information about x) there has to be a way to combine these results. Therefore, we assume that the sets of node and edge labels (Δ^v and Δ^e) are equipped with an operation $+$, making them into commutative monoids. In Figure 2 (b), if γ represents the combined result of the action of t_1^s and t_3^t on α, and the action t_2^s and t_3^t on β, then $\gamma = t_1^s(t_3^t(\alpha)) + t_2^s(t_3^t(\beta))$. Moreover, since in Figure 2 (b), t_1 and t_2 are not causally dependent on t_3, they can possibly happen before t_3, and then one would expect $\gamma = t_3^t(t_1^s(\alpha) + t_2^s(\beta))$. For this reason, it is assumed that the t^v, t^s and t^t are endomorphisms (t^v on Δ^v, t^s and t^t on Δ^e).

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

Summarizing, the basic assumptions for alphabets for Local Action Systems are the following:

— The set of node labels $(\Delta^v, +^v)$ and the set of edge labels $(\Delta^e, +^e)$ are commutative monoids. The neutral elements of Δ^v and Δ^e are denoted by $\mathbf{n}^\mathbf{v}$ and $\mathbf{n}^\mathbf{e}$, respectively.

— A local action is a triple $t = (t^v, t^s, t^t)$ in $T^v \times T^s \times T^t$, where T^v is a set of endomorphisms on $(\Delta^v, +^v)$ and T^s, T^t are sets of endomorphisms on $(\Delta^e, +^e)$. The sum of endomorphisms t_1, t_2 is defined by $(t_1 + t_2)(x) = t_1(x) + t_2(x)$. It is assumed that T^v, T^s and T^t are closed under sum and composition.

— For each $t^s \in T^s$ and each $t^t \in T^t$, we have $\langle t^s * t^t \rangle = \langle t^t * t^s \rangle$.

Recall from Section 2 that $*$ denotes function composition, which is an associative operation. The identity endomorphisms on Δ^v and Δ^e are denoted by $\mathbf{1}^\mathbf{v}$ and $\mathbf{1}^\mathbf{e}$, respectively, and the constant endomorphisms on Δ^v and Δ^e mapping everything into the neutral element are denoted by $\mathbf{0}^\mathbf{v}$ and $\mathbf{0}^\mathbf{e}$, respectively. To avoid unnecessarily heavy notation, we often simply write $\mathbf{1}$ instead of $\mathbf{1}^\mathbf{v}$, $\mathbf{1}^\mathbf{e}$, and similarly for $\mathbf{0}$ and $\mathbf{n}$. Also, the monoid operation is often denoted simply by $+$. Throughout this paper, Δ^v, Δ^e, T^s and T^t are assumed fixed.

As the LAS approach is based on the manipulation of labels, it is convenient to think of graphs as sets of nodes equipped with two labelling functions: one for node labels and one for edge labels.

Definition 1. A *graph* g on a set of nodes V is a pair of functions (g^v, g^e), where $g^v : V \to \Delta^v$ and $g^e : V \times V \to \Delta^e$.

If one wishes to represent graphs that are not complete, the neutral element $\mathbf{n}^\mathbf{e}$ of Δ^e may be interpreted as indicating the absence of an edge; so in that case the set $d(g^e)$ is viewed as the set of edges of g.

Example 1. Let $\Sigma_1 = \{a, b, c\}$, $\Sigma_2 = \{\alpha, \beta, \gamma\}$, $\Delta^v = FCM(\Sigma_1)$ and $\Delta^e = (\mathscr{P}(\Sigma_2), \cup)$. Then Figure 3 depicts a graph g on $\{1, 2, 3\}$, where $d(g^v) = \{1, 2, 3\}$ and $d(g^e) = \{(1, 2), (3, 2)\}$.

In the figures, only non-zero labels are mentioned, and only edges with a non-zero label are drawn. Observe that, using the FCM construction, one can always embed an arbitrary set into a commutative monoid, as we have done in this example.

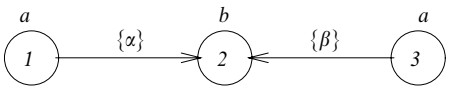


Fig. 3. A graph.

We will use the following notions and notation for graphs.

— The graph $g_\mathbf{0}^V$ is the graph (h^v, h^e) on V such that, for each $x, y \in V$, we have $h^v(x) = \mathbf{n}^\mathbf{v}$ and $h^e(x, y) = \mathbf{n}^\mathbf{e}$.

— Let V be a set, and let $A \subseteq V$ and $V \subseteq B$. For a graph g on V, the *subgraph of g induced on A*, denoted $g|_A$, is the graph $(g^v|_A, g^e|_{A \times A})$ on A. The *extension of g to B*, denoted by $g|^B$, is the graph $(g^v|^B, g^e|^{B \times B})$ on B.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

A central part of the description of a rewriting history is the description of the local actions that take place between causally dependent nodes. For a pair (x, y) of nodes, the local action t associated with it computes information about y on the basis of information about x. In this case, y is causally dependent on x. Hence the local actions of a rewriting history may be represented by a graph with labelled edges, where the nodes are the nodes of the rewriting history and the edge labels represent the local actions. This graph is called the dynamic structure, as opposed to the static structure, which consists of the initial graph and the structure created *a priori* in the course of the rewriting, for example, as part of the right-hand sides of productions.

Definition 2. Let V be a set. A *dynamic structure* on V is a function $D : V \times V \to T^v \times T^s \times T^t$.

We will use the following notions and notation for dynamic structures.

— The three components of a dynamic structure D are denoted by D^v, D^s and D^t: hence $D(x, y) = (D^v(x, y), D^s(x, y), D^t(x, y))$.

— $D_{\mathbf{0}}^V$ denotes the dynamic structure on V such that, for each $x, y \in V$, we have $D_{\mathbf{0}}^V(x, y) = (\mathbf{0}, \mathbf{0}, \mathbf{0})$. For sets $A \subseteq V$, $\mathbf{1}_A^V$ is the dynamic structure on V such that

$$\mathbf{1}_A^V(x, y) = \begin{cases} (\mathbf{1}, \mathbf{1}, \mathbf{1}), & \text{if } x, y \in A \\ (\mathbf{0}, \mathbf{0}, \mathbf{0}), & \text{otherwise.} \end{cases}$$

— For a dynamic structure D on V and a set W such that $V \subseteq W$, we use $D|^W$ to denote the extension of D to $W \times W$, that is, we write $D|^W$ for $D|^{W \times W}$. We use $d(D)$ to denote the subset of $V \times V$ where not all components of D are $\mathbf{0}$, that is, $d(D) = \{(x, y) \mid D(x, y) \neq (\mathbf{0}, \mathbf{0}, \mathbf{0})\}$. If $d(D)$ is acyclic, we use $depth(D)$ to denote $depth(d(D))$.

The following conventions are used when depicting a dynamic structure D. For a pair of nodes (x, y), a non-zero $D(x, y)$ is represented by a dashed arrow from x to y, and $D^s(x, y)$ and $D^t(x, y)$ are depicted to the left and the right of that arrow, respectively. Since in all the examples considered in the paper, the D^v-components of a dynamic structure are $\mathbf{0}$, these are not represented in the figures.

Example 2. Let Δ^v, Δ^e be the monoids from Example 1, let T^s be the set of all endomorphisms on Δ^e, let $T^t = \{\mathbf{0}, \mathbf{1}\}$, and, for each $a, b \in \Sigma_2$, let t_{ab} be the endomorphisms on Δ^e determined by

$$t_{ab}(\{z\}) = \begin{cases} \{b\}, & \text{if } z = a \\ \varnothing, & \text{otherwise.} \end{cases}$$

Let $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and let D be the dynamic structure on V where all components are $\mathbf{0}$, except the following ones: $D^s(1, 4) = t_{\alpha\beta}$, $D^s(1, 5) = t_{\alpha\gamma}$, $D^s(3, 6) = t_{\beta\beta}$, $D^s(4, 7) = \mathbf{1}$, $D^s(5, 7) = t_{\gamma\gamma}$, $D^s(6, 8) = t_{\beta\alpha}$, $D^t(2, 5) = D^t(5, 7) = \mathbf{1}$. Then D is the dynamic structure depicted in Figure 5 (a).

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

The monoid operations on Δ^v and Δ^e are extended to graphs and dynamic structures as follows.

Definition 3. Let V_1, V_2 be sets and let $V = V_1 \cup V_2$. Let g_1, g_2 be graphs on V_1, V_2, respectively, and let D_1, D_2 be dynamic structures on V_1, V_2, respectively.

1 $g_1 + g_2$ is the graph $(g_1^v|^V + g_2^v|^V, g_1^e|^{V\times V} + g_2^e|^{V\times V})$ on V.
2 $D_1 + D_2$ is the dynamic structure $(D_1|^V + D_2|^V)$ on V.

Let V be a set. Let $\mathcal{G}(V) = \{g \mid g \text{ is a graph on } W, W \subseteq V\}$ and let $\mathcal{D}(V) = \{D \mid D \text{ is a dynamic structure on } W, W \subseteq V\}$. Then Definition 3 makes $\mathcal{G}(V)$ and $\mathcal{D}(V)$ into commutative monoids.

Example 3. Let D be the dynamic structure from Example 2. Let $V_1 = \{1,2,3,4,5,6\}$ and $V_2 = \{4,5,6,7,8\}$. Let D_1 and D_2 be the dynamic structures on V_1 and V_2, respectively, shown in Figure 4. Then $D = D_1 + D_2$.

