GRAPH GRAMMARS FOR DISTRIBUTED SYSTEMS

by Ilaria Castellani and Ugo Montanari
Istituto di Scienze dell'Informazione
University of Pisa, Italy

Abstract

In the paper we define grammars on a class of labeled, partially ordered hypergraphs, called distributed systems. A distributed system models both the spatial and the temporal aspects of a real system through the relations of adjacency and causality. Terminal symbols represent the (deterministic, certain) past history of the system while nonterminal symbols model the (possibly nondeterministic, potential) future history of the system. The (context free) productions of a grammar represent the possible stand-alone evolutions of system components. From the productions, we obtain a (possibly infinite) number of rewriting rules, which model the synchronized evolution of adjacent system components. The (terminal) distributed systems derived within a given grammar represent the alternative deterministic, concurrent computations of a single nondeterministic system which is thus completely modeled by the grammar.

1. Introduction

Many models exist in the literature for describing the behaviour of concurrent programs. Among others, we may mention Petri nets /5/, Hoare CSP /4,6/, Winskel event structures /7,8/, Milner synchronization trees and synchronous and asynchronous CCS /2,3/ and Winkowski concurrent systems /9/. The authors have recently defined a model called Labeled Event Structures (LES) inspired by nets, synchronization trees and event structures /10,11/.

The LES model is characterized by two aspects. First, it represents a nondeterministic, concurrent computation as a (possibly infinite) partial order of events. More precisely, since branches corresponding to different nondeterministic choices never join again, a computation has the gross structure of a "thick" tree. Second, the way a concurrent system is interfaced with the external world is carefully considered, and the notion of concurrent observer is defined.

On many of the above mentioned models, a number of operations is defined, giving the semantics of basic language constructs. The starting point of the present work was the observation that essentially all those operations can be substituted by a single, rather standard type of graph rewriting rule . Grammars based on such rules provide in our view a clear operational model of the behaviour of concurrent communicating systems. In a later paper we plan to show that the semantics of a grammar could also be expressed by a LES.

2. Distributed systems

In this section we introduce our notion of distributed system. As it is the case for string grammars, the alphabet is partitioned into a terminal and a nonterminal alphabet. Furthermore, every symbol has an associated n-arity or rank, since it represents a subsystem with a fixed number of interaction points with the external world. Finally, a terminal symbol of rank k is a k-tuple of symbols of a primitive alphabet. In fact a k-ary terminal symbol represents the synchronized occurrence of k events on k interaction points. Formally we have:

T_1, the primitive terminal alphabet.

$T_k=T_1^k$, the alphabet of the terminal symbols of rank k, k=1,2,... .

$T=\bigcup_{k=1}^{\infty} T_k$, the terminal ranked alphabet.

N_k, k=1,2,..., the primitive, disjoint alphabets of the nonterminal symbols of rank k.

$N=\bigcup_{k=1}^{\infty} N_k$, the nonterminal ranked alphabet.

$V_k=T_k+N_k$, the alphabet of symbols of rank k.

$V=T+N=\bigcup_{k=1}^{\infty} V_k$, the ranked (terminal and nonterminal) alphabet.

In our model we will use the concept of hypergraph. We define a hypergraph as a triple (P,S,f) where P is a set of nodes, S is a set of hyperarcs and f is a connection function

$$f:\ S \longrightarrow \bigcup_{k=1}^{\infty} P^k$$

which assigns to every hyperarc a k-tuple specifying the nodes to which it is connected. The value k is called the rank of the hyperarc. Two hyperarcs sharing one or more nodes are called adjacent. Notice that our hypergraphs are the generalization of directed graphs with multiple parallel arcs. In Fig. 1 we see an example of a hypergraph with two hyperarcs of rank one, two of rank two and one of rank three. Notice that there is a self-adjacent hyperarc. On hyperarcs with rank larger than one, the ordering among the connected nodes will be indicated with numbers 0,1,... whenever necessary.

The next step is to define our notion of distributed system as a hypergraph, whose hyperarcs are labeled on V and are partially ordered. The meaning of the nodes or ports, is the places in space where the various parts of the system interact. The terminal hyperarcs represent elementary events having no extension in time (they are instantaneous) but possibly extension in space (they may be connected to more than one node). The nonterminal hyperarcs represent subsystems having possibly both temporal and spatial extension. The partial ordering represents the temporal or causal dependency relation among subsystems.

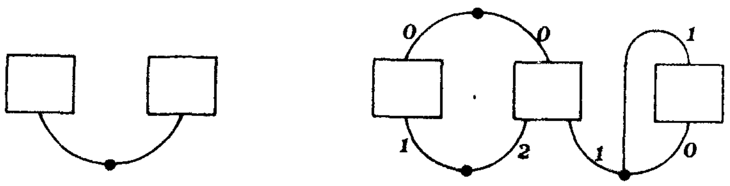

Fig. 1 A hypergraph

Formally, a distributed system on an alphabet V: D= $(P,S,f,l,\leq)$ comprises the following.

i) A hypergraph (P,S,f). The nodes P and the hyperarcs S may be also called ports and subsystems.

ii) A labeling function l: $S \rightarrow V$ such that if l(s)=x then the hyperarc s and the symbol x have the same rank. A subsystem labeled with a terminal symbol may be called an event. Events must not be self-adjacent.

iii) A partial ordering $\leq$ on S called temporal or causal relation. Two subsystems s_1 and s_2 such that either $s_1 \leq s_2$ or $s_2 \leq s_1$ are called causally related. Two subsystems which are not causally related are concurrent. We require the set of all events (i.e. terminal subsystems) to be left closed, namely that all predecessors of an event are events. Furthermore, an event cannot be concurrent with an adjacent subsystem (terminal or not).

