Math. Struct. in Comp. Science (2002), *vol.* 12, *pp.* 377–403. © 2002 Cambridge University Press
DOI: 10.1017/S0960129501003589 Printed in the United Kingdom

An event structure based semantics for high-level message sequence charts[†]

LOÏC HÉLOUËT[‡], CLAUDE JARD[§] and BENOÎT CAILLAUD[¶]

[‡] *INRIA, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France*
Email: `loic.helouet@irisa.fr`

[§] *CNRS, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France*
Email: `jard@irisa.fr`

[¶] *INRIA, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France*
Email: `bcaillau@irisa.fr`

Received 13 July 2000; revised 15 December 2000

This paper details a partial order semantics for families of scenarios represented by High-Level Message Sequence Charts (HMSCs): graph grammars generating event structures are used to represent HMSCs. A decision procedure for HMSC equivalence is then described. This can be considered as a first step towards the formal manipulation of scenarios.

1. Introduction

1.1. *Scenario-based languages*

The supervision and formalisation of the very first phases of design in software development are the objects of increasing attention. The passage from abstract to formal descriptions appears to be a crucial stage. Formal techniques often neglect this aspect, which has been marginalised by methodological aspects. The appearance of scenario-based languages is a response to this deficiency, in particular, in the context of distributed systems.

Scenarios consist of drawings, in which vertical lines model time elapsing on communicating entities, and arrows model message exchanges. They have many advantages: they are easily understood, even by the non-specialist; they bring a clear graphical representation of complex systems, and abstract most of the implementation details, concentrating on causality and independence of events, the two cornerstones of distributed systems modelling; and though informal, they are precise enough to contain information about processes, message types, *etc.*

Scenarios are widely used in telecommunication standards, where they help document a model, and provide a global understanding of distributed protocols. They are also used

[†] 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.

to present simulation traces or to specify properties, as in the SDL tools SDT[†] or Object-Géode[‡], or in the verification tool SPIN (Holzmann 1997) for instance.

Several formal notations such as 'Message Sequence Charts' (MSCs) have been proposed. MSCs are mostly used for describing the activities of communicating entities in distributed systems. A closely related formalism called Sequence Diagrams can be found in the Unified Modelling Language (OMG 1997) (a convergence between these two notations is expected in the near future). An MSC graphically describes the communication between processes (called *Instances*). MSCs give a very intuitive and visual representation of system behaviours.

However, scenarios are not used at all stages of development, and their use is often restricted to documentation purposes. The main reason is the lack of flexibility when composing scenarios. The difficulty is introduced by the number of scenarios that have to be combined so that a complete system specification can be obtained. A simple answer to this problem is to build families of scenarios, using a kind of 'scenario automaton'. The most elaborate proposal for MSCs that follows this idea is called 'High level MSC' or HMSC (Rudolph *et al.* 1996). HMSCs define MSC compositions with operators such as sequencing, choice, environmental composition and hierarchical composition. An HMSC is a graph, the nodes of which can be starting nodes, end nodes, choices, MSCs or references to other HMSCs. In order to keep to the essential, this paper will only deal with a subset of HMSCs, limited to choice and sequencing operators.

1.2. *Why a partial order semantics?*

The question of MSC semantics has been treated in many papers (Grabowski *et al.* 1993; Reniers and Mauw 1994; Mauw 1996; Leue and Ladkin 1994a; Leue and Ladkin 1994b; Leue and Ladkin 1994c; Kosiuczenko 1997), but the semantics for HMSCs is still in its infancy. Providing a good semantics with enough expressiveness and computability for infinite families of MSCs appeared a difficult challenge. A first interleaving semantics for HMSCs was proposed in Reniers and Mauw (1996) and Reniers (1998), and is now considered as the 'official' semantics of HMSCs. More recently, Katoen and Lambert (1998) and Heymer (1998) defined partial order semantics. Another vision of HMSCs was proposed in Harel and Damm (1998), where some scenarios are 'mandatory', and some are 'optional'.

One of the main advantages of MSCs is that the independence of events is represented graphically. This can be the case between two events belonging to different instances or when two events of the same instances are in the same coregion (depicted by a dashed line in Figure 1b). A coregion releases the sequential order defined along an instance axis, and describes a potential event independence. The interleaving semantics based on the set of total orderings defined by the notation does not capture the independence and does not distinguish between the two situations of Figure 1. In the leftmost figure,

[†] See `http://www.telelogic.com/products/tau/languages/msc.cfm`
[‡] See `http://www.telelogic.com/products/objectgeode/overview.cfm`

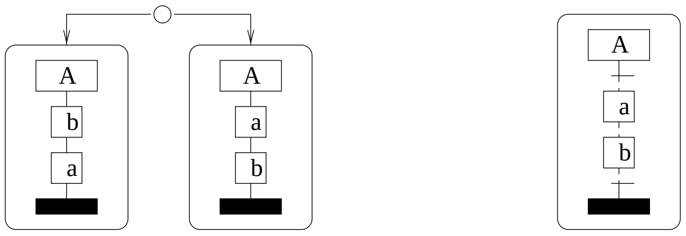


Fig. 1. Example MSCs: a) non-determinism b) potential parallel execution

two sequences $a;b$ and $b;a$ are allowed, while in the rightmost figure, a and b could be executed concurrently. This is the usual argument for a partial order semantics.

A second aspect concerns the semantics of concatenation (sequential composition, written $M1;M2$ from now on) of two MSCs M_1 and M_2. Many tools use a strong sequential composition that, in contrast to the standard, considers that every event of the first MSC M_1 must precede every event of the second MSC M_2. This implicitly defines a synchronisation barrier between each pattern of behaviour as described by the MSCs. This reduces HMSCs to finite state machines. This low expressive power is not consistent with the goal of a high level specification language. We agree with the standard that a weaker sequential composition, allowing the description of infinite behaviours with possibly unbounded communication queues and asynchrony between events, should be used. Problems of this kind are to be solved in a further step of the refinement of the specification. We have in mind that HMSCs are intended to describe incomplete specifications.

The semantics of choices is the third point. The representation of system behaviours by MSCs allows for the introduction of *non-local choices*. For example, HMSC P_0, presented in Figure 3-a, includes an exclusive distributed choice between two scenarios M_2 and M_3. To be consistent with the specification, an implementation will have to ensure that only these two scenarios $M_1;M_2$ and $M_1;M_3$ can be performed, with the decision taken after $M1$. As the decision to perform M_2 or M_3 is distributed, this cannot be done without a synchronisation between A and B. HMSCs P_0 in Figure 3-a and P_1 in Figure 3-b are different, as in P_1, the choice to perform $M_1;M_2$ or $M_1;M_3$ is taken at the beginning of the specification, and by a single instance, which is not the case in P_0. A semantics of HMSCs must help in the detection of such problems, and take into account the control structure expressed within an HMSC, that is, where decisions take place when an alternative occurs. Such a semantics should distinguish between the specifications given in Figure 3-a and Figure 3-b.

Prime event structures (Winskel *et al.* 1981) are such a model that both uses a non-interleaving semantics, and can help to underline choices. Of course, these models can be infinite when representing an infinite state-space system, but we will show that a finite representation of their covering graphs can be computed using a graph grammar.

Language equality on HMSCs is reputed to be undecidable (Muscholl *et al.* 1998; Caillaud *et al.* 2000). Nevertheless, this does not mean that there is no way of comparing two specifications. Following this idea, we define an equivalence of HMSCs as the isomorphism of their event structure representation. This equivalence turns out to be strictly more discriminating than language equality. The decision procedure relies on existing results, allowing us to decide in polynomial time whether two graphs are isomorphic or not in the case of deterministic graph grammars generating finite branching connected graphs (Caucal 1992). The hard technical point is to ensure the finite branching type of our graphs. This is made through a grammar transformation that eliminates infinite branching in graph grammars obtained from HMSCs, while preserving equivalence properties.

1.3. *Outline of the paper*

This paper is organised as follows. Section 2 introduces the Message Sequence Charts notation, and their partial order semantics. Section 3 recalls basic notions about graph grammars, and details the computation of a graph grammar generating an infinite event structure from a HMSC. Section 4 proposes an equivalence decision algorithm based on the semantics of Section 2.

2. bMSCs and HMSCs

Message Sequence Charts are used to give graphical definitions of distributed systems and the communications between the components of these systems. The purpose of this section is to describe just the features and vocabulary we will use in the rest of the paper, rather than provide a complete description of MSCs, which can be found in the ITU standard Z.120 (ITU 1996) and in Reniers (1998).

2.1. *bMSCs*

A bMSC (basic MSC) defines a simple scenario, for example, an abstraction of a system behaviour. Within bMSCs, processes are called *instances*, and are represented by a vertical axis along which *events* are put in a top-down order. An event can be an emission (events e_1 and e_4 in Figure 2) or a reception of a message (e_3 and e_6) an operation (set, reset, timeout) on a timer (e_2 and e_5), an atomic action (event e_7). Message exchanges are represented by arrows labelled by message names from the emitting to the receiving instance. No assumption whatsoever is made about the communication medium. A bMSC defines a precedence relation between events:

— The emission of a message precedes its reception.

— For any event e, all events situated above e on the same instance axis are predecessors of e. The order on the axis can be relaxed in some parts of the instance called *coregions.* These coregions are represented by dashed parts of the instance axis (as in Figure 1-b). Events situated in a coregion are not necessarily concurrent: their order is not specified yet, or is not important for the specification.

The set of finite bMSCs is denoted by $\mathscr{M}$.

2.2. *HMSCs*

A bMSC defines only one scenario. Extending bMSCs requires a higher-level notation, allowing for the specification of a set of scenarios, made up of combinations of bMSCs.

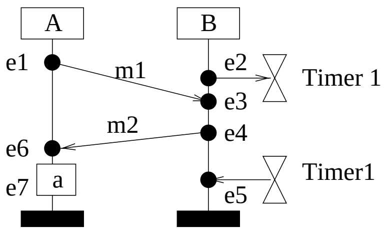
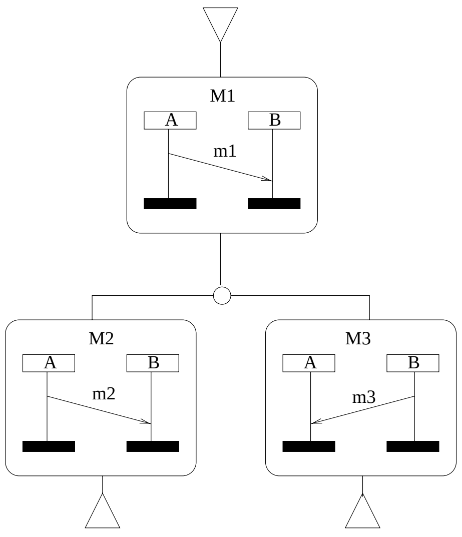


Fig. 2. An example bMSC.