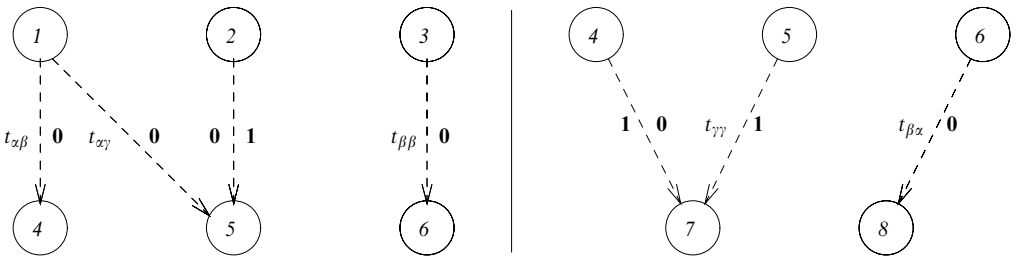


Fig. 4. The dynamic structures D_1 and D_2.

One can now define the main notion used to describe the behaviour of Local Action Systems: the notion of a process. A process describes a system run or rewriting history; it specifies all the nodes occurring in the course of a run, the graph structure that is either initially present or created, the causality relation between the nodes, and the dynamic structure. Thus, in defining the notion of a process, the structure present at the beginning of a rewriting history is formally treated in the same way as the structure created in the course of it.

Definition 4. A *process* is a 4-tuple (V, C, D, S), where V is a finite set, $C \subseteq V \times V$ is an acyclic relation, D is a dynamic structure on V such that $d(D) \subseteq C$, and S is a graph on V.

The partial order corresponding to C is called the *causal order* of *proc*, and it is denoted by $<_C$. Thus $x <_C y$ if there is a path $p : x \to y$ of length at least one in C. *Min*(*proc*) and *Max*(*proc*) denote the sets of minimal and maximal nodes of V with respect to $<_C$, respectively. The graph g on a set V may be identified with the process $(V, \varnothing, D_{\mathbf{0}}^V, g)$, and thus operations defined on *proc* (such as the composition, Definition 5) can be applied to graphs.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

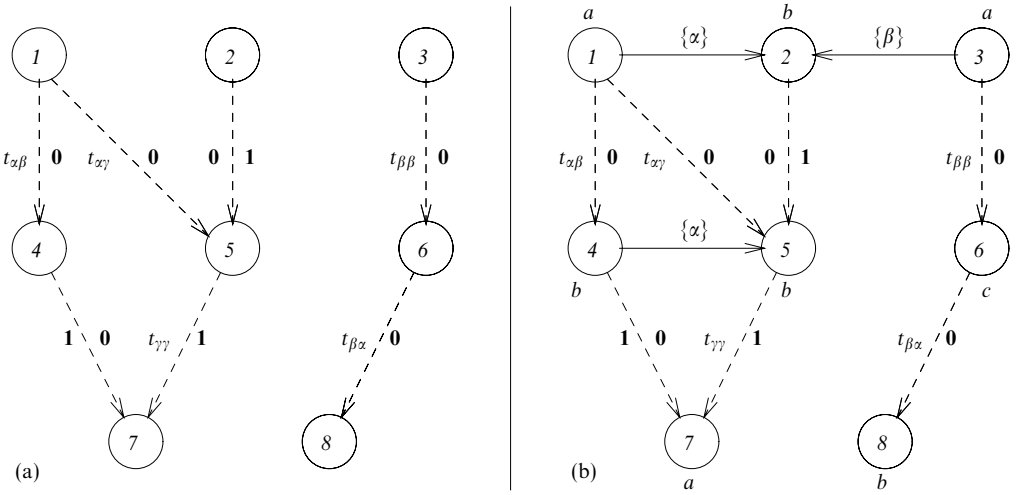


Fig. 5. A dynamic structure (a) and a process (b).

Example 4. Let $V = \{1,2,3,4,5,6,7,8\}$ and let $C = \{(1,4),(1,5),(2,4),(2,5),(3,6),(4,7),(4,8),(5,7),(5,8),(6,7),(6,8)\}$. Then $proc = (V, C, D, S)$ is the process shown in Figure 5 (b), where $d(S^v) = \{1,2,3,4,5,6,7,8\}$, $d(S^e) = \{(1,2),(3,2),(4,5)\}$ and D is the dynamic structure from Example 2.

(Note that pictures of processes do not show the causality relation explicitly.)

Observe that the last component of a process may contain edges with a non-zero label between causally related nodes. Obviously, such edges do not occur in any configuration, but, despite this, they may be useful, for example, if one wants a process to contain edges that are used as annotations expressing explicitly relationships between causally related nodes, such as 'every a-labelled node has a b-labelled causal predecessor'. These edges may facilitate the analysis of the processes of a system.

Since processes may be quite large, it is natural to seek ways to decompose them, and to infer properties of the whole process from properties of its parts. Moreover, when the set of processes of a system is defined (Section 5), one needs a way to express the fact that a process can be decomposed into building blocks that are occurrences of productions. For a process $proc$ on a set of nodes V, such a decomposition yields concrete processes, $proc_1$ and $proc_2$, say, on subsets of V, and the way these subsets overlap determines how they interact; for example, some output (maximal) nodes of $proc_1$ may be input (minimal) nodes of $proc_2$. Thus, in this context, it is sufficient to introduce a very simple composition operation on concrete processes. A more abstract presentation of processes and dynamic structures can be obtained using a technique similar to the one used for concatenable processes (Baldan *et al.* 1998), by considering classes of triples (σ, p, τ), where p is a process and σ, τ are morphisms relating concrete graphs to the initial and result graphs of p (Definition 8), and replacing $\oplus$ by a pushout construction. A more abstract approach was also taken for ESM systems in Janssens (1994), but there the pairs of nodes to be identified in composing two processes is given by an explicit interface.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

Definition 5. Let $proc_1 = (V_1, C_1, D_1, S_1)$ and $proc_2 = (V_2, C_2, D_2, S_2)$ be processes such that $V_1 \cap V_2 = (Max(proc_1) \cap Min(proc_2)) \cup (Min(proc_1) \cap Max(proc_2))$, and $C_1 \cup C_2$ is acyclic. The *composition* of $proc_1$ and $proc_2$, denoted $proc_1 \oplus proc_2$, is the process $(V_1 \cup V_2, C_1 \cup C_2, D_1 + D_2, S_1 + S_2)$.

Note that the composition of $proc_1$ and $proc_2$ is defined only if their sets of nodes overlap in a particular way: input (that is, minimal) nodes of $proc_1$ may occur as output (maximal) nodes of $proc_2$ and *vice versa*. An important special case occurs when the processes $proc_1$ and $proc_2$ overlap only in nodes that are maximal in $proc_1$ and minimal in $proc_2$, that is, $V_1 \cap V_2 \subseteq Max(proc_1) \cap Min(proc_2)$. In this case no node of $proc_1$ is causally dependent on a node of $proc_2$. Informally, this means that the execution of $proc_1$ may be completed before that of $proc_2$ begins. If, moreover, $V_1 \cap V_2 = Max(proc_1) = Min(proc_2)$, then $proc_1 \oplus proc_2$ is called a sequential composition.

The next result states that, for a process $proc = (V, C, D, S)$ and a cut K of $<_C$, we have K divides $proc$ into two parts $proc_1$ and $proc_2$, with sets of nodes $Pre(K)$ and $Post(K)$, respectively. Informally, this means that K is the set of maximal nodes of an initial part $proc_1$ of $proc$, and hence K corresponds to an intermediate situation of the rewriting history described by $proc$: the situation reached after the execution of $proc_1$. The extra conditions on $proc$ that occur in the statement of the theorem are always satisfied for processes built from production occurrences (Section 5).

Theorem 1. Let $proc = (V, C, D, S)$ be a process and let K be cut of $<_C$ such that C and $d(S^e)$ are subsets of $(Pre(K) \times Pre(K)) \cup (Post(K) \times Post(K))$. Then there exist processes $proc_1$ and $proc_2$ such that $Min(proc_2) = Max(proc_1) = K$ and $proc = proc_1 \oplus proc_2$.

Proof. Let $proc_1 = (V_1, C_1, D_1, S_1)$ and $proc_2 = (V_2, C_2, D_2, S_2)$ where $V_1 = Pre(K)$, $C_1 = C \cap (V_1 \times V_1)$. $D_1 = D|_{V_1 \times V_1}$, $S_1 = S|_{V_1}$, $V_2 = Post(K)$, $C_2 = C \cap (V_2 \times V_2)$, $D_2 = D|_{V_2 \times V_2}$, $S_2^v = (S^v|_{V_2 \setminus V_1})\Big|^{V_2}$ and $S_2^e = (S^e|_{(V_2 \times V_2) \setminus (V_1 \times V_1)})\Big|^{V_2 \times V_2}$. The conditions on C and $d(S^e)$ assure that $C = C_1 \cup C_2$ and $S = S_1 + S_2$. The result then follows. □

4. Configurations

In order to characterize the set of processes generated by a set of productions (in Section 5), we need to take into account what is usually called the left-hand side of a production: a graph that should occur in the configuration that is reached at the time the production is applied. Thus, we first need a characterization of the configurations.