The two restrictions in iii) have an intuitive explanation. In fact a hypergraph in our model intends to represent a system at some stage of its evolution. The terminal hyperarcs have the meaning of events which have already happened at that stage, namely they describe the (deterministic, certain) past history of the system. Similarly, the nonterminal hyperarcs describe the (possibly nondeterministic, potential) future history of the system. Thus a non terminal hyperarc cannot precede a terminal hyperarc. Furthermore, two concurrent subsystems are meant to be possibly overlapping in time and two adjacent subsystems are meant to be possibly overlapping in space. But at some point in time and space only one event may happen.Therefore a terminal hyperarc cannot be adjacent and concurrent to another terminal hyperarc. In fact the two events either are incompatible (i.e. they represent different activities on the common ports) or they are compatible and synchronized and thus they should be merged in a single event. Similarly, a terminal hyperarc cannot be adjacent and concurrent to a nonterminal hyperarc. In fact the event would impose a precise consistency constraint to the (still unexpanded) nonterminal subsystem, which possibly could not be met. But in this case the event itself could not have happened. Thus the nonterminal hyperarc should have been consistently expanded at the same stage of the graph generation process at which the terminal hyperarc was produced. Notice that as a consequence of the combination of the two constraints above, a terminal hyperarc must precede an adjacent nonterminal hyperarc.

The following definition will be useful later. A distributed system $D_1=(P_1,S_1, f_1,l_1, \leq_1)$ is called a subgraph of a distributed system $D=(P,S,f,l,\leq)$ on the same alphabet iff there exist two injective mappings $h_P: P_1 \longrightarrow P$ and $h_S: S_1 \longrightarrow S$ such that for all $s_1,s_1' \in S_1$ we have $h_P(f_1(s_1))=f(h_S(s_1))$, $l_1(s_1)=l(h_S(s_1))$ and $s_1 \leq_1 s_1' = h_S(s_1) \leq h_S(s_1')$. The part of D in correspondence to D_1 is called an occurrence of D_1 in D.

In Fig. 2 we show an example of a distributed system. To graphically represent our distributed systems, we use the two dimensions of the page to express time (vertical, flowing from top to bottom) and space (horizontal). Thus nodes, which have only a temporal extension, are represented usually as vertical lines (of medium thickness). Terminal (nonterminal) subsystems are expressed by circles (boxes) and the connection function is described by horizontal thin lines, decorated when necessary with a natural number giving the ordering of the connection. The intersection with the node is marked with a dot. Lower (upper) case symbols in the circles (boxes) represent the labeling function l. Remember that terminal symbols of rank k are k-tuples of symbols of the primitive alphabet T_1. Thus in the circles we draw the primitive symbols near the corresponding connections. Nonterminal symbols of rank 1 are expressed with $A_0,A_1,\ldots$, of rank 2 with $B_0,B_1,\ldots$ and so on. Finally the causal partial ordering is represented by its Hasse diagram, drawn from top to bottom with thick, nonhorizontal lines.
In our example we have:

$D=(P,S,f,l,\leq)$, with

$P=\{n_0,n_1,n_2\}$

$S=\{e_0,\ldots,e_3,s_0,\ldots,s_2\}$

$f(e_0)=(n_0,n_1,),f(e_1)=(n_1,n_2),f(e_2)=(n_0),f(e_3)=(n_2),f(s_0)=(n_1,n_0),f(s_1)=(n_1,n_2)$

$f(s_2)=(n_1)$

$l(e_0)=(a,b),l(e_1)=(c,b),l(e_2)=(a),l(e_3)=c,l(s_0)=l(s_1)=B_0,l(s_2)=A_0$

$\leq$ is the reflexive and transitive closure of the following relation: $e_0\leq e_1,e_2$;

$e_1\leq e_3,s_0$; $e_2\leq s_0$; $e_3\leq s_1$; $s_1\leq s_2$.

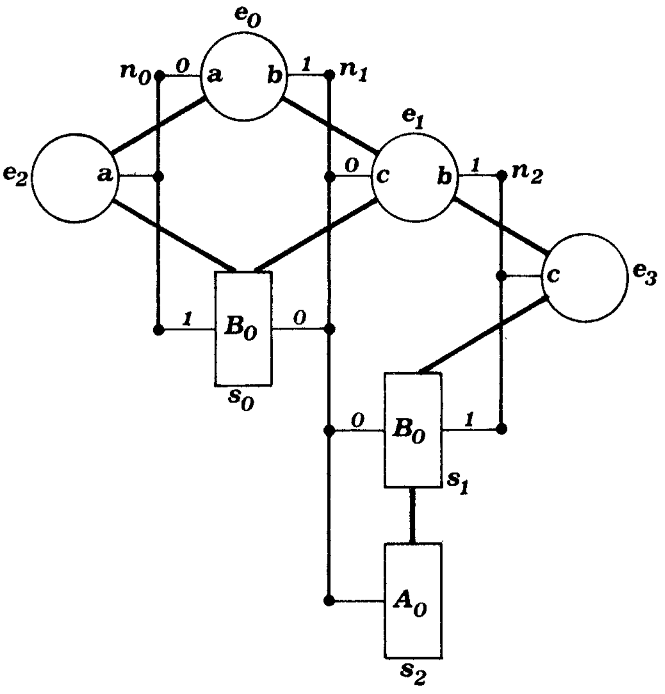

Fig. 2 A distributed system

Notice that subsystems s_0 and e_3 are concurrent but not adjacent, e_0 and e_2 are adjacent but causally related, while s_0 and s_1 are adjacent and concurrent but both nonterminal.

3. Productions

In this section we give our definition of productions. They are essentially context-free and can be applied to hyperarcs. Thus they describe the evolution of subsystems as isolated entities, corresponding in this sense to the first step of the methodology discussed in /1/.