An HMSC can be seen as a high level graph, the nodes of which are start nodes, end nodes, bMSCs, connection symbols, or references to other HMSCs.

HMSCs allow for the use of alternatives, and therefore define sets of scenarios. HMSC P_0 in Figure 3-a defines two possible scenarios, represented in Figure 4. An HMSC can also describe an infinite behaviour, as in Figure 5.



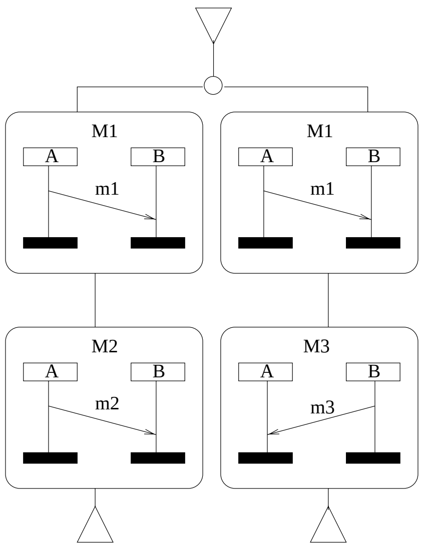


Fig. 3. a) HMSC P_0 (with non-local choice) b) HMSC P_1

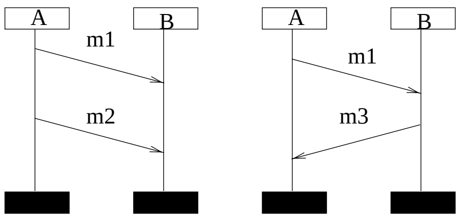


Fig. 4. Scenarios defined by the HMSCs in Figure 3-a and Figure 3-b

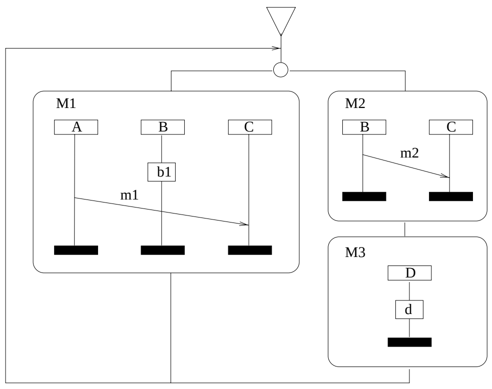


Fig. 5. HMSC P_2

We now define a convenient notation for manipulating HMSCs. The structure of the graph is expressed by means of regular expressions over bMSCs. To each HMSC, we associate an expression: $P ::= \epsilon \mid M;P \mid \sum_{k\in 1..K} M_k;P_k \mid recX.(P) \mid X$, where $M \in \mathscr{M}$ is a bMSC, ; is a sequence operator, ϵ denotes the end of an HMSC, X is a bound variable, $\sum_{k\in 1..K} M_k;P_k$ is a *choice* on a finite number of expressions of the kind $M;P$, and *rec* denotes recursion. For a given k, $M_k;P_k$ will be called a *branch* of the choice. An instance is said to be *active* in a branch if it performs at least one action in this branch. The expression associated to the HMSC in Figure 5 is $P_2 ::= recX.(M1;X + M2;M3;X)$.

2.3. *Partial order semantics for MSCs*

Let us consider a bMSC M describing the behaviour of a finite set of instances, hereafter called I. Two events performed by the same instance are ordered according to their coordinates on the instance axis, provided they are not in a coregion. If they are performed by different instances, they are ordered if and only if they are separated by at least one message exchange. So, a bMSC defines a partial order between events. In Figure 2, e_1 and e_4 are ordered, but e_1 and e_2 are not. The semantics of a bMSC can be formalised by a poset $< E, \leqslant, \alpha, A, I >$, where E is a set of events, $\leqslant$ is a partial order relation (reflexive, transitive and antisymmetric binary relation) on E called the *causal dependence relation*, α is a labelling function from E to $A \times I$, and A is a set of atomic action names ($!m$ and $?m$ will denote sending and receiving of a message, $set(T)$, $reset(T)$ and $timeout(T)$ will denote operations on a timer T, and atomic actions will be labelled by the action name enclosed in the rectangular action symbol).

Given an event $e \in E$, we write $\phi(e) = i$ when $\exists a \in A \mid \alpha(e) = (a, i)$ (event e is performed by instance $i \in I$). The partial ordering on events easily extends to sequences of bMSCs.

Let $M_1 =< E_1, \leqslant_1, \alpha_1, A_1, I_1 >$ and $M_2 =< E_2, \leqslant_2, \alpha_2, A_2, I_2 >$ be the partial orders defined by two MSCs. The *concatenation* of M_1 and M_2, written $M_1 \circ M_2$ is defined by

$$M_1 \circ M_2 =< E_1 \cup E_2, \leqslant_{M_1 \circ M_2}, \alpha_1 \cup \alpha_2, A_1 \cup A_2, I_1 \cup I_2 >,$$

where:

$$\begin{aligned}\leqslant_{M_1 \circ M_2} \;=\; & \leqslant_1 \cup \leqslant_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \\ & \exists (e_1', e_2') \in E_1 \times E_2 \wedge \phi_1(e_1') = \phi_2(e_2') \wedge e_1 \leqslant_1 e_1' \wedge e_2' \leqslant_2 e_2\}.\end{aligned}$$

More intuitively, a concatenation 'glues' together two MSCs along their common instance axis. This operation is similar to the local sequencing of orders defined by Pratt (1986)[†], and is often referred to in the MSC-related literature as the weak sequential composition.

Thus, when the choice operator is included, it can be seen that an HMSC defines a set of scenarios, which are the concatenations of bMSCs along all its paths. Such sets will be called *partial order families* (POF for short) in the rest of this paper. POF semantics for HMSCs have been defined by Katoen and Lambert (1998) and Heymer (1998). However, the POF representation abstracts the branching information at choices, which can be a very important characteristic of an HMSC, as shown previously. Furthermore, POFs cannot be manipulated easily because they are infinite. We now define a finite representation of an HMSC based on event structures and graph grammars that preserves both the partial order semantics of bMSCs and the branching information of HMSCs. This representation will be computed directly from an HMSC, and will allow for equivalence decision.

2.4. *An event structure semantics for HMSCs*

A prime event structure (Winskel *et al.* 1981), which we will abbreviate to ES, is a 6-tuple $< E, \leqslant, \sharp, \alpha, A, I >$, where E, A , I, $\leqslant$ and α have the same meaning as in Section 2.3, and $\sharp$ is a symmetric anti-reflexive binary relation called *conflict relation*, such that

$$\forall e \in E, \forall e' \in E, e \sharp e' \Leftrightarrow \forall e'' \in E, (e' \leqslant e'' \Rightarrow e \sharp e'')$$

(conflicts are inherited through the causality relation).

A conflict between two events e and e' is said to be *minimal* if it cannot be deduced from the conflict inheritance property, that is, $\nexists e'' \in E \mid (e'' \leqslant e' \wedge e \sharp e'') \vee (e'' \leqslant e \wedge e' \sharp e'')$. An event structure defines a domain of *configurations*, which can be seen as possible states of the system. A configuration is a subset C of E that is conflict-free ($\forall e \in C, \nexists e' \in C \mid e \sharp e'$), and causally downward-closed ($\forall e \in C, e' \leqslant e \Rightarrow e' \in C$). A *maximal configuration* of an ES $S =< E, \leqslant, \sharp, \alpha, A, I >$ is a subset C of E such that

$$\forall e \in E - C, \exists e' \in C \mid e' \sharp e$$

For example, $C \cup \{e\}$ is not a configuration.

The event structure defined by the application of the *conflict operation* to two ESs S_1 and S_2 is denoted $S_1 \sharp S_2$ and is the ES:

$$S_1 \sharp S_2 =< E_1 \cup E_2, \leqslant_1 \cup \leqslant_2, \sharp, \alpha_1 \cup \alpha_2 >, \text{ such that } \sharp = (E_1 \times E_2) \cup (E_2 \times E_1) \cup \sharp_1 \cup \sharp_2.$$

We can now generalise the notation to an arbitrary number of ES with $\underset{i \in 1..n}{\sharp}(S_i)$. We also

[†] The same operation has been defined by Rensink and Wehrheim (1994) for process algebra

extend the concatenation operator $\circ$ such that for a bMSC M and a structure S:

$$M \circ S =< E_M \cup E_S, \leqslant_{M \circ S}, \sharp_M, \alpha_M \cup \alpha_S, A_M \cup A_S, I_M \cup I_S > .$$

Clearly, an event structure semantics can be associated to an HMSC, by constructing the order relation with the sequence operator along all branches of the HMSC, and requiring any pair of events on different branches to be conflicting. We can define inductively the ES of index k associated to any HMSC $P \in (\mathscr{M}, ;, \Sigma, rec, \epsilon)$:

— $ES_0(P) =< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$
— $ES_k(M; P) = M \circ ES_{k-1}(P)$
— $ES_k(\sum_{i \in 1..n} E_i = \sharp_{i \in 1..n} (ES_{k-1}(E_i))$
— $ES_k(recX(P)) = ES_{k-1}(P_{[X:=recX.(P)]})$
— $ES_k(\epsilon) =< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$

The potentially infinite event structure associated to an HMSC P is $ES_\omega(P)$.

2.5. *Covering graphs*

An event structure is often represented by a graph that associates a vertex to each event, an edge labelled with $\sharp$ to each minimal conflict, and a directed edge to each pair in the covering of the order relation. The other conflict and causality edges can be deduced from the graph representation, using the conflict inheritance property and the transitivity of the order relation. Furthermore, when no confusion can arise, we will represent events by their action name.

Infinite behaviours lead to infinite graphs. Unfortunately, the resulting graph is not necessarily a regular graph, as an event can be connected to an infinite number of events via a conflict. Figure 7 shows the graphical representation of the event structure obtained from the HMSC in Figure 6 (for clarity, all edges are represented, and a dashed line leaving a node represents an infinite set of outgoing edges). By restricting this graph to events labelled by a and conflict edges, we can extract the graph in Figure 8, which is a well-known irregular graph.