Intuitively, for a process (V, C, D, S) describing a rewriting history, C describes the direct causality relation between nodes, and thus the minimal and maximal nodes with respect to the causal order $<_C$ are the nodes of the initial graph and of the final graph of the rewriting history, respectively. The other nodes are nodes of intermediate configurations. As also happens in Petri Net processes, the maximal sets of causally unrelated nodes (the cuts) are the sets of nodes of intermediate configurations.

In order to define the graph structure of intermediate configurations, consider the way labels are transformed by local actions. First, consider node labels. For a path p from u into x in C, the local actions on p are applied in the specified order, transforming

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

the node label of u into a node label for x. However, to compute the node label of x, we have to take into account all paths from u into x, and to combine them using the monoid operation $+^v$ on Δ^v. Thus the label l of u is transformed into the sum of the labels l_p obtained by applying the actions on paths p, from u into x, to l. Second, consider edge labels. The mechanism is similar to that for node labels, except that we have to use pairs of paths: for a pair of paths $p : u \to x$ and $q : v \to y$, the label l of (u, v) is transformed into an edge label l_{pq} by applying the actions on p and q (or rather, their s- and t-components, respectively) to l. The assumptions about the local actions ensure that the relative order of actions on p and on q does not matter. Again, all pairs of paths p, q have to be taken into account, and the label l has to be transformed into the sum, over all p and q, of the labels l_{pq}.

In a process (V, C, D, S), the graph S represents all structure that is present without being produced by applying local actions. This is the structure that belongs to the right-hand side of the productions (to be introduced in Section 5) or to the initial graph of a process. The labels present in a configuration are obtained by taking the sum of all contributions corresponding to S: for a node x (a pair of nodes (x, y)), each node label (edge label) of S yields a contribution to the node label of x (edge label of (x, y)).

Formally, for a process $proc = (V, C, D, S)$, first the actions D^v_p, D^s_p, D^t_p corresponding to a path in the causal relation are defined, then the action D_{pq} corresponding to a pair of paths is defined, and then the graph $Conf(proc, W)$ is defined for each subset W of V: it is the graph on W obtained by applying local actions to S in the way sketched above.

Definition 6. Let (V, C, D, S) be a process. Let $p = x_0, \ldots, x_n$ be a path in C. Then D^v_p, D^s_p and D^t_p are the endomorphisms on Δ^v, Δ^e and Δ^e, respectively, defined as follows.

— If $n = 0$, then $D^v_p = \mathbf{1}^{\mathbf{v}}$ and $D^s_p = D^t_p = \mathbf{1}^{\mathbf{e}}$.
— If $n \geqslant 1$, then

$$\begin{aligned} D^v_p &= \langle D^v(x_0, x_1) * D^v(x_1, x_2) * \ldots * D^v(x_{n-1}, x_n)\rangle \\ D^s_p &= \langle D^s(x_0, x_1) * D^s(x_1, x_2) * \ldots * D^s(x_{n-1}, x_n)\rangle \\ D^t_p &= \langle D^t(x_0, x_1) * D^t(x_1, x_2) * \ldots * D^t(x_{n-1}, x_n)\rangle. \end{aligned}$$

Furthermore, for paths p and q in C, D_{pq} is the endomorphism on Δ^e defined by $D_{pq} = \langle D^s_p * D^t_q\rangle$.

Example 5. Let D be the dynamic structure from Example 2. Consider the paths $p_1 = 3, 6, 8$, $p_2 = 1, 5, 7$ and $q = 2, 5, 7$. Then $D_{p_1 q} = t_{\beta\beta} * t_{\beta\alpha} * \mathbf{1} * \mathbf{1} = t_{\beta\alpha}$ and $D_{p_2 q} = t_{\alpha\gamma} * t_{\gamma\gamma} * \mathbf{1} * \mathbf{1} = t_{\alpha\gamma}$.

The following technical lemma will be useful.

Lemma 1. Let (V, C, D, S) be a process. Let p and q be paths in C such that $p = p_1 \odot p_2$ and $q = q_1 \odot q_2$. Then

$$D_{pq} = \langle D_{p_1 q_1} * D_{p_2 q_2}\rangle.$$

Proof. By the associativity of $*$, $D^s_p = \langle D^s_{p_1} * D^s_{p_2}\rangle$, and, similarly, $D^t_q = \langle D^t_{q_1} * D^t_{q_2}\rangle$. Furthermore, since the s- and t-components of local actions commute, $\langle D^s_{p_2} * D^t_{q_1}\rangle =$

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

$\langle D^t_{q_1} * D^s_{p_2} \rangle$. The result now follows easily:

$$\begin{aligned} D_{pq} &= \langle D^s_p * D^t_q \rangle = \langle D^s_{p_1} * D^s_{p_2} * D^t_{q_1} * D^t_{q_2} \rangle \\ &= \langle D^s_{p_1} * D^t_{q_1} * D^s_{p_2} * D^t_{q_2} \rangle = \langle D_{p_1 q_1} * D_{p_2 q_2} \rangle. \end{aligned}$$

□

The graph $Conf(proc, W)$ is defined as follows.

Definition 7. Let $proc = (V, C, D, S)$ be a process and let $W \subseteq V$. $Conf(proc, W)$ is the graph (g^v, g^e) on W such that, for each $x, y \in W$,

$$\begin{aligned} g^v(x) &= \sum_{u \in V} \sum_{p : u \to x} D^v_p(S^v(u)) \\ g^e(x, y) &= \sum_{u,v \in V} \sum_{\substack{p : u \to x, \\ q : v \to y}} D_{pq}(S^e(u, v)) \end{aligned}$$

where p ranges over paths from u into x in C, and q ranges over paths from v into y in C.

In the above, it is assumed that a sum of 0 terms stands for the neutral element. Four special cases of the construction $Conf(proc, W)$ play a special role: $W = V$, W is a cut, $W = Min(proc)$ and $W = Max(proc)$.

Definition 8. Let $proc = (V, C, D, S)$ be a process.

1 The computed structure of *proc*, denoted $CS(proc)$, is the graph $Conf(proc, V)$.
2 A configuration of *proc* is a graph $Conf(proc, K)$ where K is a cut of $<_C$.
3 The initial graph of *proc*, denoted $Init(proc)$, is the graph $Conf(proc, Min(proc))$.
4 The result graph of *proc*, denoted $Res(proc)$, is the graph $Conf(proc, Max(proc))$.

Obviously, for each $W \subseteq V$, $Conf(proc, W)$ is an induced subgraph of $CS(proc)$. The initial and result graph of a process are special cases of configurations, corresponding to the minimal and maximal nodes, respectively. Informally, a process *proc* describes a graph rewriting history that transforms $Init(proc)$ into $Res(proc)$.

Example 6. Let *proc* be the process from Example 4. $Init(proc)$ is the graph depicted in Figure 3. The graphs depicted in Figure 6 are the intermediate configurations $Conf(proc, \{3, 4, 5\})$ and $Conf(proc, \{1, 2, 6\})$. $Res(proc)$ is the graph depicted in Figure 7.

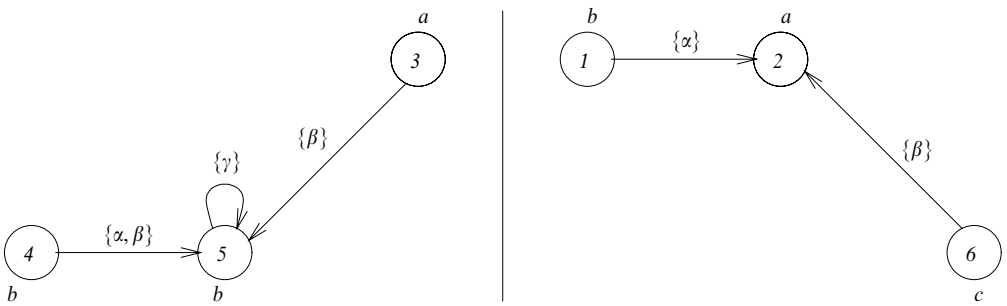


Fig. 6. Intermediate configurations.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

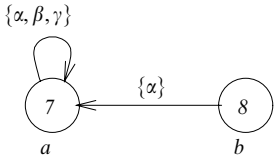


Fig. 7. The result graph.

An important special case of the composition $proc_1 \oplus proc_2$ with $V_1 \cap V_2 \subseteq Max(proc_1) \cap Min(proc_2)$ is that where $proc_1$ is a graph; that is, $proc_1$ is of the form $(V, \varnothing, D_{\mathbf{0}}^V, S)$. Then $proc_2$ can be viewed as an operation transforming $proc_1$ into the graph $Res(proc_1 \oplus proc_2)$: informally, this operation consists of executing the rewriting history described by $proc_2$. For a graph g and a process $proc$ such that $V_g \cap V_{proc} = Min(proc)$, the process $g \oplus proc$ is called the *application of proc to g*. In general, it is not true that $g = Init(g \oplus proc)$, due to the fact that the graph structure S of $proc = (V, C, D, S)$ may be non-trivial on the minimal nodes. If one wants to view *proc* as an operation transforming g, however, it seems natural to assume that *proc* creates only new structure on nodes that are causally dependent on the input (that is, minimal) nodes, and hence to consider the situation where this structure is indeed trivial: $d(S^v_{proc}) \cap Min(proc) = \varnothing$ and $d(S^e_{proc}) \cap (Min(proc) \times Min(proc)) = \varnothing$. In particular, this will be the case in processes generated by a set of productions (Section 5). Formally, we have the following proposition.