Formally, a production P of rank k on the alphabet V

$X^k \longrightarrow (D,(n_1,\ldots,n_k))$

is a pair, whose first (left) element is a nonterminal of rank k and whose second (right) element is a pair $(D,(n_1,\ldots,n_k))$ where $D=(P,S,f,l,\leq)$ is a distributed system on V and where $(n_1,\ldots,n_k)$ is a k-tuple of distinct nodes of D. We require that D is rooted, namely that S contains exactly one terminal hyperarc and that this hyperarc is smaller than any other hyperarc of S in the causal relation $\leq$. Nodes in $(n_1,\ldots n_k)$ are called global or external, and the other nodes of P are called local or internal. We impose the above restrictions since we want to have a direct correspondence between a generated event and the application of a rule. Thus two events generated by the same rule could only be contemporary and thus should be identified. Similarly, if an event e and a (nonadjacent) nonterminal subsystem s are generated by a production P, the nonterminal s can be expanded obviously only after the application of P (i.e. its creation) and thus should be larger than e in the temporal ordering.

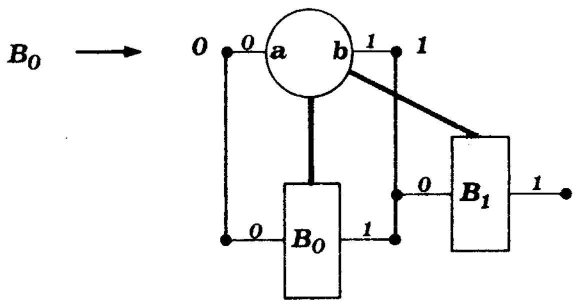

Fig. 3 A production

In Fig. 3 we show an example of a production of rank two. We assume that nodes belonging to the k-tuple in the right member are labeled with natural numbers 0,1,...,k-1.

Intuitively, a production $X^k \longrightarrow (D,(n_1,\dots,n_k))$ can be applied to a graph by replacing any nonterminal subsystem labeled with X^k with graph D. But if we look for instance at the graph in Fig. 2, we see that the production in Fig. 3 cannot be applied to subsystem s_0.In fact the generated event would result, in the obtained graph, both adjacent and concurrent to subsystem s_1. Thus also s_1 should be replaced at the same time.

4. Rewriting rules

To express the synchronized application of many productions, we proceed as follows. We first explain how a rewriting rule can be derived from one or more (consistent) productions, and then we define how a rewriting rule can be applied to a distributed system. This corresponds to the second step of the methodology /1/. Informally speaking, a rewriting rule has a left member specifying how the various productions should be synchronized and a right member which is obtained by merging the right members of the productions.

More formally, a rewriting rule r is a pair

$D_1 \longrightarrow (D_2,g,R)$

whose first (left) element is a distributed system $D_1=(P_1,S_1,f_1,l_1,\leq_1)$. We require that all subsystems of D_1 are nonterminal and pairwise concurrent.

The right member (D_2,g,R) is a triple, where:

$D_2=(P_2,S_2,f_2,l_2,\leq_2)$ is a rooted distributed system; let e be the unique event in S_2;

$g:P_1 \longrightarrow P_2$ is an injective function called spatial embedding function;

$R \subseteq S_1 \times S_2$ is the temporal embedding relation. We require that R is invariant with respect to right composition with $\geq_2$, namely s R x and $x' \leq_2 x$ implies s R x'. Furthermore for all s in S_1 we must have s R e.

The purpose of g and R is to specify how, when applying the rewriting rule, its right part can be inserted in the original hypergraph. We need a definition which will be useful later. A node $n \in P_1$ is called a synchronization node iff g(n) is connected with event e.

Now we describe a procedure which, given the left member D_1 of a rewriting rule and a production for every subsystem in it, derives, if possible, the right member (D_2,g,R) of the rule. We proceed as follows.

Procedure A

Step 1 We obtain a first approximation $(D'_2=(P'_2,S'_2,f'_2,l'_2,\leq'_2),g',R')$ of the right member by executing, for every subsystem s of D_1, the following operations. Initially D'_2 is D_1, g' is the identity and R' is empty.
Let $X^k \xrightarrow{} (D,(n_1,\dots,n_k))$ be the production we want to use for subsystem s. We must have $l_1(s)=X^k$. Let $f_1(s)=(n'_1,\dots n'_k)$. We erase s from D'_2, we add D to D'_2 and we merge nodes n_i and n'_i, i=1,...,k (*).
Furthermore, we let s R'x for all subsystems x in D.

Step 2 We check if D'_2 is event-connected, namely if for every pair of events e',e" of D'_2 there exists a sequence of events $e'=e_1,e_2,\dots,e_n=e''$ with e_i adjacent to e_{i+1}, i=1,...,n-1. If D'_2 is not event-connected, we do not allow the derivation of a single rewriting rule, since not all the productions in this case would be naturally synchronized. Otherwise, a compatibility test is performed. For every pair of (possibly coincident) adjacent events e_1 and e_2 of D'_2, let n be any common node, say the i-th for e_1 and the j-th for e_2. Thus $f'(e_1)|_i = f'(e_2)|_j = n$, where $|_k$ is the tuple projection operation. We must have $l'(e_1)|_i=l'(e_2)|_j$. If the test is not satisfied, the rewriting rule cannot be derived since the given rules were incompatible.

Step 3 The final right member (D_2,g,R) is obtained as follows. D_2 has the same set of nodes of D'_2. All the events in D'_2 are merged in one single event $\bar{e}$. Its rank k is thus equal to the number of nodes in D'_2 which at least an event was connected to. These nodes form the k-tuple $f_2(\bar{e})$, but their ordering in it is arbitrary, and thus this procedure usually generates more than one rewriting rule. To define the label of $\bar{e}$, let n be any node in the k-tuple above and let e be any event of D'_2 connected to node n. Say $f'(e)|_j=f(\bar{e})|_i$ =n. We let $l(\bar{e})|_i=l'(e)|_j$. Notice that the choice of e is inessential, since the test in step 2 was satisfied. The nonterminal subsystems of D_2 and their labels are the same as in D'_2. Function f is the same, and relations $\leq$ and R are the same as in D'_2 except for the merging above.

(*) Formally, adding D to D'_2 means componentwise union (functions and relations are considered sets of pairs). Merging two nodes means to replace each of the nodes with a single new node in all sets, functions and relations.