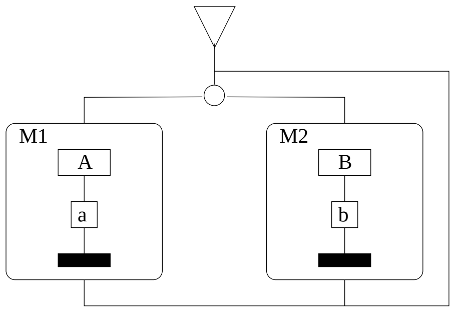


Fig. 6. HMSC with an irregular ES representation

In order to solve the problem of regularity caused by conflict edges in the graph representation of an event structure, we define a new type of edge, called a conflict

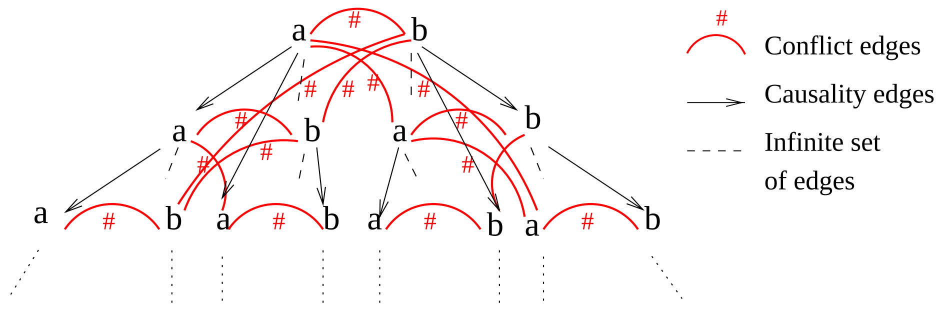


Fig. 7. Irregular ES graph for HMSC of Figure 6

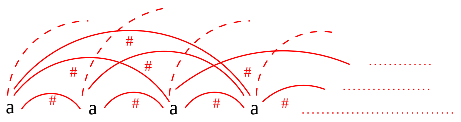


Fig. 8. An irregular subgraph extracted from the graph in Figure 7

inheritance edge. This relation will allow an event e' to inherit all conflicts from an event e, without being causally dependent on e.

An ES can be represented by a *covering graph*: $< E, \longrightarrow, \rightsquigarrow, \sharp_c, \alpha, A, I >$, such that:

— $\longrightarrow = \{(e,e') \in \leqslant \mid e \neq e' \wedge \nexists e'' \in E - \{e,e'\}, e \leqslant e'' \leqslant e'\}$ is the covering of $\leqslant$,

— $\rightsquigarrow$ is a conflict inheritance relation: $e_1 \rightsquigarrow e_2$ if and only if $\forall e' \in E \mid e' \sharp e_1$, then $e' \sharp e_2$,

— $\sharp_c = \{(e,e') \mid \nexists e'' \in E, ((e'' \rightsquigarrow e \vee e'' \leqslant e) \wedge e'' \sharp e') \vee ((e'' \rightsquigarrow e' \vee e'' \leqslant e') \wedge e'' \sharp e)\}$ is the set of minimal conflicts with respect to $\leqslant$ and $\rightsquigarrow$.

The conflict inheritance relation allows us to generate covering graphs such that a finite set of conflict edges starts from any event $e \in E$ ($\forall e \in E, \{e' \mid e \sharp_c e'\}$ is finite).

A covering graph of the event structure of Figure 7 is represented in Figure 9. The new relation $\rightsquigarrow$ is represented by dotted arrows, and other edges have the same meaning.

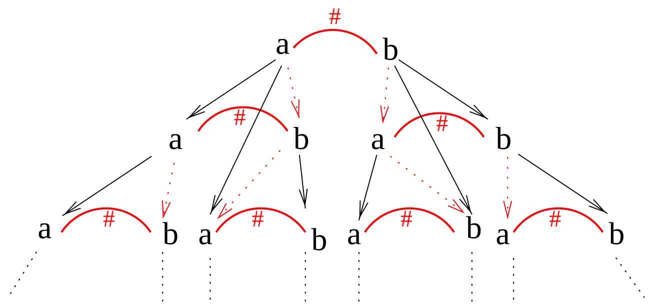


Fig. 9. Regular covering graph for HMSC Figure 6

Let $M \in \mathscr{M}$ be a bMSC, and $\mathscr{G}r(M) =< E, \longrightarrow, \rightsquigarrow, \sharp, \alpha, A, I >$ be its (finite) covering graph. We define the following operations:

— For all $i \in I, E_i = \{e \in E \mid \phi(e) = i\}$.

— $Inf(\mathcal{G}r(M)) = \{e \in E \mid \forall e' \in E, (e', e) \notin \longrightarrow\}$ is the set of minimal events of $\mathcal{G}r(M)$ with respect to $\longrightarrow$.
— $Sup(\mathcal{G}r(M)) = \{e \in E \mid \forall e' \in E, (e, e') \notin \longrightarrow\}$ is the set of maximal events of $\mathcal{G}r(M)$ with respect to $\longrightarrow$.
— For all $i \in I, Inf_i(\mathcal{G}r(M)) = \{e \in E_i \mid \forall e' \in E_i, (e', e) \notin \longrightarrow\}$ is the set of minimal events on instance i.
— For all $i \in I, Sup_i(\mathcal{G}r(M)) = \{e \in E_i \mid \forall e' \in E_i, (e, e') \notin \longrightarrow\}$ is the set of maximal events on instance i.
— $Act(\mathcal{G}r(M)) = \{i \in I \mid \exists e \in S, \phi(e) = i\}$: is the set of instances that perform at least an event within M.

We will also write $E_{\mathcal{G}r(M)}$ and $\underset{\mathcal{G}r(M)}{\longrightarrow}$ for the vertices and causality edges of $\mathcal{G}r(M)$. The graphical representation of the covering graph of bMSC M_1 in Figure 2 is represented in Figure 10. For this example, we have the following sets:

— $E_A(\mathcal{G}r(M_1)) = \{e_1, e_6, e_7\}$, $E_B(\mathcal{G}r(M_1)) = \{e_2, e_3, e_4, e_5\}$,
— $Inf(\mathcal{G}r(M_1)) = \{e_1, e_2\}$,
— $Sup(\mathcal{G}r(M_1)) = \{e_5, e_7\}$,
— $Act(\mathcal{G}r(M_1)) = \{A, B\}$,
— $Inf_A(\mathcal{G}r(M_1)) = \{e_1\}$, $Inf_B(\mathcal{G}r(M_1)) = \{e_2\}$,
— $Sup_A(\mathcal{G}r(M_1)) = \{e_7\}$, $Sup_B(\mathcal{G}r(M_1)) = \{e_5\}$.

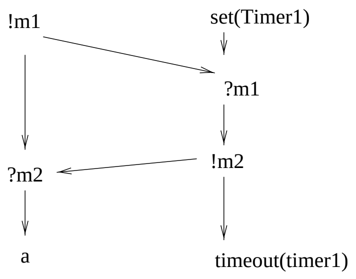


Fig. 10. $\mathcal{G}r(M_1)$: covering graph of bMSC M_1 in Figure 2

3. Graph grammar generation from an HMSC

An event structure semantics has been defined, but this representation of HMSCs may be infinite. This section details how the infinite covering graph $\mathcal{G}r(ES_\omega(P))$ of a Message Sequence Chart P can be expressed finitely by a hyperedge replacement graph grammar. First, a short introduction to hyperedge replacement grammars (Habel 1989) is given. Then a method for generating graph grammars from an HMSC is described.

3.1. *Graph grammars*

Graph grammars can be considered as a generalisation of classical grammars in which strings are replaced by graphs. Non-terminal elements are called *hyperarcs*, and terminal elements are vertices and edges.

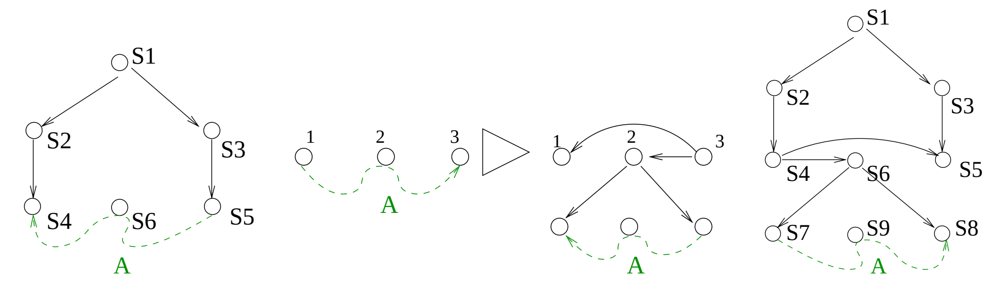


Fig. 11 a) Hypergraph H b) rule R c) Hypergraph H': derivation of H by R

A *hyperarc* is a word $X = l.s_1.\cdots s_n$ such that l is a letter from a finite alphabet L, and $s_1.\cdots s_n$ is a list of vertices. Graphically, a hyperarc will be represented by a dashed line linking vertices $s_1.\cdots s_n$, and labelled by l. In the example of Figure 3.1-a, vertices s_4, s_5 and s_6 are linked by a hyperarc labelled by A. This hyperarc will then be written $A.s_5.s_6.s_4$.

A *hypergraph* is a pair $G = (T, H)$, where T is a finite graph (for instance a part of a covering graph), and H is a set of hyperarcs linking vertices of T. Figure 3.1-a is an example of a hypergraph containing one hyperarc.

A *graph grammar* $\mathscr{G} = (G_0, \mathscr{R})$ consists of a hypergraph G_0 called the *axiom* of the grammar, and a set $\mathscr{R}$ of rewriting rules.

A *rewriting rule* is a pair $r = (X, Y)$, where X is a hyperarc called the *left part* of r, and Y is a hypergraph called the *right part* of r. We shall often write this rule $X \triangleright Y$. Such a rewriting rule indicates how a hyperarc X can be replaced by a pattern Y. In rule R of Figure 3.1-b, a hyperarc labelled by A linking 3 vertices (1, 2 and 3) can be replaced by a hypergraph with 6 vertices (including vertices 1, 2 and 3), 4 edges, and a hyperarc labelled by A. The rewriting of a hyperarc X in a hypergraph M using a rule $r = (X, Y)$ can be performed in two steps:

— remove hyperarc X from M,

— replace vertices of X by the pattern defined by Y.

Note that this definition allows vertices of X to be deleted during the rewriting phase. Let M and N be hypergraphs, and let $r = (X, Y)$ be a rule of a grammar $\mathscr{G}$. We will say that N is a *direct derivation* of M by rule (X, Y) (written $M \underset{\mathscr{G},(X,Y)}{\longrightarrow} N$ if M can be rewritten into N by replacing X by Y in M. A sequence of direct derivations $H \underset{\mathscr{G},(X_1,Y_1)}{\longrightarrow} H_1 \underset{\mathscr{G},(X_2,Y_2)}{\longrightarrow} \ldots \underset{\mathscr{G},(X_n,Y_n)}{\longrightarrow} H_n$ is called a *derivation of length n*. Hypergraph H' in Figure 3.1-c shows the hypergraph obtained by derivation of hypergraph H of Figure 3.1-a by rule R of Figure 3.1- b.