Proposition 1. Let g be a graph on V_g and let *proc* be a process such that $V_g \cap V_{proc} = Min(proc)$, $d(S^v_{proc}) \cap Min(proc) = \varnothing$ and $d(S^e_{proc}) \cap (Min(proc) \times Min(proc)) = \varnothing$. Then $Init(g \oplus proc) = g$.

The construction of configurations makes sense only if it is coherent, that is, if two processes $proc_1$ and $proc_2$ are sequentially composed, then the application of $proc_2$ to the result graph of $proc_1$ should yield the result graph of $proc_1 \oplus proc_2$ (see Figure 8). Next it is demonstrated that this is the case; first we will show a technical lemma.

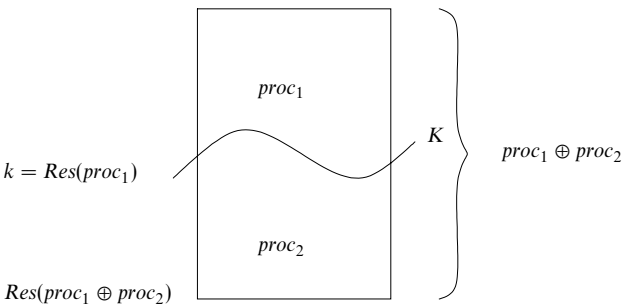


Fig. 8. Sequential composition.

Lemma 2. Let $proc_1 = (V_1, C_1, D_1, S_1)$ and $proc_2 = (V_2, C_2, D_2, S_2)$ be processes such that $V_1 \cap V_2 = Max(proc_1) \cap Min(proc_2)$. Let $p = x_1, \ldots, x_n$ be a path in $C_1 \cup C_2$ such that $x_1 \in V_1$ and $x_n \in V_2$. Then there is exactly one i, $1 \leqslant i \leqslant n$, such that $x_i \in Max(proc_1)$.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

Proof. Let i be minimal such that $x_i \in V_2$ (i exists because $x_n \in V_2$).Then $x_i \in V_1 \cap V_2$ because either $i = 1$ or $(x_{i-1}, x_i) \in C_1$. Thus $x_i \in Max(proc_1)$, and either $i = n$ or $(x_i, x_{i+1}) \in C_2$. In the latter case, $x_{i+1} \notin Min(proc_2)$, and thus $x_{i+1} \in V_2 \setminus V_1$. The assumption that $V_1 \cap V_2 = Max(proc_1) \cap Min(proc_2)$ implies that there are no paths from $V_2 \setminus V_1$ into V_1 in $C_1 \cup C_2$, and hence there is no $j > i$ such that $x_j \in V_1$. The result follows. □

Theorem 2. Let $proc_1 = (V_1, C_1, D_1, S_1)$ and $proc_2 = (V_2, C_2, D_2, S_2)$ be processes such that $V_1 \cap V_2 = Max(proc_1) \cap Min(proc_2)$. Then

$$Res(Res(proc_1) \oplus proc_2) = Res(proc_1 \oplus proc_2).$$

Proof. Let $(V, C, D, S) = proc_1 \oplus proc_2$, $K = Max(proc_1)$, $k = Res(proc_1)$, $h = Res(k \oplus proc_2)$ and $g = Res(proc_1 \oplus proc_2)$. It is obvious that $V_h = V_g = Max(proc_1 \oplus proc_2)$. Next, we show that $h^e = g^e$; the proof of $h^v = g^v$ is similar but simpler. Let $u, v \in K$ and $w, z \in Max(proc_1 \oplus proc_2)$. Consider the effect $E(u, v, w, z)$ of $k^e(u, v)$ on $h^e(w, z)$:

$$E(u, v, w, z) = \sum_{(p_2, q_2)} D_{p_2 q_2}(k^e(u, v)),$$

where (p_2, q_2) ranges over all pairs of paths $p_2 : u \to w$ and $q_2 : v \to z$ in C_2. On the other hand,

$$k^e(u, v) = \sum_{x,y \in V_1} \left(\sum_{(p_1, q_1)} D_{p_1 q_1}(S_1^e(x, y)) \right),$$

where (p_1, q_1) ranges over all pairs of paths $p_1 : x \to u$ and $q_1 : y \to v$ in C_1. It follows from Lemma 2 that, for $x \in V_1$ and $w \in V_2$, there is a one-to-one correspondence between the set of paths $p : x \to w$ in C and the set of 3-tuples (p_1, u, p_2) where $p_1 : x \to u$ is a path in C_1, $u \in Max(proc_1)$ and $p_2 : u \to w$ is a path in C_2. Moreover, by Lemma 1, if $p = p_1 \odot p_2$ and $q = q_1 \odot q_2$, for paths p_1, q_1 in C_1 and p_2, q_2 in C_2, then $D_{pq} = \langle D_{p_1 q_1} * D_{p_2 q_2} \rangle$. It follows that

$$\begin{aligned} E(u, v, w, z) &= \sum_{x,y \in V_1} \sum_{\substack{(p_1, q_1) \\ (p_2, q_2)}} \langle D_{p_1 q_1} * D_{p_2 q_2} \rangle (S_1^e(x, y)) \\ &= \sum_{x,y \in V_1} \sum_{(p_u, q_v)} D_{p_u q_v}(S_1^e(x, y)) \end{aligned}$$

where (p_u, q_v) ranges over all pairs of paths $p_u : x \to w$ and $q_v : y \to z$ in C such that u is the unique node of $Max(proc_1)$ on p_u, and v is the unique node of $Max(proc_1)$ on q_v. It follows that

$$\begin{aligned} \sum_{u,v \in K} E(u, v, w, z) &= \sum_{u,v \in K} \left(\sum_{x,y \in V_1} \sum_{(p_u, q_v)} D_{p_u q_v}(S_1^e(x, y)) \right) \\ &= \sum_{x,y \in V_1} \sum_{(p,q)} D_{pq}(S_1^e(x, y)) \end{aligned}$$

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

where (p, q) ranges over all pairs of paths $p : x \to w$ and $q : y \to z$ in C. Finally,

$$h^e(w, z) = \sum_{x,y \in K \cup V_2} \sum_{(p,q)} D_{pq}((k + S_2)^e(x, y))$$

where (p, q) ranges over all pairs of paths $p : x \to w$ and $q : y \to z$ in C_2. However, for $x, y \in K \cup V_2$, each path $p : x \to w$ in C is a path in C_2, and similarly for paths $q : y \to z$. Thus,

$$h^e(w, z) = \sum_{u,v \in K} E(u, v, w, z) + \sum_{x,y \in V_2} \sum_{(p,q)} D_{pq}(S_2^e(x, y))$$

where (p, q) ranges over all pairs of paths $p : x \to w$ and $q : y \to z$ in C instead of C_2. Thus

$$\begin{aligned} h^e(w, z) &= \sum_{x,y \in V_1} \sum_{(p,q)} D_{pq}(S_1^e(x, y)) + \sum_{x,y \in V_2} \sum_{(p,q)} D_{pq}(S_2^e(x, y)) \\ &= \sum_{x,y \in V} \sum_{(p,q)} D_{pq}(S_1^e(x, y)) + \sum_{x,y \in V} \sum_{(p,q)} D_{pq}(S_2^e(x, y)) \\ &= \sum_{x,y \in V} \sum_{(p,q)} D_{pq}(S^e(x, y)) \\ &= g^e(w, z). \end{aligned}$$

□

5. Local Action Systems and process semantics

The aim of this section is to introduce the notions of a production, a system, and the process semantics of a system. A production is a pair consisting of a process of a somewhat restricted form, and a graph on the minimal nodes of it. This graph serves as the left-hand side of the production: an occurrence of the production is only allowed if the left-hand side occurs in the graph rewritten by that occurrence. A system is simply a set of productions.

Definition 9. A *production* is a pair $(proc, L)$ such that $proc = (V, C, D, S)$ is a process, $(Min(proc), Max(proc))$ is a partition of V, $C = Min(proc) \times Max(proc)$, $d(S^v) \subseteq Max(proc)$, $d(S^e) \subseteq (Max(proc) \times Max(proc))$, and L is a graph on $Min(proc)$. A *system* is a set of productions.

For a production π, its components are denoted by $proc_\pi$ and L_π, respectively. The components of $proc_\pi$, in turn, are denoted by V_π,C_π,D_π and S_π.

Example 7. Let Δ^v, Δ^e be the monoids from Example 1 and let T^s, T^t be the sets of endomorphisms from Example 2. Then three productions π_1, π_2 and π_3 are depicted in Figure 9, where $C_{\pi_1} = \{1, 2\} \times \{3, 4\}$, $C_{\pi_2} = \{(1, 2)\}$ and $C_{\pi_3} = \{1, 2, 3\} \times \{4, 5\}$.