Formally, given a set of productions P and a rewriting rule r, we say that r is derivable from P iff for every subsystem in the left member of r there exists a production in P such that the right member of r can be generated with the above procedure A.

To exemplify the application of procedure A let us consider as D_1 the left member in Fig. 4. We want to apply the production in Fig. 3 to both s_0 and s_1. The right member shown in Fig. 4 is the result of step 1 of the procedure above. We label nodes and hyperarcs related by function g and relation R respectively with the same marks $n_0, n_1, \ldots$ and $s_0, s_1, \ldots$ in the right and left members. Since R is invariant with respect to composition with $\geq$, we mark in the right member only the maximal elements of the ordering $\leq$.

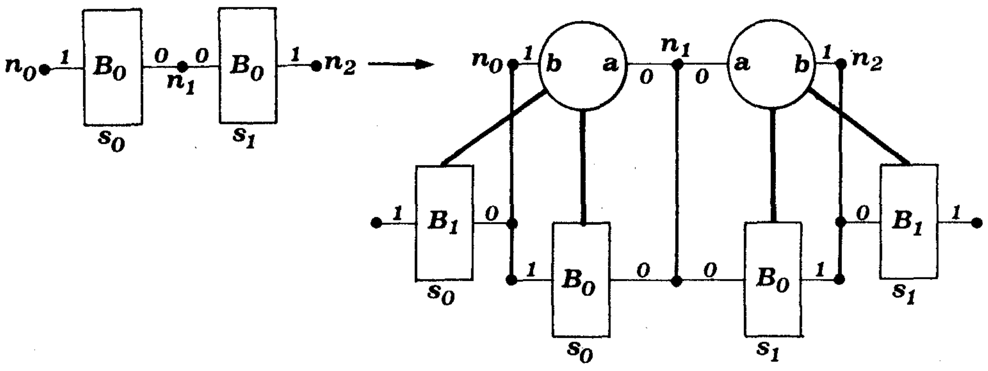

Fig. 4 An intermediate step in the derivation of a rewriting rule

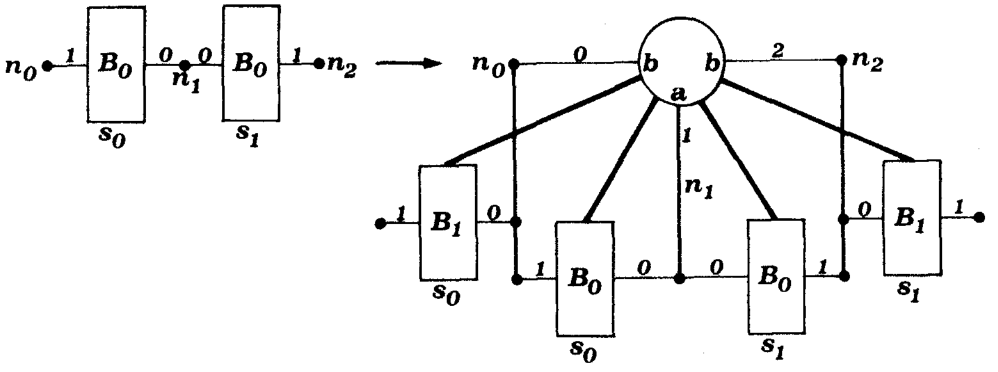

Fig. 5 A rewriting rule derived from the synchronization of two copies of the production in Fig. 3

The test in step 2 is satisfied since here we have only two adjacent events and they are compatible, because both of them assign the same character a to the common node marked n_1. Thus we can pass to step 3 of our procedure, whose result is the rewriting rule shown in Fig. 5. Here the two events of rank 2 are merged in one event of rank 3. Notice that the alternative rewriting rules which can be generated by step 3 would differ from the rule in Fig. 4 only by a permutation of numbers 0,1 and 2 specifying the connection ordering of the event. The association of symbols b, a and b to nodes marked n_0, n_1 and n_2 would not change instead.

Our definition of rewriting rule is in a sense more general than necessary, since not every rewriting rule can be obtained by the synchronization of suitable productions. For instance, the rewriting rule in Fig. 6 has a nonterminal subsystem in its right member which is related with R to two subsystems in the left member (marked with s_0 and s_1). It is easy to see that this is never the case for rewriting rules derived from productions. This property has an intuitive meaning. In fact, rewriting rules are in general intrinsically context sensitive, since they allow to recombine subsystems together (for instance the two subsystems labeled B_0 can be replaced by the subsystem labeled C_0 in the rule in Fig. 6), while productions permit only the decomposition of subsystems.

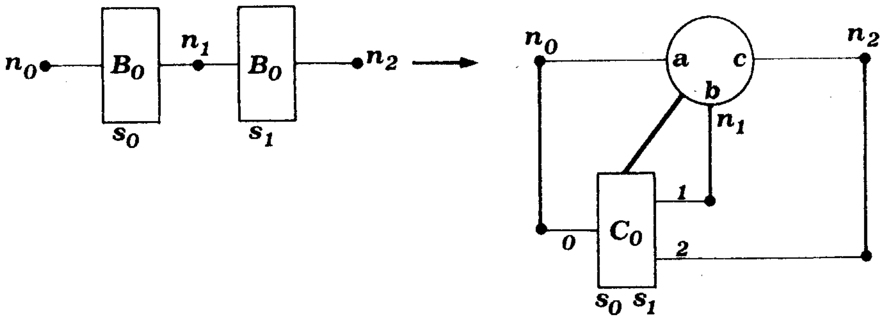

Fig. 6
A rewriting rule which cannot be derived by the synchronization of any production.

5. Applying rewriting rules

In this section we define a procedure B, which, given a distributed system $D=(P,S,f,l,\leq)$, a rewriting rule $r= D_1 \longrightarrow (D_2,g,R)$ and an occurrence of D_1 in D derives, if possible, a new distributed system D'. For simplicity we identify, in the following, system D_1 with its occurrence in D.