Parallel derivation generalises direct derivation by enabling simultaneous hyperarc replacements. We write $M \underset{\mathscr{G}}{\Longrightarrow} N$ (N is a parallel derivation of M by $\mathscr{G}$) iff there exists a set of rules $P = (X_1, Y_1)...(X_n, Y_n)$ of $\mathscr{G}$, M comprises exactly n hyperarcs, and for any permutation π on $\{1..n\}$, $M \underset{\mathscr{G},(X_{\pi(1)},Y_{\pi(1)})}{\longrightarrow} \ldots \underset{\mathscr{G},(X_{\pi(n)},Y_{\pi(n)})}{\longrightarrow} N$.

We will use $[M]$ to denote the projection of a hypergraph M on its terminal edges. $\mathscr{G}^{\omega}(M)$ denotes the possibly infinite graph generated from M, defined by $\mathscr{G}^{\omega}(M) = \bigcup_{n \in I\!N} [\mathscr{G}^{n}(M)]$, where $\mathscr{G}^{0}(M) = M$ and $\mathscr{G}^{n}(M) \underset{\mathscr{G}}{\Longrightarrow} \mathscr{G}^{n+1}(M)$.

3.2. *Deriving graph grammars from an HMSC*

Because we can use recursive definitions, HMSCs can define infinite event structures. However, the covering graph of an HMSC can be represented finitely by a graph grammar. Let $P \in (\mathscr{M}, ;, \Sigma, rec, \epsilon)$ be an HMSC, given as a regular expression on a set of bMSCs B. The main role of a grammar construction algorithm is to associate a rewriting rule to any sub-expression P' of P. Vertices and edges in the right parts of rules are produced from the covering graphs of bMSCs in B, left parts and hyperarcs in right parts express the structure of the HMSC P.

Furthermore, the rewriting rules associated to a subexpression P' have to take into account how P' was derived from unfolding P, since this may affect the way vertices in the right part of the rule are merged into an already unfolded graph. Consider for example the graph grammar of Figure 13 obtained from HMSC P_2 in Figure 5. This grammar contains a hyperarc labelled by $M1; X + M2; M3; X_1$, and another one by $M1; X + M2; M3; X_2$. These hyperarcs both link 3 vertices, and are both associated to the same subexpression. However, in case 1, vertex 3 represents the last event performed by instance C, while in case 2, the same vertex represents the last event performed by instance D. Consequently, the rules associated to each case will be different. So, derivations of rules have to take into account expressions, but also a *context*, for example, the set of active instances, the branch of the choice that is evaluated, and so on. Note that here the word 'context' represents a kind of history of unfoldings performed before the application of a rule, and does not mean that our grammars are context-sensitive. The alphabet L of hyperarc labels is then composed of couples (P' = sub-expressions of the axiom, con = context), which will be written as P'_{con}.

Let $P \in (\mathscr{M}, ;, \Sigma, rec, \epsilon)$ be a regular expression over partial orders, E be a set of vertices from which a rule will be rewritten (and therefore that will be obtained in the left part of a rule), $LE(E)$ a predicate indicating if a vertex e corresponds to the last event performed by instance $\phi(e)$, Her a predicate on E indicating if a vertex can be the origin of an inheritance edge, Var be a set of pairs $(X, Expr)$ where X is a variable name and $Expr$ a sub-expression of P, and let Br be a list of instances that performed at least an action in the branch of the HMSC currently studied.

We will now define the function $Rules(P, E, LE, Her, Br, Var) = \{(X_j, Y_j)\}_{j \in 1..J}$ that returns the rules that are associated to the HMSC P according to a specific context $\{LE, Her, Br, Var\}$. Rules are of the form $X_i = P_{i_{Con}}.E_i \triangleright Y_i = (T, H = \{P'_k.E'_i\}_{k \in 1..K})$, meaning that in the context Con, a hyperarc labelled by P_i linking events E_i rewrites into a hypergraph Y_i including K hyperarcs of the form $H_k = P'_k.E'_k$. For clarity, the computation of A, I, or α is not defined, and the context is not indicated on the labels of the rules. The rules associated to an HMSC are computed as follows:

— $Rules(\epsilon, E, LE, Her, Br, Var) = \epsilon.E \triangleright (< E, \emptyset, \emptyset, \emptyset, \alpha, A, I >, \emptyset)$

This rule suppresses hyperarcs labelled by ϵ, and is illustrated by Figure 12-b.

— $Rules(M;P,E,LE,Her,Br,Var) = M;P.E \rhd \left(< E_{M;P}, \underset{M;P}{\longrightarrow}, \underset{M;P}{\rightsquigarrow}, \emptyset, \alpha, A, I >, P.E'\right)$
$\cup\ Rules(P,E',LE',Her',Br \cup Act(\mathscr{G}r(M)),Var)$
where:

$$\begin{aligned}
E_{M;P} &= E \cup E_{\mathscr{G}r(M)}\\
\underset{M;P}{\longrightarrow} &= \{(e, Inf_i(\mathscr{G}r(M)) \mid e \in LE(E) \wedge \phi(e) = i \wedge i \in Act(\mathscr{G}r(M))\} \cup \underset{\mathscr{G}r(M)}{\longrightarrow}\\
\underset{M;P}{\rightsquigarrow} &= \{(e,e') \mid e \in E \wedge Her(e) \wedge e' \in Inf(\mathscr{G}r(M)) \wedge \phi(e') = i \wedge i \notin Br\}\\
LE' &= LE \cap E' \cup \{Sup_i(\mathscr{G}r(M)) \mid i \in Act(\mathscr{G}r(M))\}\\
E' &= \{e \in LE(E) \mid \phi(e) \notin Act(\mathscr{G}r(M))\} \cup \{Sup_i(\mathscr{G}r(M)) \mid i \in Act(\mathscr{G}r(M))\}\\
&\quad \cup \{e \in E \cup E_{\mathscr{G}r(M)} \mid e \in Her'\}\\
Her' &= Her \cup \{e \in Inf(\mathscr{G}r(M)) \mid \phi(e) = i \wedge i \notin Br\}.
\end{aligned}$$

This rule prepares the order concatenation between M and the MSCs contained in P. You can also note that any first action performed by an instance since the last choice is added to the origins of inheritance edges. Figure 12-d shows the construction of a rule from a sequence in a HMSC for an initially empty context.

— $Rules(\sum_{k\in 1..K} M_k;P_k,E,LE,Her,Br,Var) =$
$$\sum_{k\in 1..K} M_k;P_k.E \rhd \left(\langle E_{\sum}, \longrightarrow_{\sum}, \rightsquigarrow_{\sum}, \sharp_{\sum}, \alpha, A, I\rangle, \bigcup_{k\in 1..K} P_k.E'_k\right)$$
$\cup \bigcup_{k\in 1..K} Rules\left(P_k, E'_k, LE'_k, Her'_k, Act(\mathscr{G}r(M_k)), Var\right)$
where:

$$\begin{aligned}
E_{\sum} &= E \cup \bigcup_{k\in 1..K} E_{\mathscr{G}r(M_k)}\\
\longrightarrow_{\sum} &= \bigcup_{k\in 1..K} \Big(\{(e, Inf_i(\mathscr{G}r(M_k))) \mid\\
&\qquad e \in LE(E) \wedge \phi(e) = i \wedge i \in Act(\mathscr{G}r(M_k))\} \cup \underset{\mathscr{G}r(M_k)}{\longrightarrow}\Big)\\
\rightsquigarrow_{\sum} &= \bigcup_{k\in 1..K} \{(e,e') \mid e \in E \wedge Her(e) \wedge e' \in Inf(\mathscr{G}r(M_k)) \wedge \phi(e) \neq \phi(e')\}\\
\sharp_{\sum} &= \{(e,e') \mid e \in Inf(\mathscr{G}r(M_m)) \wedge e' \in Inf(\mathscr{G}r(M_n)) \wedge m \neq n\}\\
E'_k &= \{e \in LE(E) \mid \phi(e) \notin Act(\mathscr{G}r(M_k))\}\\
&\quad \cup \{Sup_i(\mathscr{G}r(M_k)) \mid i \in Act(\mathscr{G}r(M_k))\} \cup \{Inf(\mathscr{G}r(M_k))\}\\
Her'_k &= Inf(\mathscr{G}r(M_k))\\
LE'_k &= LE \cap E'_k \cup \{Sup_i(\mathscr{G}r(M_k)) \mid i \in I\}.
\end{aligned}$$

This rule produces the conflicts due to a choice in an HMSC. Note that predicate *Her* is set to $Inf(\mathscr{G}r(M_k))$, for example, the minimal events of each branch. This ensures that no infinite branching will be generated by inheritance edges. Figure 12-c shows the construction of a rule from a choice in a HMSC, and Figure 12-e shows an example of HMSC generating rules with conflict inheritance edges.

— $Rules(recX.(P), E, LE, Her, BR, Var) = recX(P).E \rhd \big(\langle E, \emptyset, \emptyset, \alpha, A, I\rangle,\ P.E\big)$
$\cup\ Rules(P, E, LE, Her, Br, Var \cup \{(X, P)\})$
The construction of a rule from an iterative HMSC is illustrated Figure 12-a.
— $Rules(X, E, LE, Her, Br, Var) = X.E \rhd \big(\langle E, \emptyset, \emptyset, \alpha, A, I\rangle,\ P.E\big)$ with $(X, P) \in Var$
$\cup\ Rules(P, E, LE, Her, Br, Var)$.

Let P be an HMSC. The graph grammar generating the covering graph for P is: $\mathscr{G}_P = \big(\ P, Rules(P, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)\ \big)$. It can be easily shown that the event structure calculated from the covering graph $\mathscr{G}_P^{\omega}(P)$ generated by the grammar is exactly the structure SE_{ω} generated from the HMSC P.

The graph grammar generated for HMSC $P2$ in Figure 5 is represented in Figure 13. Labels for rules are sub-expressions of $recX.(M1; X + M2; M3; X)$, but, for clarity, we have only indicated different contexts by different indexes.

We have now defined a semantics for HMSCs, based on event structures and graph grammars. This semantics must allow for formal manipulations of HMSCs, such as equivalence detection. The rest of the paper defines an equivalence calculus based on covering graphs isomorphism.