The processes corresponding to a given system are built by composing occurrences (isomorphic copies) of the processes $proc_\pi$, that is, the first components of productions of the system, and applying the composed processes to an initial graph. Each output node of a production occurrence is directly causally dependent on each input node, and

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

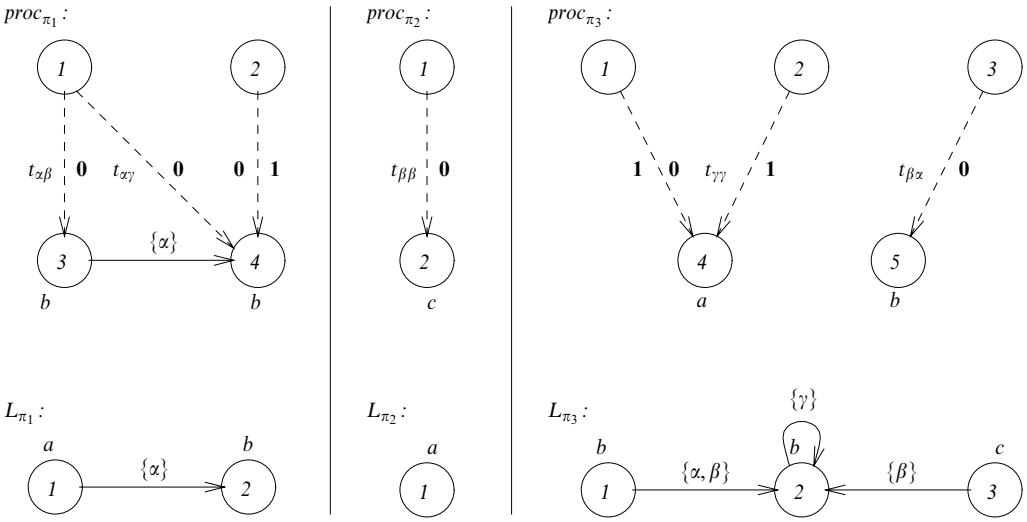


Fig. 9. The productions π_1, π_2 and π_3.

hence, each cut of a process built from production occurrences divides the production occurrences of that process into occurrences that causally precede the cut, and production occurrences that follow it. In a rewriting step new structure is only created on the newly introduced nodes, not on the ones that are rewritten. Note also that the notion of a system allows infinite sets of productions to be viewed as systems. Writing down, or implementing, such an infinite system would of course require some form of parameterization, but that aspect is not treated in this paper. The notion of a production occurrence is formally defined as follows.

Definition 10.

1 Let $proc_1 = (V_1, C_1, D_1, S_1)$ and $proc_2 = (V_2, C_2, D_2, S_2)$ be processes. An *isomorphism from* $proc_1$ *into* $proc_2$ is a bijective function $f : V_1 \to V_2$ such that $C_2 = f^{(2)}(C_1)$, $D_1 = D_2 \circ f^{(2)}$, $S_1^v = S_2^v \circ f$ and $S_1^e = S_2^e \circ f^{(2)}$.

2 An *occurrence of* a production $\pi = (proc_\pi, L_\pi)$ is a process v such that there exists an isomorphism from $proc_\pi$ into v.

Next the semantics of a system is defined. It consists of processes of the form $g \oplus proc$ with $proc = v_1 \oplus \ldots \oplus v_n$, where g is a graph and where the v_i are production occurrences such that $V_g \cap V_{proc} = Min(proc)$. Thus the process $g \oplus (v_1 \oplus \ldots \oplus v_n)$ describes the application of a process that is built from production occurrences to an initial graph g. However, not all processes of this form can be considered as valid in the sense that it describes a possible rewriting history: the left-hand sides of the productions should also be taken into account.

A set of production occurrences that can be composed is called a covering; the notion is formally defined as follows.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

Definition 11. Let *Sys* be a system. A *Sys-covering* is a set *cov* of occurrences of productions from *Sys* such that, for each $\mu, \nu \in cov$, $\mu \neq \nu$ implies $V_\mu \cap V_\nu = (Max(proc_\mu) \cap Min(proc_\nu)) \cup (Min(proc_\mu) \cap Max(proc_\nu))$, and $\bigcup_{\nu \in cov} C_\nu$ is acyclic.

A rewriting history in which a sequence of productions is applied to a graph is described by a process $g \oplus proc$, where g is the graph and *proc* describes the production occurrences. In the rewriting history described by $g \oplus proc$, each configuration reached by the time a production of *proc* is applied is a subgraph of $CS(g \oplus proc)$, as explained in Section 4, and hence the fact that the left-hand side L_π of a production π must occur in the configuration to which an occurrence ν of π is applied can be expressed by requiring that there exists an isomorphism f from $proc_\pi$ into ν such that the image of L_π under f is an induced subgraph of $CS(proc)$. Thus, a set of processes is associated to a system in the following way.

Definition 12. Let *Sys* be a system. The set of *processes of Sys* is the set of processes *proc* such that there exists a graph g and a *Sys*-covering *cov* for which $Min(proc) = V_g$, $proc = g \oplus \left(\bigoplus_{\nu \in cov} \nu\right)$, and for each $\nu \in cov$, there exist a production π of *Sys* and an isomorphism f from $proc_\pi$ into ν such that the image of L_π under f is an induced subgraph of $CS(proc)$.

Observe that in the definition we have used the fact that the graphs L_π can be viewed as processes.

One may order the elements of *cov* in a way that is compatible with the causal order, say $cov = \{\nu_1, \nu_2, \ldots, \nu_n\}$. Then $proc = (\cdots((g \oplus \nu_1) \oplus \nu_2) \oplus \cdots) \oplus \nu_n)$, with $V_{proc_i} \cap V_{\nu_i} = Min(\nu_i) \subseteq Max(proc_i)$ where $proc_i = (\cdots((g \oplus \nu_1) \oplus \nu_2) \oplus \cdots) \oplus \nu_{i-1})$ for each $i \in \{1, \ldots, n\}$. Thus each subprocess ν_i can be applied to the result graph of the subprocess $proc_i$. As a consequence of this observation and Theorem 2, the result graph of a *Sys*-process *proc* can be viewed as the result of a sequence of production applications, as illustrated by the following example.

Example 8. Let π_1, π_2, π_3 be the productions from Example 7, let $Sys = \{\pi_1, \pi_2, \pi_3\}$ and let g be the graph from Example 1. Then the process *proc* from Example 4 is a *Sys*-process: there exist occurrences ν_1, ν_2, ν_3 of π_1, π_2, π_3 (with node sets $V_1 = \{1, 2, 4, 5\}$, $V_2 = \{3, 6\}$ and $V_3 = \{4, 5, 6, 7, 8\}$, respectively) such that $proc = (((g \oplus \nu_1) \oplus \nu_2) \oplus \nu_3)$ with for each $i \in \{1, 2, 3\}$, $V_{proc_i} \cap V_{\nu_i} = Min(\nu_i) \subseteq Max(proc_i)$, where $proc_i = (\cdots((g \oplus \nu_1) \oplus \nu_2) \oplus \cdots) \oplus \nu_{i-1})$. Furthermore, there exist isomorphisms $f_1 : proc_{\pi_1} \to \nu_1$, $f_2 : proc_{\pi_2} \to \nu_2$ and $f_3 : proc_{\pi_3} \to \nu_3$ such that for each $i \in \{1, 2, 3\}$, the image of L_{π_i} under f_i equals $CS(proc)|_{Min(\nu_i)}$.

The requirements on the alphabets of labels and local actions of a LAS introduced in Section 3 may seem restrictive, severely limiting the possible concrete applications of the LAS framework. Nevertheless, it turns out that quite a lot of existing models can be described as Local Action Systems, including NLC grammars, ESM grammars, Actor grammars, Petri nets, Algebraic graph grammars with injective productions and Graph relabelling systems (Billaud *et al.* 1989). Although a formal treatment of the subject is outside the scope of this paper, a few remarks may help to clarify why this is the case. First, most models use alphabets that are just sets, and not monoids. This can be dealt

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

with by embedding those alphabets in a free commutative monoid, as we did in Example 1, provided that the productions are such that they do not introduce the new labels (formal sums of the original ones) when applied to a graph that contains only labels from the original alphabet. In fact, the alphabets of Example 1 are the ones used to encode ESM systems in Janssens (1999) and Verlinden and Janssens (2000). Second, most models use an embedding mechanism that either deletes edges or redirects them, but does not change their label. In those cases, the alphabet of local actions can be chosen to consist only of the identity and the constant zero function, which clearly satisfy the requirements.

The fact that NLC grammars, ESM grammars and Actor grammars may be viewed as LAS is demonstrated in Verlinden and Janssens (2000).

A Petri net may be encoded as a LAS system rewriting discrete graphs, where the node labels are names of places (embedded in a free commutative monoid). A marking m that assigns n tokens to a place s is represented by a discrete graph containing n nodes with label s. The only local action needed is $(\mathbf{0}, \mathbf{0}, \mathbf{0})$.

An algebraic graph grammar production with injective morphisms may be encoded by a LAS production as depicted in Figure 10, where, for each node x of K, there is a local

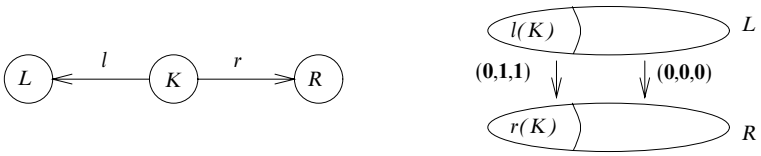


Fig. 10. An encoding of an algebraic graph grammar production into a LAS production

action $(\mathbf{0}, \mathbf{1}, \mathbf{1})$ from $l(x)$ into $r(x)$, and all the other local actions are $(\mathbf{0}, \mathbf{0}, \mathbf{0})$.