Procedure B

Step 1 In this step we evaluate two applicability conditions. The first condition imposes that no nonterminal subsystem esists in D which is smaller in the temporal ordering than any subsystem of D_1. The second condition requires that no subsystem of D which is not in D_1, and which is concurrent with all the subsystems in D_1 is connected to a synchronization node of D_1. This second condition makes sure that the rule synchronizes enough subsystems of D.

Step 2 In this step we operate the replacement of D_1 with D_2 in D. We first merge the pairs of nodes of D_1 and D_2 belonging to function g. Then we increase relation $\leq$ as follows. For every (terminal) subsystem s of D smaller than some subsystem of D_1, and for all subsystems s_2 of D_2, we let $s \leq s_2$; furthermore, for every (nonterminal) subsystem s of D larger than some subsystem s_1 of D_1, and for all subsystems s_2 of D_2 such that $s_1 R\ s_2$, we let $s_2 \leq s$. Finally we erase the subsystems of D_1. The resulting system is D'.

Formally, given two distributed systems D and D' and a rewriting rule $r= D_1 \longrightarrow (D_2,g,R)$ we write $D \xrightarrow{r} D'$ iff there exists an occurrence D_1 in D such that D' can be derived with the procedure B above.

As an example we apply the rewriting rule in Fig. 5 to the graph in Fig. 2. The resulting graph is shown in Fig. 7. Notice than the rule can be applied to the subsystems s_0 and s_1 since the only smaller subsystems are terminal and the only other nonterminal that is connected to either n_0 or n_1 or n_2 (i.e. to some synchronization node) is s_2, which however is larger than s_1. Notice that after the replacement, subsystem s_2 becomes larger than only those subsystems which are generated by its former predecessor s_1.

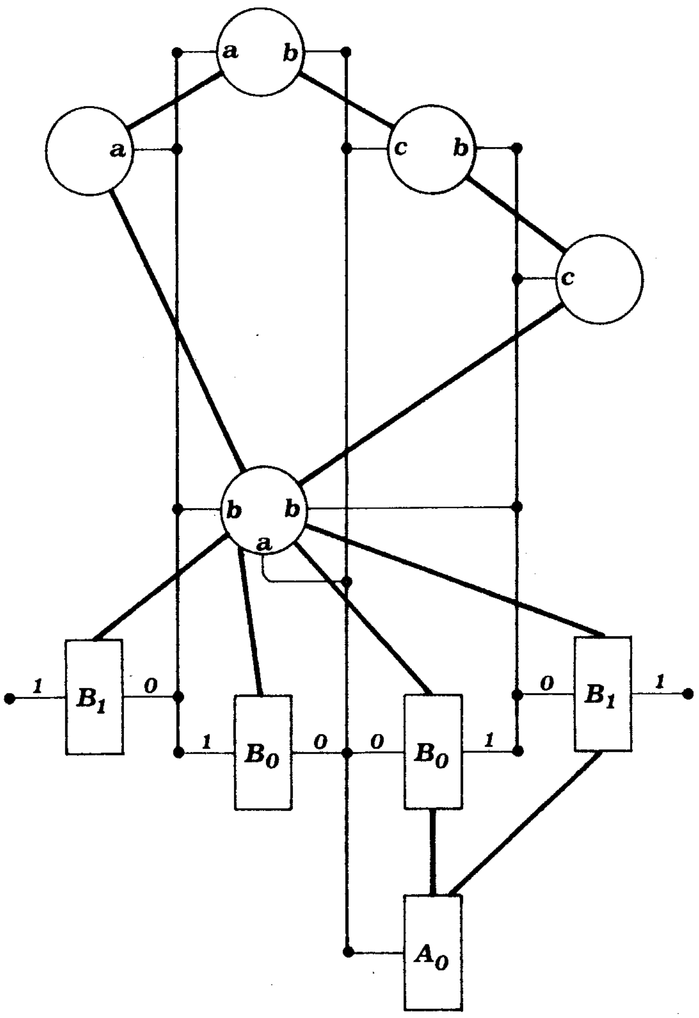

Fig. 7

A distributed system obtained by applying the rewriting rule in Fig. 5 to the system in Fig. 2.

6. Grammars for distributed systems

A grammar G is a triple (V,D_0,P), where V is the alphabet, D_0 is an initial distributed system and P is a finite set of productions. Given two distributed systems D and D' on alphabet V, we write $D \xrightarrow{G} D'$ iff there exists a rewriting rule r derivable from P such that $D \xrightarrow{r} D'$. A derivation or computation for G is a (finite or infinite) sequence D_0, D_1, ... such that $D_i \xrightarrow{G} D_{i+1}$, $i=0,1\ldots$. The language L=L(G) generated by grammar G is the set of all distributed systems D with only terminal subsystems (shortly terminal systems) such that $D_0 \xrightarrow{G} {}^* D$, i.e. derivable by a finite computation.

In Fig. 8a, 8b and 8c we see an example of a grammar, and in Fig. 9 we see an intermediate step in a derivation. The example describes a tree-like synchronous communication network connecting a number of stations. The system can increase its size dynamically. Every station decides autonomously either to receive a message or to send a message named a or a message named b. A sender-receiver pair chosen at random among all the stations which are ready is allowed to communicate. The communication itself is represented as a single event involving all nodes on the path between the sender and the receiver. The example wants to show how a synchronization policy based on pairwise synchronization (like Milner's composition /2/) can be easily represented in our model.