4. HMSCs equivalence

It is shown in Caucal (1992) that a *normal form* of any graph grammar generating a connected and finite degree graph can be calculated in polynomial space and time. A graph grammar is said to be in normal form if all the vertices added by a rule are at the same distance from a starting set of vertices. The distance can be the length of the minimal path, but we can also associate a weight to each type of edge. When two graph grammars have the same normal form (modulo a renaming of the rules), the graphs they generate are said to be isomorphic.

Section 3 has shown how to express infinite ESs described by HMSCs using deterministic graph grammars. Unfortunately, an event can have an infinite number of successors. This infinite branching is due to the creation of multiple conflicting copies of the same event with different histories (the n^{th} occurrence of an action may be produced by more than one path in the HMSC). This section first characterises the cases when infinite branching is generated, and then shows that the isomorphism decision procedure can be brought back to the isomorphism of connected graphs of finite degree.

4.1. *Infinite branching*

The covering graph of an HMSC P contains infinite branching if and only if:

— there exists a loop in P,
— this loop includes at least a choice $C = \sum_{i=1..K} M_i; P_i$,
— there are two branches $M_p; P_p$ and $M_n; P_n$ $(n, p \in 1..K)$ such that $n \neq p$, and an instance A performing at least an action in $M_p; P_p$, but not in $M_n; P_n$,
— it is possible to get back to choice C by choosing the branch $M_n; P_n$.

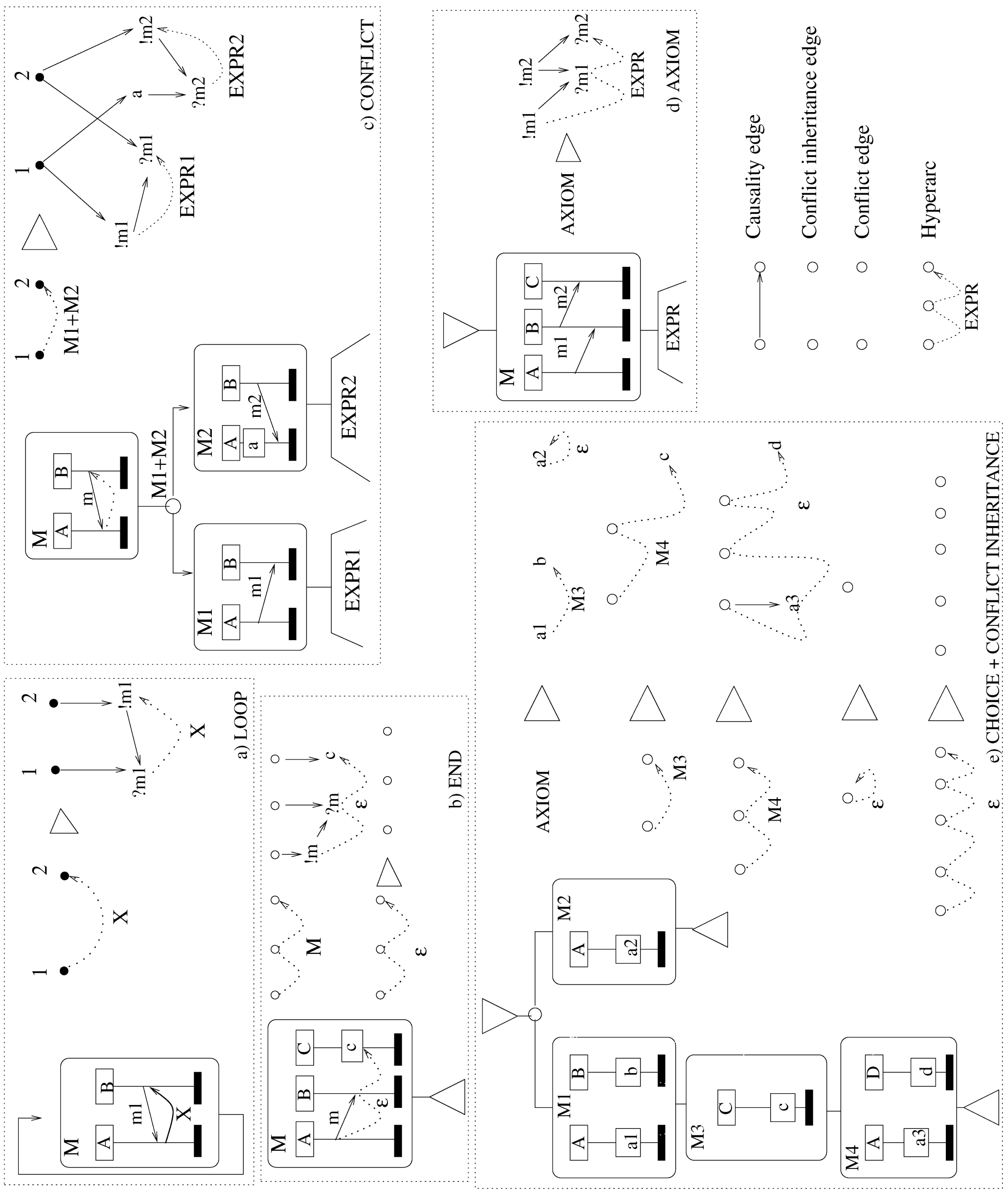


Fig. 12. Illustration of graph grammar calculus from HMSCs

An example of such a specification is given in Figure 5 (instance A is active in M_1, but not in $M_2; M_3$), so that an event labelled by $?m_1$ may have an infinite number of successors labelled with $?m1$. A graph grammar calculated from an HMSC can be modified in order to obtain finite branching covering graphs. For any rule (X, Y), let us compute $App_X = \{(rule, red)\}$, where *rule* is an applicable rule from X, and *red* is the set of vertices of X kept until *rule* is applied. Rule (X, Y) is said to *loop* if a hyperarc labelled by X can be found after applying a finite length derivation to Y $(\exists (X, red) \in App_X)$.

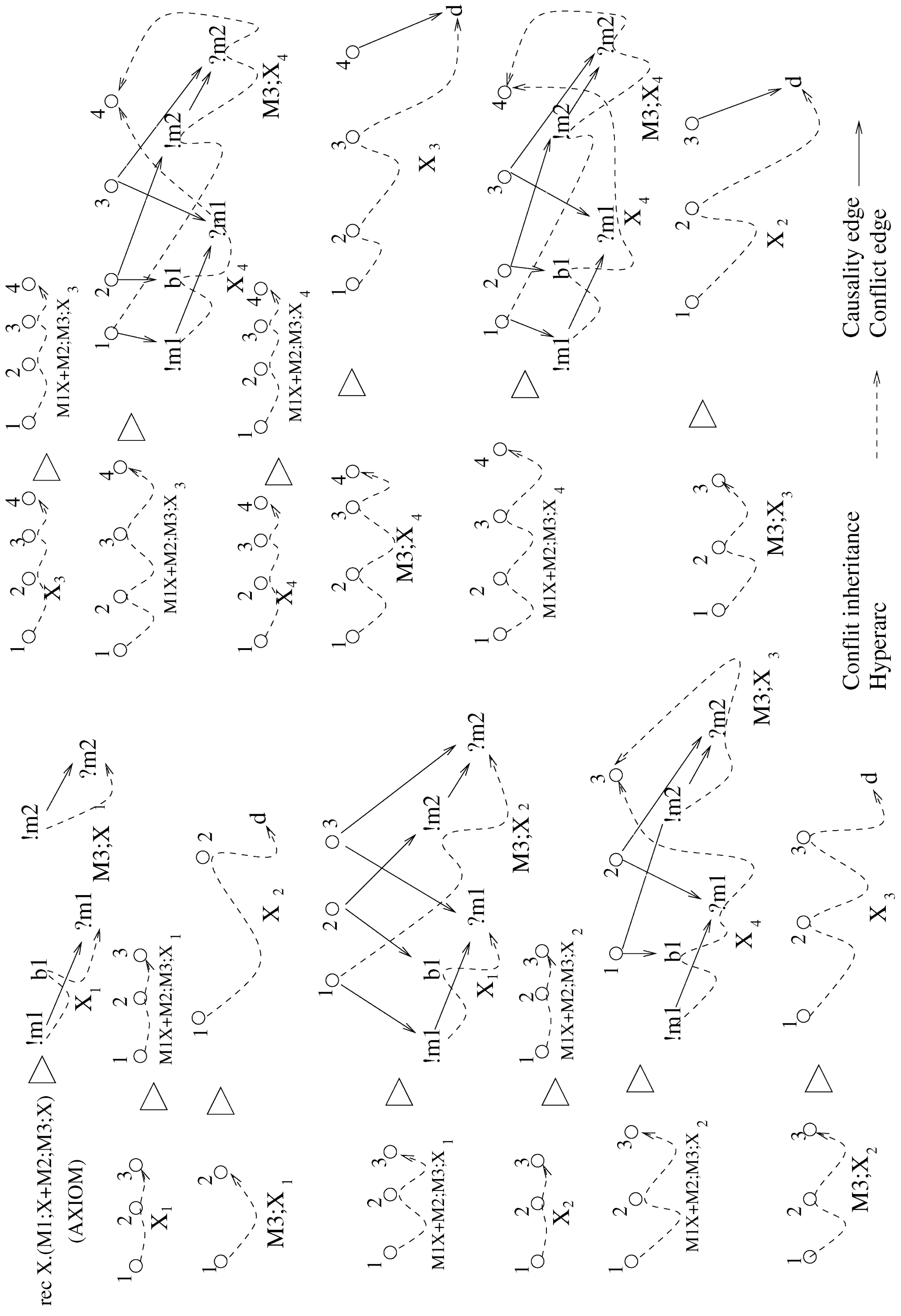


Fig. 13. Graph grammar calculated from HMSC $P2$ in Figure 5

4.2. *Redundancy elimination*

The unfolding of loops will now be eliminated. (X, Y) is said to be *redundant* with $(X', Y') \in \mathscr{G}$ if and only if:

$\exists m, n \in \mathbb{N}, [\mathscr{G}^m(Y)] = [\mathscr{G}^n(Y')] \wedge X' \in \mathscr{G}^m(Y) \wedge X'$ loops $\wedge X = X'$ (up to renaming of rule).

We suppress (X, Y) redundant with (X', Y') by removing (X, Y) in the grammar, and replacing any occurrence of X in the right part of every rule by X'. For instance, rule B of the grammar in Figure 14-a is redundant with rule C.