To encode Graph relabelling systems, one may use the obvious encoding of undirected graphs by directed ones, replacing an undirected edge by a pair of directed edges. Productions are such that each node x of the left-hand side gets replaced by a new node x', and there is a local action $(\mathbf{0}, \mathbf{1}, \mathbf{1})$ from x into x'. All other local actions are $(\mathbf{0}, \mathbf{0}, \mathbf{0})$. Node labels can be treated in the same way as in Example 1, and the original set of edge labels is embedded in a free commutative group: in this way the replacement of an edge label a by b can be obtained by taking the sum of a (preserved by the embedding) and $b + (-a)$ (in the right-hand side of the production).

6. Algebraic properties of dynamic structures

The composition operation $\oplus$ on processes is by itself only of limited use for the analysis of the relationship between a process and its configurations, because of the presence of an embedding mechanism. Unlike the case for Petri net processes, the configurations of a LAS process *proc* are not simply obtained by considering substructures (slices) of *proc*, but they have to be computed using the dynamic structure: they are substructures of $CS(proc)$. So, in order to understand the role that a particular part of the dynamic structure of *proc* plays in transforming the configurations of *proc*, we need to get an insight into the relationship between, on the one hand, the way dynamic structures are

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

composed and, on the other, the way they transform graphs. In this section we will show that each dynamic structure on a subset of a set V determines an operation on the set of graphs with nodes in V. Subsequently, we will introduce a new operation, $\star$, on dynamic structures; it represents the composition of the corresponding operations. Then we will give a number of equations relating the composition operations $+$ and $\star$ on dynamic structures to the operations $+$, restriction and extension of graphs. Finally, we will introduce an iterative operation *it* on dynamic structures, which enables us to express the graph $CS(proc)$ in a simple way, and then derive some equations for expressions containing *it*. Since the configurations of *proc* are substructures of $CS(proc)$, the equations can be used for reasoning about the configurations of a process. The use of equations is demonstrated by providing an alternative, equational proof of Theorem 2. First, we will define the application of a dynamic structure to a graph.

Definition 13. Let D be a dynamic structure on a set V_D and let g be a graph on a set V_g. Then $D[g]$ is the graph on $V = V_D \cup V_g$ defined by:

1 If $V_D = V_g$, then, for each $x, y \in V$,

$$(D[g])^v(x) = \sum_{z \in V} D^v(z,x)(g^v(z))$$

$$(D[g])^e(x,y) = \sum_{u,v \in V} \langle D^s(u,x) * D^t(v,y) \rangle (g^e(u,v))$$

2 If $V_D \neq V_g$, then $D[g] = D|^V[g|^V]$.

Example 9. Let g be the graph from Example 1. Let D be the dynamic structure from Example 2. Then $D[g]$ is the graph depicted in Figure 11 (a). Observe that all node labels equal $\mathbf{n}^{\mathbf{v}}$.

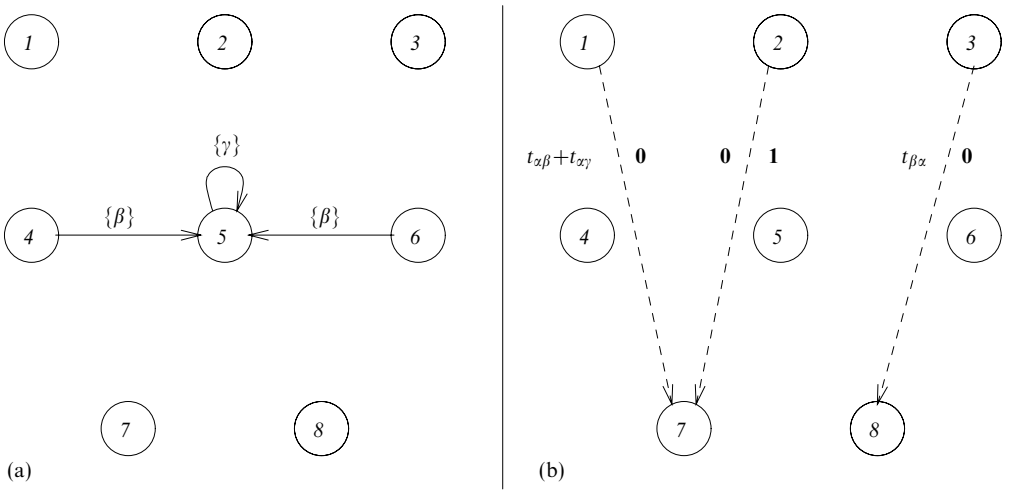


Fig. 11. The graph $D[g]$ (a) and the dynamic structure $D_1 \star D_2$ (b).

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

Thus D may be viewed as an operation transforming g into $D[g]$. Observe that the meaning of Definition 13 does not change if one replaces $\langle D^s(u,x) * D^t(v,y)\rangle$ by $\langle D^t(v,y) * D^s(u,x)\rangle$, because of the assumption that second and third components of local actions commute. Using the fact that local actions are endomorphisms, it is easily verified that, for a dynamic structure D and graphs g_1, g_2, we have $D[g_1 + g_2] = D[g_1] + D[g_2]$. Furthermore, for a dynamic structure D on V, $D[g_{\mathbf{0}}^V] = g_{\mathbf{0}}^V$. It follows that each dynamic structure on a set V determines an endomorphism on the set of graphs on V. The following theorem summarizes a few properties that follow easily from Definition 13.

Theorem 3. Let V, V_1, V_2 be sets, let g, g_1, g_2 be graphs on V, V_1, V_2, respectively, and let D, D_1 be dynamic structures on V, V_1, respectively.

1 $D[g_1 + g_2] = D[g_1] + D[g_2]$.
2 $D[g_{\mathbf{0}}^V] = g_{\mathbf{0}}^V$.
3 $1_V^V[g] = g$.
4 $D_{\mathbf{0}}^V[g] = g_{\mathbf{0}}^V$.
5 If $A \subseteq V \subseteq W$, then $g|_A|^W = (1_A^W)[g]$.
6 If $V_1, V_2 \subseteq W$, then $(D_1|^W)[g_2] = (D_1[g_2])|^W$.

The second operation, $\star$, on dynamic structures represents the composition of the corresponding endomorphisms. It is defined as follows.

Definition 14. Let D_1, D_2 be dynamic structures on sets V_1, V_2. Then $\langle D_1 \star D_2\rangle$ is the dynamic structure D on $V = V_1 \cup V_2$ defined by:

1 If $V_1 = V_2$, then for each $x, y \in V$,

$$D(x,y) = \sum_{z \in V} \langle D_1(x,z) * D_2(z,y)\rangle.$$

2 If $V_1 \neq V_2$, then $\langle D_1 \star D_2\rangle = \langle D_1|^V \star D_2|^V\rangle$.

We will often omit the brackets $\langle$ and $\rangle$ if there is no danger of confusion.

Example 10. Let D_1 and D_2 be the dynamic structures from Example 3. Then $D_1 \star D_2$ is the graph depicted in Figure 11 (b), where $t_{\alpha\beta} + t_{\alpha\gamma}$ is the endomorphism on Δ^e determined by

$$t_{\alpha\beta} + t_{\alpha\gamma}(\{z\}) = \begin{cases} \{\beta,\gamma\}, & \text{if } z = \alpha \\ \varnothing, & \text{otherwise.} \end{cases}$$

The following theorem summarizes a few properties that follow easily from Definitions 3 and 14.

Theorem 4. Let V, V_1, V_2, V_3 be sets such that $V_1, V_2 \subseteq V$, and let D, D_1, D_2, D_3 be dynamic structures on V, V_1, V_2, V_3, respectively.

1 $d(\langle D_1 \star D_2\rangle) \subseteq d(D_2) \circ d(D_1)$.
2 $D_1 + (D_2 + D_3) = (D_1 + D_2) + D_3$.
3 $D_1 + D_2 = D_2 + D_1$.
4 $D_1|^V + D_2|^V = (D_1 + D_2)|^V$.
5 $\langle D_1|^V \star D_2|^V\rangle = \langle D_1 \star D_2\rangle|^V$.
6 $1_V^V \star D = D = D \star 1_V^V$.
7 $D_1 \star \langle D_2 \star D_3\rangle = \langle D_1 \star D_2\rangle \star D_3$.
8 $D_1 \star (D_2 + D_3) = \langle D_1 \star D_2\rangle + \langle D_1 \star D_3\rangle$.
9 $(D_1 + D_2) \star D_3 = \langle D_1 \star D_3\rangle + \langle D_2 \star D_3\rangle$.
10 $1_{V_1}^V \star 1_{V_2}^V = 1_{V_1 \cap V_2}^V$.
11 $1_{V_1}^V \star D_1|^V = D_1|^V = D_1|^V \star 1_{V_1}^V$.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

Since each dynamic structure on V represents an endomorphism on the set of graphs on V, we may expect that the composition $\star$ of dynamic structures represents the composition of the corresponding endomorphisms. The following result states that this is indeed the case.

Theorem 5. Let g be a graph on V_g, and let D_1, D_2 be dynamic structures on V_1, V_2, respectively. Then $\langle D_1 \star D_2 \rangle[g] = D_2[D_1[g]]$.