To describe the example, let us consider first the initial system in Fig. 8c. There are two concurrent and adjacent subsystems. The rightmost subsystem labeled Astop can only evolve in a termination event labeled stop (see the production for Astop in Fig. 8a). Subsystem labeled A_0 can also evolve in a stop event (see the last production for A_0), and thus a single stop event is a distributed system generated by the grammar. Alternatively, a subsystem labeled A_0 can reproduce two copies of itself connected by an element of the communication network labeled C (see the first production for A_0). We call this triplet a cell of the system. Notice that subsystem C is the only one connected to the outside world, while the two subsystems labeled A_0 are both connected to newly generated nodes. Thus iterating the application of this production (which does not need any synchronization) we can generate a binary tree of communication elements, whose leaves are A_0 -subsystems. The remaining productions for A_0 correspond to the decisions of either receiving or sending a character (here an a or a b) and in this last case which character to send. Notice that all these productions create an event (modeling the

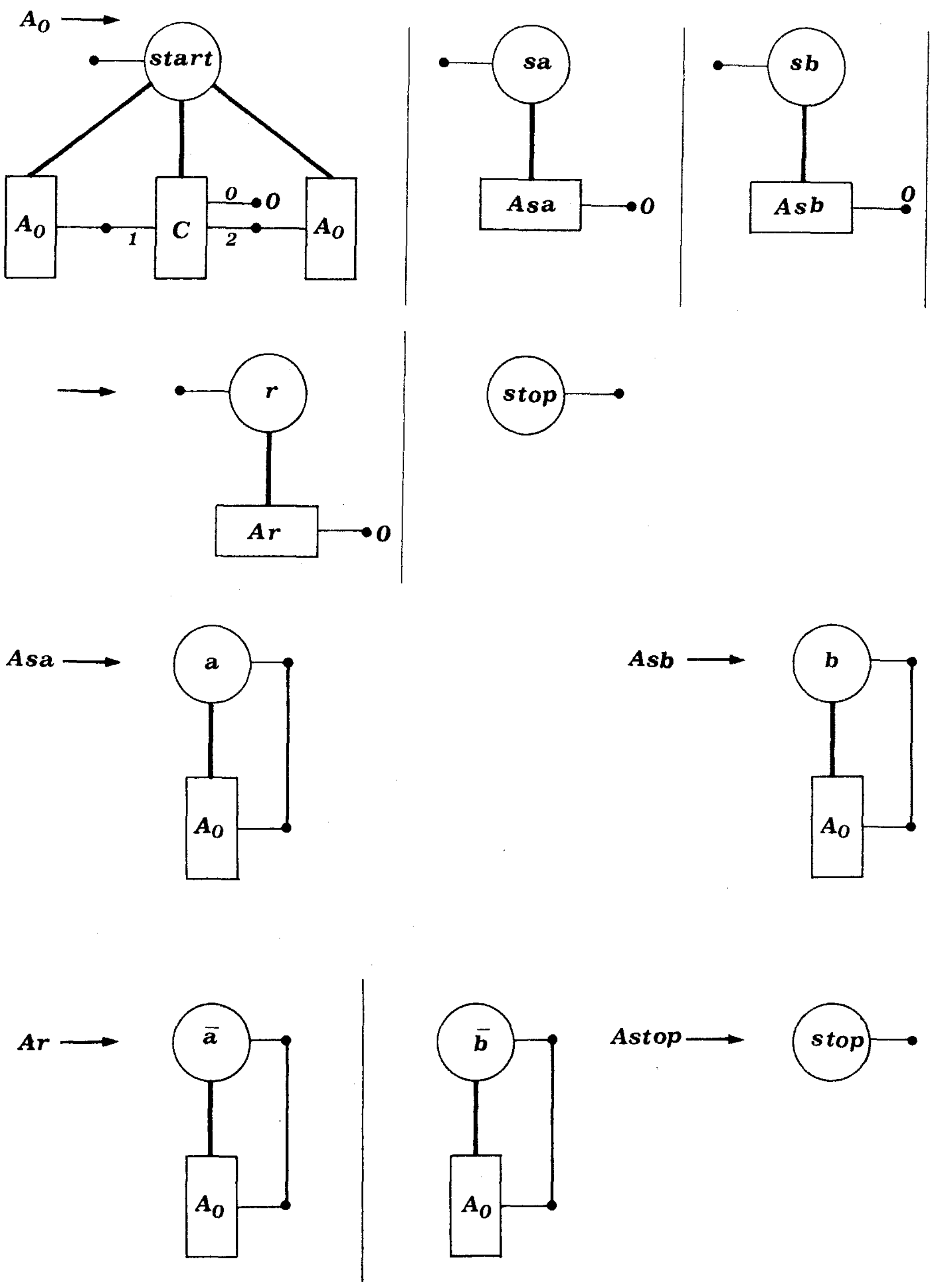

Fig. 8a A set of productions for synchronous communication.