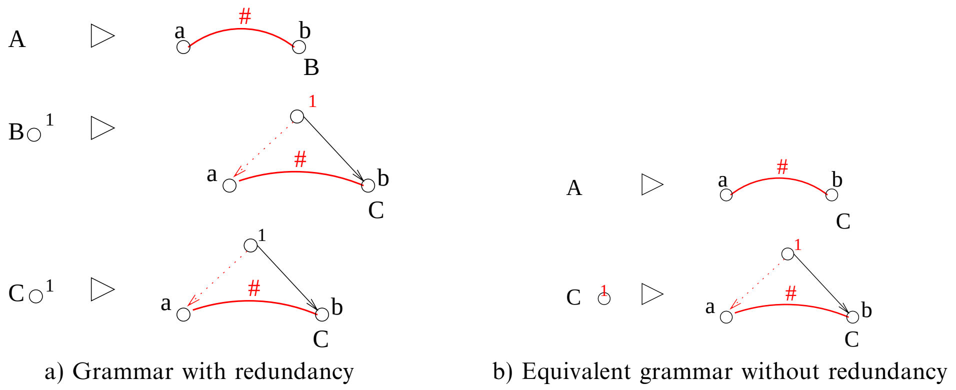

A
a
#
b
B
B
1
1
a
#
b
C
C
1
1
a
#
b
C
a) Grammar with redundancy
A
a
#
b
C
C
1
1
a
#
b
C
b) Equivalent grammar without redundancy

Fig. 14. Redundancy elimination

4.3. *Removing infinite branching*

This subsection shows how a grammar $\mathscr{F}\mathscr{B}\mathscr{G}_P$ generating a graph with finite branching can be computed from a grammar $\mathscr{G}_P$.

For any looping rule (X, Y) of $\mathscr{G}_P$, if there is a vertex v kept throughout the successive rewritings of Y $(\exists (X, red) \in App_X \mid red \neq \emptyset)$, then any edge originating from v creates an infinite branching. Therefore, edges leaving v must be deleted from R, and vertex v can be removed from any hyperarc labelled by X in the grammar, (since no edge will ever be connected to v, it becomes useless). Such a vertex will be called an *infinite branching vertex* (or ib-vertex, for short).

For the example in Figure 15-a, the App relation is:

$$App_A = \{(B, \emptyset); (C, \emptyset)\} App_B = \{(C, \{1\})\} App_C = \{(C, \{2\})\}$$

As a causality edge starts from vertex 2 of rule C, there is an infinite branching. The grammar of Figure 15-b is obtained by removing this vertex.

Theorem 4.1. Removing all ib-vertices for a grammar $\mathscr{G}$ suppresses all infinite branchings in the covering graph $\mathscr{G}^\omega$.

Proof. From the definitions of Section 4.1, we know that infinite branching appears in the graph if there is a loop $(X \in App_X)$, and an instance that is active in one branch of the choice and inactive in another. As long as an instance performs no action, the vertex

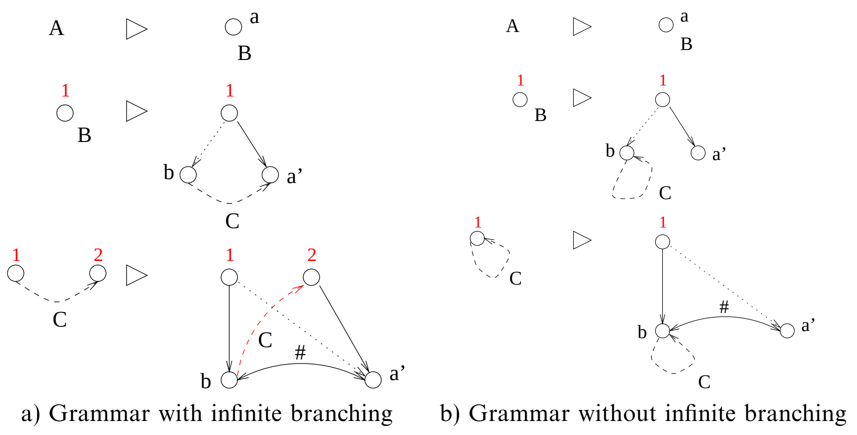

A
a
B
1
B
1
b
a'
C
1
2
C
1
2
C
#
b
a'
a) Grammar with infinite branching
A
a
B
1
B
1
b
a'
C
1
C
1
#
b
a'
C
b) Grammar without infinite branching

Fig. 15. Removing infinite branching

corresponding to its last action is kept by rewritings, and so $\exists (X, red) \in App_X \mid red \neq \emptyset$. $\square$

Theorem 4.2. Removing ib-vertices only affects infinite branchings (inheritance edges are not modified).

Proof. By construction, inheritance edges do not generate infinite branching (a vertex can be used only once as the origin of inheritance edges). So, the edges connected to an ib-vertex can only be causality edges. $\square$

4.4. *Removing end-vertices from hyperarcs*

An *end-vertex* in a rule $r = (X, Y)$ is a vertex v such that $\forall n \in \mathbb{N}, \nexists (v, v') \in \mathcal{G}^n(X)$. An end-vertex represents the last action performed by an instance in a specification. Therefore, this instance must not be taken into account in the rest of the grammar, and can be removed from the rules.

For each rule $r = (P_{con}.E, G) \in \mathcal{G}$, let E' be a set of end-vertices of E. A rule $r' = (P_{con}.E', G_{/E-E'})$ is generated from r. Let us call $\mathcal{RE}(\mathcal{G})$ the grammar $\mathcal{G}$ from which all end-vertices have been removed. It is obvious that $\mathcal{RE}(\mathcal{G})^\omega = \mathcal{G}^\omega$.

4.5. *Connection preservation*

Removing ib-vertices does not create new connected components in the graph. Infinite branching takes place between an origin vertex s_o, and an infinite set of successors, $\{s'_{o_i}\}, i \in \mathbb{N}$, located on the same instance $(\forall i \in \mathbb{N}, \phi(s_o) = \phi(s'_{o_i}))$.

The causality relation between s_o and one of its successors s'_{o_i} cannot be due to a message. Effectively, standard Z.120 specifies that a message must be sent and received within the same bMSC (unless constructs like gates are used, but these are not considered within our approach). Consequently, we cannot have a message sent once only but with multiple receptions.

Figure 16 illustrates a typical shape of covering graph containing infinite branching. Let (c_{i-1}, c_{i+1}) be an edge removed by the transformation such that c_i and c_{i+1} are events involved in minimal conflicts. According to the definition of infinite branching, there is loop, in which a branch enables an event c_i such that $\phi(c_i) = \phi(c_{i-1})$ and another branch in which $\phi(c_i)$ is inactive. Therefore, there exists an event d_i such that $\phi(d_i) \neq \phi(c)$. d_i becomes a minimal event for the inheritance relation, and any event e such that $\phi(d_i) \neq \phi(e)$ that is minimal at the occurrence $i+1$ of the choice is connected to d_i by an inheritance edge. As event c_i can also be chosen at occurrence i of the choice, there is a conflict edge between c_i and d_i, and there is a causality edge from c_{i-1} to c_i. Therefore, even after removing the edge (c_{i-1}, c_{i+1}) , c_{i-1} and c_{i+1}, remain connected.

Now, consider an edge (s_{i-1}, s_{i+1}) such that s_i and s_{i+1} are not minimal events for conflicts. Then s_i and s_{i+1} are connected to minimal events c_i and c_{i+1}, and remain attached to the rest of the graph.

Thanks to inheritance and conflict edges, any successor of a vertex remains connected after the infinite branching is removed. In Figure 16, it is easy to see that the graph remains connected even if causality edges originating from infinite branching vertices are discarded.

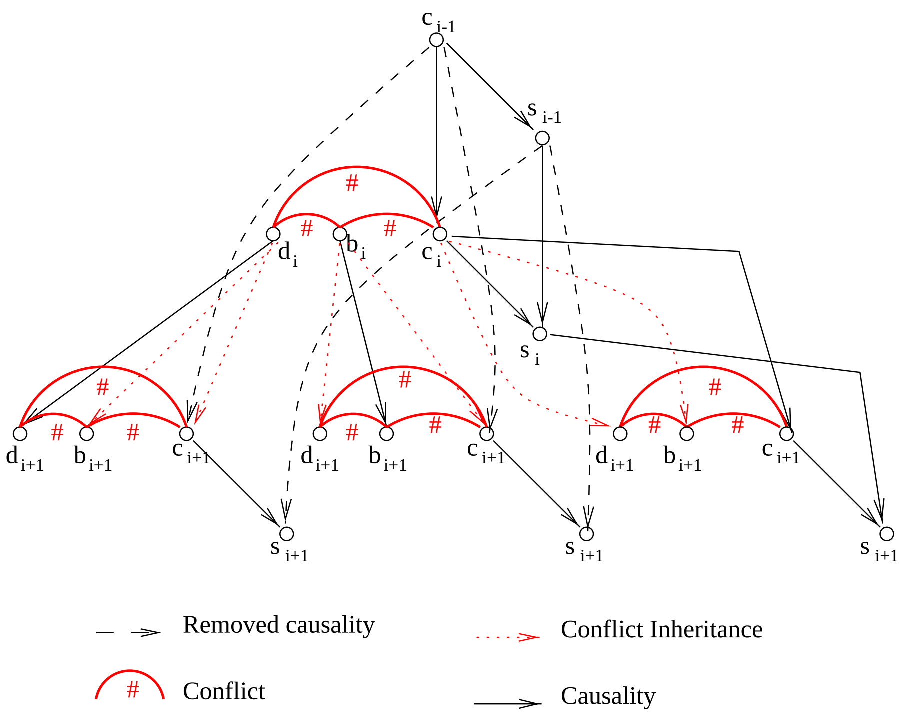


Fig. 16. Connexity preservation

Origin vertices of infinite branchings also remain connected after removing the incriminated edges. Such a vertex can be:

— a vertex generated inside the loop, in which case, it is also a successor vertex, and is connected to preceding vertices by the reasoning developed above.
— a vertex that appears outside the looping rule. In this case, the first occurrence of a choice is described by a non-looping set of rules $\{(X_i, Y_i)\}$ without inheritance edges (see for example rule B in Figure 15). The next occurrences of that choice are expressed by a set of looping rules $\{(B_i, S_i)\}$, which might be modified. As the rules $\{(X_i, Y_i)\}$ do not loop, they will not be modified, and any vertex generated by one of these rules will remain connected to the rest of the graph.

So, if a covering graph is connected, it remains connected after removing infinite branchings, and a normal form computation can be performed. If a covering graph is not connected, it means that at least two subsets of instances run concurrently without exchanging a message. They can be treated separately.

4.6. *Properties of the covering graphs*

The covering graph generated from an HMSC contains redundant information. Thanks to the inheritance and conflict edges, the structure of the HMSC can still be deduced after removing infinite branching. Therefore, the full covering graph can be reconstructed.