Proof. Both $\langle D_1 \star D_2 \rangle[g]$ and $D_2[D_1[g]]$ are graphs on $V = V_1 \cup V_2 \cup V_g$. If $V_1 = V_2 = V_g$, then, for each $x, y \in V$,

$$
\begin{aligned}
D_2[D_1[g]]^e(x,y) &= \sum_{u,v \in V} \langle D_2^s(u,x) * D_2^t(v,y) \rangle ((D_1[g])^e(u,v)) \\
&= \sum_{w,z \in V} \sum_{u,v \in V} \langle D_1^s(w,u) * D_1^t(z,v) * D_2^s(u,x) * D_2^t(v,y) \rangle (g^e(w,z)) \\
&= \sum_{w,z \in V} \langle \left(\sum_{u \in V} D_1^s(w,u) * D_2^s(u,x) \right) * \left(\sum_{v \in V} D_1^t(z,v) * D_2^t(v,y) \right) \rangle (g^e(w,z)) \\
&= \sum_{w,z \in V} \langle (D_1 \star D_2)^s(w,z) * (D_1 \star D_2)^t(v,y) \rangle (g^e(w,z)) = (\langle D_1 \star D_2 \rangle)[g])^e.
\end{aligned}
$$

One may prove in a similar (but simpler) way that $D_2[D_1[g]]^v = (\langle D_1 \star D_2 \rangle [g])^v$. If it is not true that $V_1 = V_2 = V_g$, then $D_2[D_1[g]] = D_2|^V [D_1|^V [g|^V]] = \langle D_1|^V \star D_2|^V \rangle [g|^V] = \langle D_1 \star D_2 \rangle [g]$. □

We use the notation D^n to denote the repeated composition of a dynamic structure D on V with itself: for a non-negative integer n, D^n is the dynamic structure $\langle D \star D \star \cdots \star D \rangle$ (n times D). D^0 equals 1_V^V. Informally, D^n is the dynamic structure that describes the application of all possible sequences of length n of local actions specified by D. Evidently, each such sequence corresponds to a path in $V \times V$ of length n. Observe that D^n is the zero function $D_{\mathbf{0}}^V$ on $V \times V$ if $n > depth(D)$.

We can now use the operations $+$ and $\star$ to define a third operation, *it*, on dynamic structures D on finite sets such that $d(D)$ is acyclic. This operation enables us to express the graph $CS(proc)$ that contains the configurations of a process in a convenient way.

Definition 15. Let V be a finite set. Let D be a dynamic structure on V such that $d(D)$ is acyclic. The *iteration of* D is the dynamic structure

$$
it\ D = \sum_{0 \leqslant n \leqslant depth(D)} D^n.
$$

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

Informally, *it D* is the dynamic structure that describes the application of all possible sequences of arbitrary length of non-zero local actions specified by D. Evidently, each such sequence corresponds to a path in $d(D)$. This is formally expressed by the following lemma.

Lemma 3. Let V be a finite set. Let D be a dynamic structure on V such that $d(D)$ is acyclic. Then, for each $x, y, u, v \in V$,

$$\sum_{p:u\to x \text{ in } d(D)} D_p^v = (it\ D)^v(u,x),$$

$$\sum_{\substack{p:u\to x \text{ in } d(D),\\ q:v\to y \text{ in } d(D)}} D_{pq} = \langle (it\ D)^s(u,x) * (it\ D)^t(v,y)\rangle.$$

Proof. We will first prove, by induction on n, that for each $a \in \{v, s, t\}$ and each integer $n \geqslant 0$,

$$\sum_{\substack{p:u\to x \text{ in } d(D)\\ length(p)=n}} D_p^a = (D^n)^a(u,x).$$

The case $n = 0$ is easy because $D_p^a = \mathbf{1}$ if $length(p) = 0$. For $n \geqslant 1$ we have the following:

$$\begin{aligned}
(D^n)^a(u,x) &= \langle D^{n-1} \star D\rangle^a(u,x)\\
&= \sum_{z\in V} \langle (D^{n-1})^a(u,z) * D^a(z,x)\rangle\\
&= \sum_{\substack{z\in V,\\ (u,z)\in d(D^{n-1}),\\ (z,x)\in d(D)}} \langle (D^{n-1})^a(u,z) * D^a(z,x)\rangle\\
&= \sum_{\substack{z\in V,\\ (z,x)\in d(D)}} \sum_{\substack{p_1:u\to z \text{ in } d(D),\\ length(p_1)=n-1}} \langle D_{p_1}^a * D^a(z,x)\rangle\\
&= \sum_{\substack{p:u\to x \text{ in } d(D),\\ length(p)=n}} D_p^a.
\end{aligned}$$

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

The second equation of the statement of the lemma is obtained as follows.

$$\sum_{\substack{p:u\to x \text{ in } d(D),\\ q:v\to y \text{ in } d(D)}} D_{pq} = \left(\sum_{p:u\to x \text{ in } d(D)} D_p^s\right) * \left(\sum_{q:v\to y \text{ in } d(D)} D_q^t\right)$$

$$= \left(\sum_{0\leqslant m\leqslant depth(D)} \sum_{\substack{p:u\to x \text{ in } d(D),\\ length(p)=m}} D_p^s\right) * \left(\sum_{0\leqslant n\leqslant depth(D)} \sum_{\substack{p:v\to y \text{ in } d(D),\\ length(p)=n}} D_q^t\right)$$

$$= \left(\sum_{0\leqslant m\leqslant depth(D)} (D^m)^s(u,x)\right) * \left(\sum_{0\leqslant n\leqslant depth(D)} (D^n)^t(u,x)\right)$$

$$= \langle (it\ D)^s(u,x) * (it\ D)^t(v,y)\rangle.$$

The proof of the first equation is similar, but simpler. □

The following result states a few properties concerning *it*.

Theorem 6. Let A, B, V, W be sets such that W is finite, $A \subseteq V$ and $B \subseteq W$, let g be a graph on A and let D be a dynamic structure on B.

1 $(it\ D)|^W = 1_B^W \star it(D|^W) = it(D|^W) \star 1_B^W$.
2 If $A \subseteq B$ then $it(D|^W)[g|^V] = (it\ D)[g]|^{V\cup W}$.
3 If $d(D) \cap (A \times B) = \varnothing$ then $(it\ D)[g] = g|^{A\cup B}$.

Proof.

1 Observe that $depth(D) = depth(D|^W)$. Using Theorem 4,

$$1_B^W \star it(D|^W)$$

$$= 1_B^W \star \left(\sum_{0\leqslant n\leqslant depth(D|^W)} (D|^W)^n\right) = 1_B^W \star \left(1_W^W + \sum_{1\leqslant n\leqslant depth(D)} D^n|^W\right)$$

$$= 1_B^W + \sum_{1\leqslant n\leqslant depth(D)} D^n|^W = \left(\sum_{0\leqslant n\leqslant depth(D)} D^n\right)\Bigg|^W$$

$$= (it\ D)|^W.$$

Similarly, $(it\ D)|^W = it(D|^W) \star 1_B^W$.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

2 It follows from 5 of Theorem 3 that

$$it(D|^W)[g|^V] = it(D|^W)[g|^{V\cup W}] = it(D|^W)[g|_A|^{V\cup W}] = (1_A^{V\cup W} \star it(D|^W))[g].$$

By 1 above and by 10 of Theorem 4,

$$1_A^{V\cup W} \star it(D|^W) = 1_A^{V\cup W} \star 1_B^W \star it(D|^W) = 1_A^{V\cup W} \star (it\ D)|^W = 1_A^A \star (it\ D)|^{V\cup W}.$$

Finally, using 5 and 6 of Theorem 3,

$$(1_A^A \star (it\ D)|^{V\cup W})[g] = ((it\ D)|^{V\cup W})[g] = (it\ D)[g]|^{V\cup W}.$$

3 It follows from 5 of Theorem 3 that

$$(it\ D)[g] = (it\ D)[g|_A|^{A\cup B}] = (1_A^{A\cup B} \star (it\ D))[g].$$

If $d(D) \cap (A \times B) = \varnothing$, then $1_A^{A\cup B} \star D^n = D_{\mathbf{0}}^{A\cup B}$, for each $n \geqslant 1$. Thus $1_A^{A\cup B} \star (it\ D) = 1_A^{A\cup B} \star D^0 = 1_A^{A\cup B}$. Furthermore $1_A^{A\cup B}[g] = g|_A|^{A\cup B} = g|^{A\cup B}$. □

The next result expresses the fact that if two processes can be sequentially composed, then applying all local actions specified by the dynamic structure of the composed process is the same as first applying all local actions specified by the dynamic structure of the first one, followed by all local actions specified by the dynamic structure of the second one.