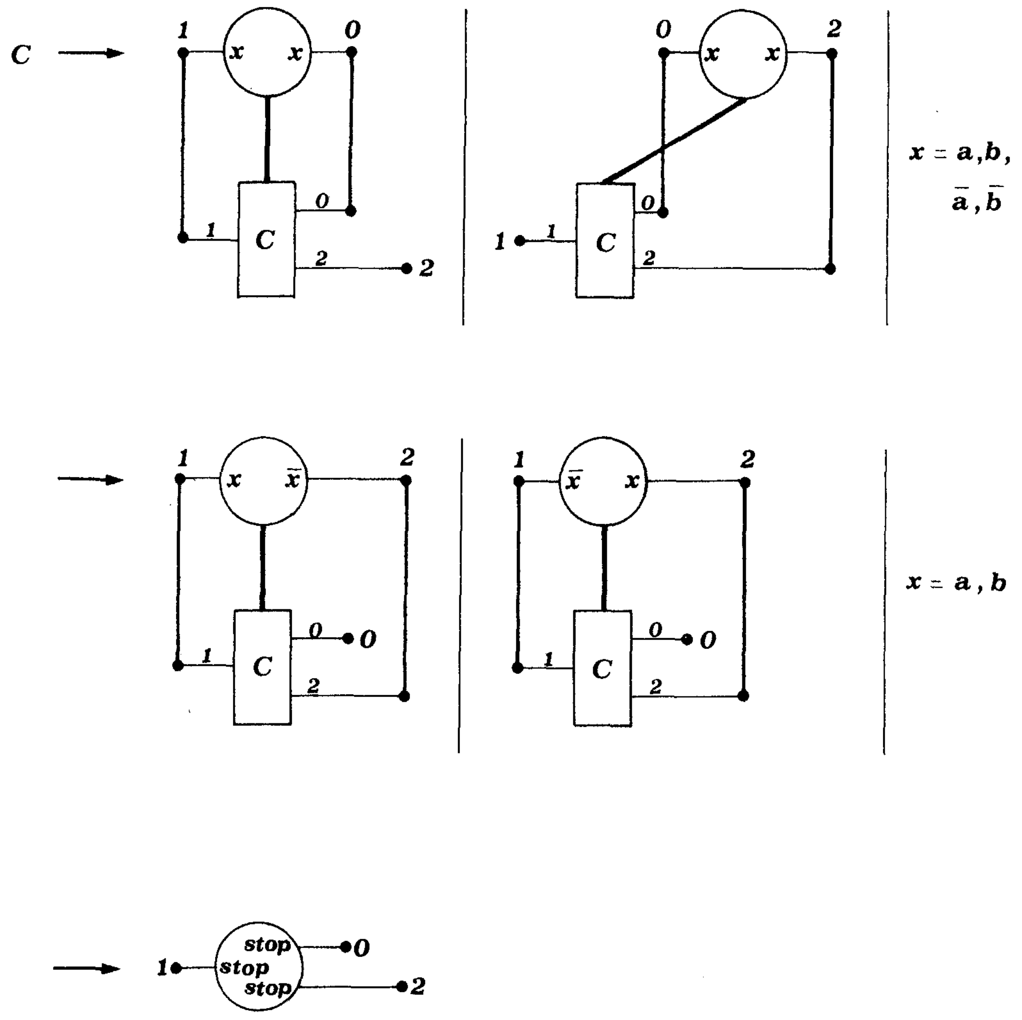

Fig. 8b A set of productions for synchronous communication.

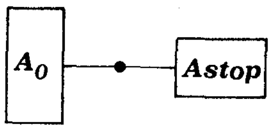

Fig. 8c An initial distributed system.

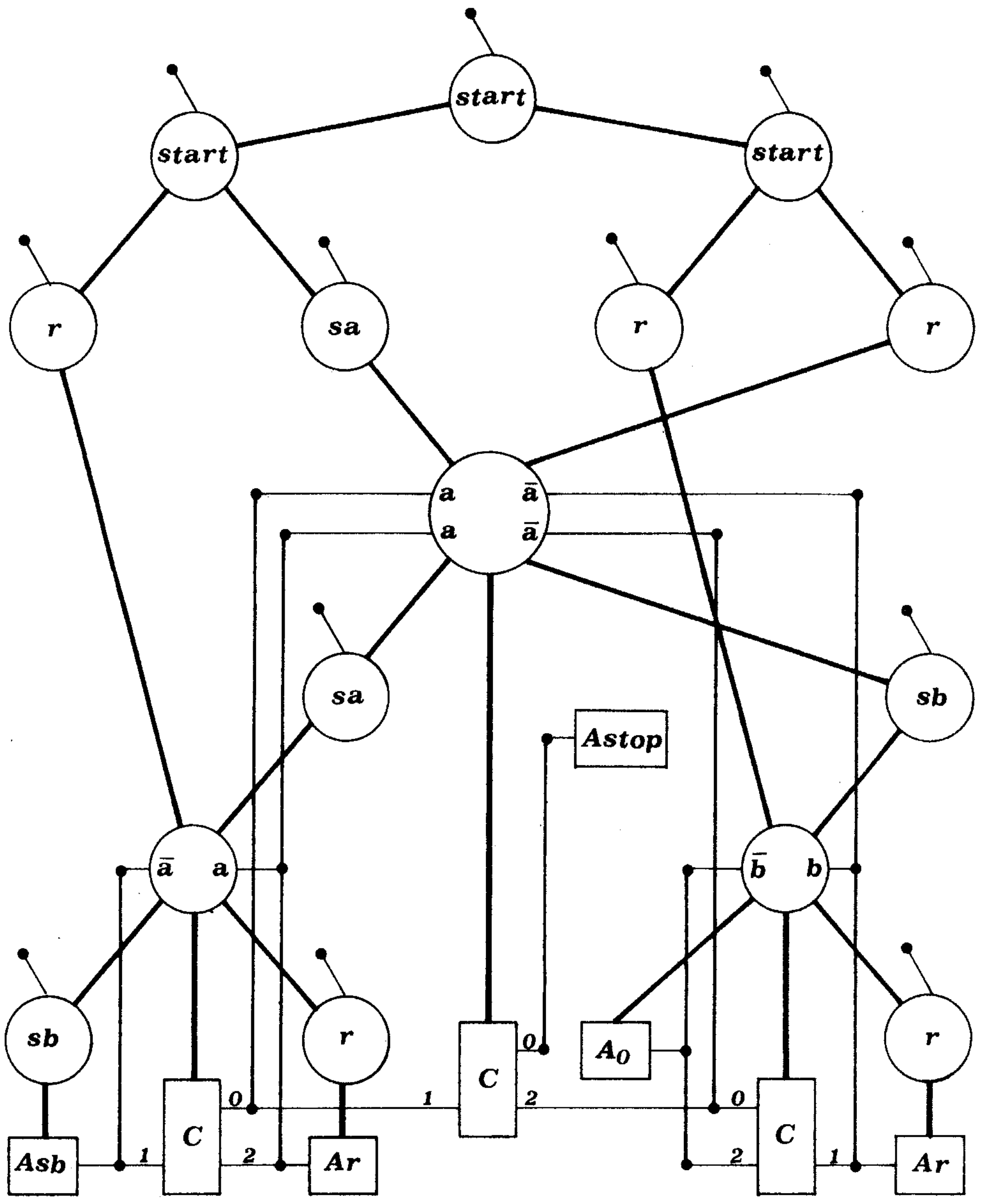

Fig. 9 An intermediate step in a derivation.

decision taken) connected to a local port, and thus they do not need any synchronization. Therefore the involved nondeterminism might be called local. The subsystems generated in the case of a send decision are labeled with symbols (Asa and Asb) with only one production. Thus the actual send operation is completely determinate. On the contrary, the receive symbol Ar can be rewritten in two ways corresponding to receiving an a or a b (dashed characters are used, to forbid communication between two senders or two receivers). Since in both cases the event is connected to an external node, the choice between the two is taken according to the possibilities of synchronization with external entities (here the communication network and, eventually, the sender). Thus the involved nondeterminism might be called global.