Let $CG =< E, \longrightarrow, \leadsto, \sharp >$ be a covering graph. A *link* from an event e_0 to an event e_n is a word $w = e_0 ... e_n \in E^n$ such that $\forall (i \neq j) e_i \neq e_j \wedge (e_i, e_{i+1}) \in (\longrightarrow \cup \leadsto \cup \sharp)$.

More intuitively, a link is an acyclic directed path of the covering graph CG (cycles may be introduced if a conflict is used as an edge in both directions).

A *causal link* of CG is a link $w = e_0 ... e_n \in E^n$ such that:

— $\phi(e_0) = \phi(e_n)$,
— $(e_0, e_1) \notin \sharp$,
— $\forall e_i, i \in 1..n-1, \phi(e_i) = \phi(e_0) \Longrightarrow (e_i, e_{i+1}) \in \sharp$.

A causal link defines a path in the covering graph of the form $\{\{\longrightarrow + \leadsto\}^+\{\sharp\}^*\}^*$

Theorem 4.3. Let $CG =< E, \longrightarrow, \leadsto, \sharp >$ be a covering graph. If $w = e_0 ... e_n \in E^n$ is a causal link of CG, then $e_0 \longrightarrow e_n$.

Proof. From the structure of HMSCs, we know that if $e \overset{*}{\leadsto} e'$, then e, e' and any event e_i situated between them are connected by a causality or inheritance edge belonging to a common scenario. If $e \sharp e'$, the conflict extends to successors but predecessors of e and e' are not in conflict, either with e and its successors or e' and its successors. Furthermore, if for each e_i, $i \in 1..n-1$, we have $\phi(e_i) = \phi(e_0) \Longrightarrow e_i \sharp e_{i+1}$, then e_n is the first event contained in the same scenario as e_0 such that $\phi(e_n) = \phi(e_0)$. So, $e_0 \longrightarrow e_n$. □

Let $Brsucc(e)$ be the set of events causally dependent on e up to the next choice.

$$Brsucc(e) = \{e' \mid e = e_1 \longrightarrow e_2 ... \longrightarrow e_n = e' \wedge \forall i \in 2..n, \nexists x, x \sharp e_i\}.$$

Theorem 4.4. Let e and e' be two events from $\mathcal{G}^w(Axiom)$ such that:

— there is a causal link $w = e.e_1 ... e_n.e'$ from e to e'.
— e and e' are minimal events for a conflict.

Then $\forall v \in Brsucc(e), \forall v' \in Brsucc(e'), \phi(v) = \phi(v') \Longrightarrow v \leqslant v'$

Proof. The statement is obvious from the structure of the HMSC. e and e' are minimal events for branches of a choice, and, furthermore, as there is a causal link from e to e', we have that e and all the events on the same branch precede e' and $Brsucc(e')$ in a path of the HMSC. □

Theorem 4.5. Any causality edge (s, s') removed by the infinite branching reduction can be reconstructed from the causal links of the graph.

Proof. If (s, s') is removed by infinite branching reduction, there is a choice between n branches $B_1..B_n$ such that $\phi(s)$ is not active in $B_1..B_n$ and a branch B such that $\phi(s)$ is active in B. Two cases can appear:

— s and s' are minimal events for the choice (consider edge (c_{i-1}, c_{i+1}) in Figure 16). Then there is a causal link from s to s', which can be replaced by a causality edge (Theorem 4.3).

— s or s' is not minimal for the choice (consider edge (s_{i-1}, s_{i+1}) in Figure 16). It is obvious that there are events x and x' minimal for the choice, and such that $s \in Brsucc(x)$ and $s' \in Brsucc(x')$. As s and s' must remain connected, there is a link from x to x'. From Theorem 4.4, we know that $\leqslant$ can be reconstructed, and therefore $\longrightarrow$ can also be reconstructed. □

4.7. *Isomorphism preservation*

We will now show that removing infinite-branching vertices does not affect the isomorphism decision.

Proof. Let $\mathscr{G}_1$ and $\mathscr{G}_2$ be two graph grammars without redundancies calculated from HMSCs. We want to show that $\mathscr{G}_1^\omega \equiv \mathscr{G}_2^\omega \Longleftrightarrow \mathscr{FBG}_1^\omega \equiv \mathscr{FBG}_2^\omega$.

— $\mathscr{G}_1^\omega \equiv \mathscr{G}_2^\omega \Longrightarrow \mathscr{FBG}_1^\omega \equiv \mathscr{FBG}_2^\omega$?
We know that infinite branching suppression does not affect vertices, conflicts, or inheritance edges on a covering graph. So, if $\mathscr{FBG}_1^\omega \not\equiv \mathscr{FBG}_2^\omega$, then there exists a causality edge within $\mathscr{FBG}_1^\omega$ that does not belong to $\mathscr{FBG}_2^\omega$ (or conversely). Such a causality edge is generated by a rule (X_1, Y_1) that does not loop in $\mathscr{G}_1$, and by a rule (X_2, Y_2) that loops within $\mathscr{G}_2$. As $\mathscr{G}_1^\omega \equiv \mathscr{G}_2^\omega$, and as (X_2, Y_2) loops, we know that (X_1, Y_1) and (X_2, Y_2) represent the same part of a recursion in an HMSC. But as (X_1, Y_1) does not loop, either this rule adds a new active instance within the HMSC, or it is the first occurrence of a choice.

– If (X_1, Y_1) adds a new active instance to the HMSC, then, as (X_2, Y_2) loops, no instance is added to the HMSC at this time, so $\mathscr{G}_1^\omega \equiv \mathscr{G}_2^\omega$ cannot hold.

– If (X_1, Y_1) is the first occurrence of a choice, then the next occurrences of this choice will have to include inheritance edges. As (X_2, Y_2) loops, it cannot represent the same pattern as (X_1, Y_1), and we cannot have $\mathscr{G}_1^\omega \equiv \mathscr{G}_2^\omega$.

This proves $\mathscr{G}_1^\omega \equiv \mathscr{G}_2^\omega \Longrightarrow \mathscr{FBG}_1^\omega \equiv \mathscr{FBG}_2^\omega$.

— $\mathscr{G}_1^\omega \equiv \mathscr{G}_2^\omega \Longleftarrow \mathscr{FBG}_1^\omega \equiv \mathscr{FBG}_2^\omega$?

From Theorem 4.5, we know that any removed causality edge can be reconstructed using causal links. It follows that $\mathscr{FBG}_1^\omega \equiv \mathscr{FBG}_2^\omega \Longrightarrow \mathscr{G}_1^\omega \equiv \mathscr{G}_2^\omega$. □

4.8. *A normal form for HMSCs*

This section describes a normal form calculus for a deterministic finite branching graph grammar.

4.8.1. *Standard form grammars* The normalisation algorithm first transforms a deterministic finite branching graph grammar into a standard form grammar. This section first recalls the definition of standard form, and shows that a finite branching grammar obtained from an HMSC can be transformed into a standard form grammar by unfolding rules.

A grammar $\mathscr{G}$ is *proper* if, $\forall (X, Y) \in \mathscr{R}$, no event of X is removed by rewriting.

A grammar $\mathscr{G}$ is *normal* if, $\forall (X, Y) \in \mathscr{R}$, we have $\forall h$ hyperarc of Y, X and h have no common vertices.

A grammar $\mathscr{G}$ is *separated* if, $\forall (X, Y) \in \mathscr{R}$, we have $\forall h_1, h_2$ hyperarcs of Y, h_1 and h_2 have no common vertices.

A grammar $\mathscr{G}$ is said to be *in standard form* if it is proper, normal and separated.

Let $\mathscr{G} = (Axiom, \mathscr{R})$ be a graph grammar calculated from an HMSC with infinite branching and end-vertices removed.

Theorem 4.6. $\mathscr{G}$ is proper.

Proof. The statement is obvious from the construction method, since every vertex added is a terminal one. □

Theorem 4.7. $\forall r = (X, Y) \in \mathscr{R}$, r can be transformed into a normal rule by a finite number of rewritings of Y.

Proof. As infinite branching and end vertices have been removed, any vertex kept in a rule will eventually have a successor. So, for each rule (X, Y) of $\mathscr{G}$, there is a minimal number of unfoldings n such that $(X, \mathscr{G}^n(Y))$ is a normal rule. □

Theorem 4.8. $\forall r = (X, Y) \in \mathscr{R}$, r can be transformed into a separated rule by a finite number of rewritings of Y.

Proof. A rule containing two hyperarcs h_1, h_2 such that $h_1 \cap h_2 \neq \emptyset$ represents a choice in the HMSC. The common vertices of h_1 and h_2 are also vertices of X. As we have seen that vertices that are kept all along a loop are removed (infinite branching and end vertices are discarded), if such a vertex is kept, it cannot be a looping vertex, and therefore must have a successor after a finite number of rewritings. □

The computation of a standard form for our grammar is straightforward: it consists of unfolding the rules until they are normal and separated. Because of the tree-like structure of our covering graphs, the algorithm is simpler than Caucal's algorithm (Caucal 1992). As the standard form calculus is performed by rewritings, it is obvious that the standard grammar generates the same covering graph as the non-standard one.

4.8.2. *Normalised grammars* A grammar $\mathcal{G} = (Axiom, \mathcal{R})$ is said to be *normalised* if and only if:

— it is in standard form,

— it is *uniform*: $\forall (X, Y) \in \mathcal{R}$, any vertex of Y added by rewriting is connected to a vertex of X by an edge.

A uniformisation algorithm is provided in Caucal (1992), the proposal being to generate uniform rules from connected sets situated at the same distance. Let us define a metric on the covering graph $\mathcal{G}^{\omega}(Axiom)$. The *root* of a covering graph is the set of minimal events for the relation $(\longrightarrow \cup \rightsquigarrow)$. The *distance* $d(v)$ of a vertex v is the least number of edges connecting v to any vertex of the root. We will use G_n to denote the covering graph restricted to vertices of distance strictly less than n.

$$G_n = \mathcal{G}^{\omega}(Axiom)_{/\{s|dist(s)<n\}}.$$

For a given set S of vertices, let $Conn(S, G)$ be set of vertices connected to S in G.

$$Conn(S, G) = \{v \in G | \exists v' \in S \wedge \big((v, v') \in (\longrightarrow \cup \dashrightarrow \cup \sharp) \vee \exists (v', v) \in (\longrightarrow \cup \dashrightarrow \cup \sharp)\big)\}$$

Given a standard form, deterministic and finite branching grammar $\mathcal{G}$, we can compute a normalised grammar. The normalisation algorithm consists of generating from $\mathcal{G}$ a uniform grammar $\mathcal{G}' = (root, \mathcal{R}')$, that is, a grammar such that $\mathcal{R}' = \{r_i = (X_i, R_i)\}$, and:

— $\forall i, X_i$ is a set of vertices that are situated at the same distance n and connected in $\mathcal{G}^{\omega}(Axiom) - G_n$.