Theorem 7. Let $proc_1 = (V_1, C_1, D_1, S_1)$ and $proc_2 = (V_2, C_2, D_2, S_2)$ be processes such that $V_1 \cap V_2 = Min(proc_2) \cap Max(proc_1)$. Let $V = V_1 \cup V_2$. Then

$$it(D_1 + D_2) = \langle it\ (D_1|^V) \star it\ (D_2|^V)\rangle.$$

Proof. Since $V_1 \cap V_2 = Min(proc_2) \cap Max(proc_1)$, $\langle D_2|^V \star D_1|^V\rangle = D_{\mathbf{0}}^V$. Hence using the distributivity of $\star$ with respect to $+$ (8 and 9 of Theorem 4),

$$\begin{aligned}
it(D_1 + D_2) &= it(D_1|^V + D_2|^V) \\
&= \sum_{0\leqslant n\leqslant depth(D_1|^V + D_2|^V)} (D_1|^V + D_2|^V)^n \\
&= \sum_{\substack{0\leqslant i\leqslant depth(D_1|^V),\\ 0\leqslant j\leqslant depth(D_2|^V)}} \langle (D_1|^V)^i \star (D_2|^V)^j\rangle \\
&= \left\langle \left(\sum_{0\leqslant i\leqslant depth(D_1|^V)} (D_1|^V)^i \right) \star \left(\sum_{0\leqslant j\leqslant depth(D_2|^V)} (D_2|^V)^j \right) \right\rangle \\
&= \langle it\ (D_1|^V) \star it\ (D_2|^V)\rangle.
\end{aligned}$$

□

The next result states that the computed structure of a process (V, C, D, S) can be expressed using *it D*. This enables us to use the equations derived for dynamic structures when reasoning about configurations. Since the computed structures are descriptions of

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

rewriting histories, this provides a powerful tool for the analysis of the behaviour of a graph transformation system.

Theorem 8. Let $proc = (V, C, D, S)$ be a process. Then

$$CS(proc) = (it\ D)[S].$$

Proof. Using Lemma 3, the desired result is obtained as follows. Let x, y be elements of V. Then

$$\begin{aligned} CS(proc)^e(x,y) &= \sum_{u,v \in V} \sum_{\substack{p:u \to x \text{ in } d(D),\\ q:v \to y \text{ in } d(D)}} D_{pq}(S^e(u,v)) \\ &= \sum_{u,v \in V} \langle (it\ D)^s(u,x) * (it\ D)^t(v,y) \rangle (S^e(u,v)) \\ &= ((it\ D)[S])^e(x,y). \end{aligned}$$

It can be shown in a similar way that $CS(proc)^v = ((it\ D)[S])^v$. □

Now we can give an alternative proof for Theorem 2. It demonstrates how the algebraic properties of dynamic structures can be used to get an equational proof, based on the equation of Theorem 8.

Theorem 9. Let $proc_1 = (V_1, C_1, D_1, S_1)$ and $proc_2 = (V_2, C_2, D_2, S_2)$ be processes such that $V_1 \cap V_2 = Min(proc_2) \cap Max(proc_1)$. Then

$$Res(Res(proc_1) \oplus proc_2) = Res(proc_1 \oplus proc_2).$$

Proof. Let $proc = proc_1 \oplus proc_2 = (V, C, D, S)$, $K = Max(proc_1)$, $k = Res(proc_1)$, $M = Max(proc)$, and $proc_3 = (k \oplus proc_2) = (V_3, C_3, D_3, S_3)$. Then $M = Max(proc_3)$, $V_3 = K \cup V_2$, $C_3 = C_2$, $D_3|^V = D_2|^V$ and $S_3 = k + S_2$. We have to show that $Res(proc) = Res(proc_3)$. By Theorem 8 and Theorem 3,

$$\begin{aligned} Res(proc) &= ((it\ D)[S])|_M &&= ((it\ D)[S_1])|_M + ((it\ D)[S_2])|_M \\ Res(proc_3) &= ((it\ D_3)[S_3])|_M &&= ((it\ D_3)[k])|_M + ((it\ D_3)[S_2])|_M \end{aligned}$$

Thus it suffices to show that

(1) $((it\ D)[S_1])|_M = ((it\ D_3)[k])|_M$, and

(2) $((it\ D)[S_2])|_M = ((it\ D_3)[S_2])|_M$.

(1) Using 5 of Theorem 3 and the fact that $M \subseteq V_3$,

$$(it\ D)[S_1]|_M = (it\ D)[S_1|_{V_1}|^V]|_{V_3}|^V|_M = (1^V_{V_1} \star (it\ D) \star 1^V_{V_3})[S_1]|_M.$$

Using Theorems 4, 6, and 7 and the fact that $D_2|^V = D_3|^V$ and $V_1 \cap V_3 = K$,

$$\begin{aligned} 1^V_{V_1} \star (it\ D) \star 1^V_{V_3} &= 1^V_{V_1} \star it(D_1|^V) \star it(D_2|^V) \star 1^V_{V_3} \\ &= (it(D_1|^V) \star 1^V_{V_1}) \star (1^V_{V_3} \star it(D_3|^V)) \\ &= it(D_1|^V) \star 1^V_K \star it(D_3|^V) \end{aligned}$$

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

It follows from Theorems 5 and 6, and from 5 of Theorem 3 that

$$(it(D_1|^V) \star 1_K^V)[S_1] = 1_K^V[(it\ D_1)[S_1]]|^V = (it\ D_1)[S_1]|^V|_K|^V = k|^V,$$

and hence, by Theorem 6, we can conclude that

$$(it\ D)[S_1]|_M = it(D_3|^V)[k|^V]|_M = (it\ D_3)[k]|_M.$$

(2) On the other hand, using Theorems 5, 6, and 7,

$$(it\ D)[S_2]|_M = (it(D_1|^V) \star it(D_3|^V))[S_2]|_M = it(D_3|^V)[S_2|^V]|_M = (it\ D_3)[S_2]|_M. \quad \square$$

Conclusion

LAS processes provide a way to describe the concurrent behaviour of graph rewriting systems, in line with other process notions developed in concurrency theory. First we have shown that a number of simple operations may be defined on dynamic structures (a dynamic structure is the component of a process that describes the local actions) and that the graph $CS(proc)$ containing all configurations of a process may be expressed in terms of these operations. Then we derived a number of equational laws for expressions containing these operations, and, finally, we illustrated their use by presenting an equational proof of the coherence theorem, which expresses a relation between a structural property of processes (being the sequential composition of subprocesses) and their configurations (in particular, their result graphs).

LAS processes differ from the processes introduced in Corradini *et al.* (1996) not only because they deal with embedding based rewriting, but also because they are defined on the level of the alphabets involved, independent of the choice of a particular set of productions. This has the advantage that one may investigate the interaction of the processes of a given system with those of an arbitrary 'context' system, and then investigate semantic equivalences based on that interaction (for example, systems may be considered equivalent if no context can distinguish between them).

On a more technical level, the process notion presented in this paper is a very concrete one: processes with the same structure, but built on different sets of nodes, are distinguished (for example, the choice of concrete nodes determines whether two processes are sequential or not in the sense of Definition 5). This is not a problem if one is interested in the relationship between the configurations of a process, but it may be a disadvantage if one wants to define the abstract semantics of a system. Therefore, it may be useful to develop a more abstract presentation as developed in Baldan *et al.* (1998).

References

Baldan, P., Corradini, A. and Montanari, U. (1998) Concatenable Graph Processes: Relating Processes and Derivation Traces. In: Automata, Languages and Programming '98. *Springer-Verlag Lecture Notes in Computer Science* **1443** 283–295.

Billaud, M., Lafon, P., Métivier, Y. and Sopena, E. (1989) Graph Rewriting systems with priorities. In: 15th Workshop on Graphs '89. *Springer-Verlag Lecture Notes in Computer Science* **411** 94–106.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press

Corradini, A. (1996) Concurrent Graph and Term Graph Rewriting. In: CONCUR'96. *Springer-Verlag Lecture Notes in Computer Science* **1119** 438–464.

Corradini, A., Montanari, U. and Rossi, F. (1996) Graph Processes. *Fundamenta Informaticae* **26** 241–265.

Ehrig, H., Engels, G., Kroewski, H.-J. and Rozenberg, G. (eds.) (2000) Theory and Applications of Graph Transformations '98. *Springer-Verlag Lecture Notes in Computer Science* **1764**.

Engelfriet, J. and Rozenberg, G. (1997) Node Replacement Graph Grammars. In: *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific 1–94.

Goltz, U. and Reisig, W. (1983) The Non-sequential Behaviour of Petri Nets. *Information and Control* **57** 125–147.

Janssens, D. (1994) ESM Systems and the Composition of Their Computations. In: Graph Transformations in Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **776** 203–217.

Janssens, D. (1999) Actor Grammars and Local Actions. In: *Handbook of Graph Grammars and Computing by Graph Transformation* **3**, World Scientific.

Janssens, D. and Mens, T. (1996) Abstract semantics for ESM systems. *Fundamenta Informaticae* **26** 315–339.

Janssens, D. and Rozenberg, G. (1989) Actor grammars. *Math. Systems Theory* **22** 75–107.

Pratt, V. R. (1987) Modelling Concurrency with Partial Orders. *J. of Parallel Programming* **15** 33–71.

Rozenberg, G. (ed.) (1997) *Handbook of Graph Grammars and Computing by Graph Transformation* **1**, World Scientific.

Sassone, V. (2000) On the Algebraic Structure of Petri Nets. *Bulletin of the EATCS* **72**, 133–148.

Verlinden, N. and Janssens, D. (2000) A Framework for NLC and ESM: Local Action Systems. In: Theory and Application of Graph Transformations. *Springer-Verlag Lecture Notes in Computer Science* **1764** 194–214.

Winskel, G. (1980) *Events in Computation*, Ph. D. Thesis, Comp. Sci. Dept., University of Edinburgh.

https://doi.org/10.1017/S096012950100353X Published online by Cambridge University Press