Symbol C labeling the elements of the communication network can be rewritten in 13 ways. The first 8 ways (first row) correspond to events synchronizing an internal node of the cell with the external node and assigning the same terminal symbol (a,b,$\overline{a}$ or $\overline{b}$) to both nodes. The second 4 ways (second row) correspond to events synchronising two internal nodes of the cell and thus assign complementary symbols to the two nodes: One is the sender and the other is the receiver. The last production (third row) corresponds to termination. Notice that since the termination event is connected to all the three nodes, the communication network behaves in this case as a broadcasting network and not as a point-to-point network.

In Fig. 9 we see a distributed system derivable with the productions in Fig. 8a and 8b from the initial system in Fig. 8c. Notice that three start events have happened, and thus four A_0 subsystems have been generated. Looking at the nonterminal subsystems at the bottom of the figure, we see that the four subsystems are organized in two cells, plus a central communication element connecting the two cells. Going back to the events, it is clear that the first communication took place between the right station of the left cell and the right station of the right cell. Notice that if all stations decide either to send or to receive, a deadlock occurs, no rewriting rule is applicable and, according to our definitions, the computation is aborted since no terminal system can be derived. Going back to the system in Fig. 9, the next step is the concurrent occurrence of two internal communications in the two cells. Thus we can say that our communication network has no centralized control, and two communication paths can be active at the same time, provided they are disjoint.

The following theorem states the link between the parallelism aspects modeled by the concurrency relation in a (terminal) system and those implicit in the possibility of applying many independent rewriting rules to the same graph. We need a definition. Since every rewriting rule generates exactly one event, a finite computation induces a total ordering on the events of its final system (i.e. its last element). This ordering is called the generation ordering.

Theorem 6.1. Given a distributed system D= $(P,S,f,l,\leq)$ generated by a grammar G, the possible generation orderings of its events are exactly those total orderings compatible with (i.e. larger than) the causal ordering $\leq$ of D.

Proof outline. A generation ordering respects the causal ordering.

A nonterminal cannot precede a terminal, and therefore any newly generated event must be a maximal of the causal ordering.

Every total ordering compatible with the causal ordering is a generation ordering.

Given two total orderings compatible with a partial ordering, it is well known that it is possible to find a sequence of compatible total orderings between the two given total orderings, such that two adjacent orderings in the sequence are equal except for the permutation of two contiguous elements.

Thus the thesis is true provided we show that if two concurrent events are contiguous in a generation ordering, it is possible to obtain another generation ordering by permuting the two events. But it is easy to see that two concurrent contiguous events can be generated only by the consecutive independent applications of two rewriting rules. Namely, every subsystem rewritten by the second rule is concurrent with every subsystem generated by the first rule. Thus the rules can also be applied in the reverse order, without changing the result.

Q.E.D.

7. Conclusions

In the previous section, the language generated by a grammar was defined as a set of finite, terminal distributed systems. While this was the simplest way of extending the concepts of formal languages theory to fit our needs, it is not yet satisfactory as a model of real systems. A number of useful extensions can be conceived. First, infinite graphs and infinite derivations might conveniently represent nonterminating systems. Furthermore, a distributed system containing a nonterminal subsystem which cannot be rewritten, may correspond to a deadlocked computation, and thus may conceivably appear in the language. Finally, a set of distributed systems may contain insufficient information, since the exact point (in time and space) where a decision was taken between two possible computations does not appear in the language. Factorization of the distributed systems of the language in the form of a "thick" tree may be possible, thus providing a LES as the semantics of a grammar.

References

/1/ E.Astesiano, G.Reggio, E.Zucca. Operational Frameworks for Semantic Description of Concurrent Languages, with an application to ADA-like languages, Internal Report CNET-61, Dept. of Mathematics, Univ. of Genova.

/2/ R.Milner. A Calculus of Communicating Systems, Springer LNCS n° 92, 1980.

/3/ R.Milner. Calculi for Synchrony and Asynchrony, Internal Report CSR-104-82, Edinburgh University, February 1982

/4/ C.A.R.Hoare, S.D.Brookes, A.W.Roscoe. A Theory of Communicating Sequential Processes, Technical Monograph PRG-16, Programming Research Group, Oxford University, 1981.

/5/ C.A.Petri. Concurrency, Proc. Net Theory and Applications, Springer LNCS n° 84, 1980, pp. 251-260.

/6/ C.A.R. Hoare. Communicating Sequential Processes, Comm. ACM 21, August 1978, pp. 666-677.

/7/ M.Nielsen, G.Plotkin, G.Winskel. Petri nets, event structures and domains, part 1, TCS 13, 1981.

/8/ G.Winskel. Event structure semantics for CCS and related languages, ICALP '82, Springer LNCS n° 140, July 1982.

/9/ J.Winkowski. Behaviours of Concurrent Systems, TCS 11, 1980, pp. 39-60.

/10/ U.Montanari, C.Simonelli. On Distinguishing Concurrency from Nondeterminism, Proc. Réseaux de Petri et Parallélisme, Colleville sur mer, May 1980.

/11/ I.Castellani, P.Franceschi, U.Montanari. Labeled Event Structures: A Model for Observable Concurrency, IFIP TC 2 - Working Conference: Formal Description of Programming Concepts II, Garmisch - Partenkirchen, June 1982, NorthHolland, to appear.