— $\forall i, R_i$ is the restriction of $\mathcal{G}^{\omega}(Axiom) - G_n$ to $X_i \cup Conn(X_i, \mathcal{G}^{\omega}(Axiom) - G_n)$.

— $\mathcal{G}'^{\omega}(root) = \mathcal{G}^{\omega}(axiom)$.

The main idea of the algorithm is to find the set $\{X_i\}$ on a limited unfolding of the grammar, since $\{R_i\}$ is calculated from $\{X_i\}$. The complete uniformisation algorithm and a termination proof can be found in Caucal (1992).

4.9. *A decision procedure for HMSC equivalence*

Section 3 showed that it is possible to generate a graph grammar defining a unique event structure from an HMSC. This graph grammar may contain infinite branchings. Section 4 has shown that a modification of this grammar allows infinite branching to be eliminated while preserving isomorphism. In Section 4.8 we described an algorithm for calculating a normal form from a finite branching deterministic graph grammar. This provides us with a decision algorithm for the equivalence of two HMSCs, called P_1 and P_2 hereafter:

— Compute $\mathcal{G}_{P_1}$ and $\mathcal{G}_{P_2}$, the graph grammars of P_1 and P_2, using the predicate *Rule*.

— Compute $\mathcal{G}'_{P_1}$ and $\mathcal{G}'_{P_2}$ by replacing the inheritance edges that are not made from choice to choice by the corresponding conflicts within $\mathcal{G}_{P_1}$ and $\mathcal{G}_{P_2}$.

— Compute $\mathcal{G}''_{P_1}$ and $\mathcal{G}''_{P_2}$, by eliminating redundancies.

— Compute $\mathscr{FBG}_1$ and $\mathscr{FBG}_2$ by eliminating infinite branchings and end-vertices within $\mathscr{G}''_{P_1}$ and $\mathscr{G}''_{P_2}$.
— Compute $\mathscr{FNG}_{P_1}$ and $\mathscr{FNG}_{P_2}$, the normal forms of $\mathscr{FBG}_{P_1}$ and $\mathscr{FBG}_{P_2}$, by unfolding rules.
— Compute $\mathscr{NG}_{P_1}$ and $\mathscr{NG}_{P_2}$, the normalised grammars for $\mathscr{FNG}_{P_1}$ and $\mathscr{FNG}_{P_2}$.

If P_1 and P_2 have the same normalised forms up to a renaming, then they are equivalent. Let us compare HMSC P_3 in Figure 17 with HMSC P_2 in Figure 5. The normal forms of $\mathscr{G}_{P_1}$ and $\mathscr{G}_{P_2}$ are the same, and generate the covering graph of Figure 18. Therefore, we can state that P_3 is isomorphic to P_2.

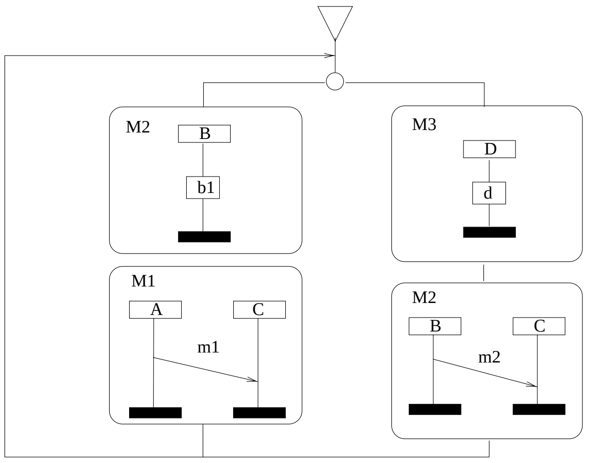


Fig. 17. HMSC P_3

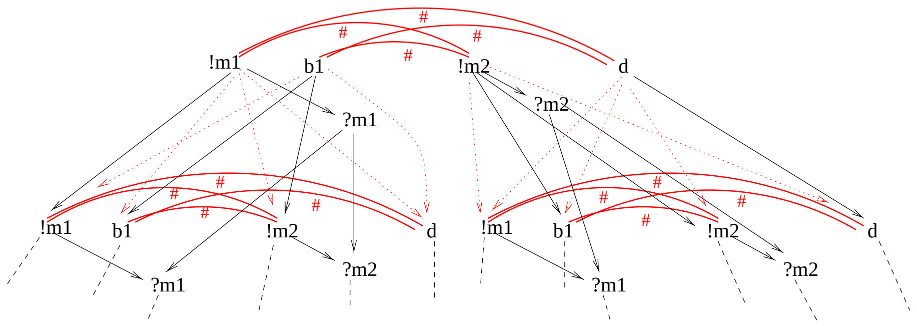


Fig. 18. Covering graph generated by the normal form

5. Conclusion

We have given a prime event structure semantics of HMSCs, and then defined rules for computing a graph grammar that generates a covering graph of this structure.

This formal representation of HMSCs has several advantages. First, it preserves the partial order semantics described in specifications (event independence, as well as weak sequential composition of bMSCs), which appears to be the main advantage of the notation. By contrast, an interleaving semantics would fail to preserve event independence. Then, the semantics defined in this paper places emphasis on conflicts. Choices in an HMSC define important control points in the specification, and should be implemented as a consensus, or a synchronisation for the case of a distributed choice. Finally, using a graph semantics for HMSCs allows us to maintain a visual and intuitive representation. Furthermore, a set of well-known algorithms on graphs is made available for HMSCs, among these a graph isomorphism decision for regular graphs.

A covering graph isomorphism defines the equivalence between two partial order families described by means of HMSCs, and can be decided through the computation of a normal form for graph grammars. This equivalence is very discriminating, as it takes into account the control structure expressed by choices in an HMSC. Therefore, it differentiates specifications in Figure 3-a and Figure 3-b, detecting that the choice of a scenario is not performed at the same moment.

A weaker equivalence can be defined from the same model. By determinising the grammar (reporting conflicts $e \sharp e'$ such that e and e' have equal predecessor sets and $\alpha(e) = \alpha(e')$ to successors of e and e') before applying the normal form calculus. After determinising, information about the location of choices is not considered, and the equivalence corresponds to the isomorphism of partial order families. The identification of classes of MSCs that can be determinised, and of a determinisation algorithm based on rule unfolding and conflict merging is currently under study.

The representation of HMSCs by means of graph grammars and event structures makes it possible to treat very abstract specifications of distributed systems formally. It can be considered as another argument for the use of formal methods even on incomplete models at early stages of specification.

References

Ben-Abdallah, H. and Leue, S. (1997) Syntactic Detection of Process Divergence and non-Local Choice in Message Sequence Charts. In: Brinksma, E. (ed.) Proceedings of the Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems TACAS'97. *Springer-Verlag Lecture Notes in Computer Science* **1217** 259–274.

Caillaud, B., Darondeau,P., Hélouët, L. and Lesventes, G. (2000) HMSCs as partial specifications...with Petri Nets as completion. INRIA research report no 3970. `ftp://ftp.inria.fr/INRIA/publication/publi-ps-gz/RR/RR-3970.ps.gz`

Caucal, D. (1992) On the regular structure of prefix rewriting. *Theoretical Computer Science* **106** (1) 61–86.

Grabowski, J., Graubman, P. and Rudolph, E. (1993) Towards a Petri net based semantics definition for Message Sequence Charts. In: Faergemand, O. and Sarma, A. (eds.) *SDL'93 - Using Objects. Proceedings of the sixth SDL Forum*, North-Holland 179–190.

Habel, A. (1989) Hyperedge replacement: grammars and graphs. In: Goos, G. and Hartmanis, J. (eds.) *Springer-Verlag Lecture Notes in Computer Science* **643**.

Harel, D. and Damm, W. (1998) LSCs: breathing life into message sequence charts. Weizmann Institute Tech. Report CS98-09.

Holzmann, G. J. (1997) The Spin Model Checker. *IEEE Trans. on Software Engineering* **23** (5) 279–295.

ITU-T (1996) Message Sequence Chart (MSC). *ITU-T Recommendation Z120.*

Heymer, S. (1998) A non-interleaving semantics for MSC. *Proceedings of SAM98:1st conference on SDL and MSC* 281–290.

Katoen, J. P. and Lambert, L. (1998) Pomsets for message sequence charts. *Proceedings of SAM98:1st conference on SDL and MSC* 291.

Kosiuczenko, P. (1997) Formalizing MSC'96: Inline expressions. Technical report, Insitute fur Informatik, Munich.

Leue, S. and Ladkin, P. (1994) Four issues concerning the semantics of message flow graphs. Technical report, INRIA Lorraine.

Leue, S. and Ladkin, P. (1995) Interpreting message flow graphs. *Formal Aspects of Computing* **7** (5) 473–509.

Leue, S. and Ladkin, P. (1994) What do message sequence charts mean? Proceedings of FORTE'93 Formal Description Techniques VI. *IFIP Transactions C*, North-Holland 301–315.

Mauw, S. (1996) The formalization of message sequence charts. *Computer Networks and ISDN Systems* **28** (12) 1643–1657.

Muscholl, A., Peled, D. and Su, Z. (1998) Deciding properties for Message Sequence Charts. *Springer-Verlag Lecture Notes in Computer Science* **1378** 226–241.

OMG (1997) Unified Modelling Language 1.1.

Pratt, V. (1986) Modelling Concurrency with Partial Orders. *International journal of Parallel Programming* **15** (1) 33–71.

Reniers, M. and Mauw, S. (1994) An algebraic semantics for basic message sequence charts. *The Computer Journal* **37** (4) 269–277.

Reniers, M. and Mauw, S. (1996) High-level message sequence charts. Technical report, Heindoven University of Technology.

Reniers, M. (1998) Message Sequence Charts: Syntax and Semantics, Ph. D. Thesis, Heindhoven University of Technology.

Rensink, A. and Wehrheim, H. (1994) Weak Sequential Composition in Process Algebras. In: Jonsson, B. and Parrow, J. (eds.) CONCUR '94: Concurrency Theory, 5th International Conference. *Springer-Verlag Lecture Notes in Computer Science* **836** 226–241.

Rudolph, E., Graubmann, P. and Grabowski, J. (1996) Tutorial on Message Sequence Charts (msc'96), *Computer Networks and ISDN Systems* **28** (12) 1629–1641.

Winskel, G., Nielsen, M. and Plotkin, G. (1981) Petri nets, event structures and domains, Part 1. *Theoretical Computer Science* **13** (1) 85–108.