Under consideration for publication in Math. Struct. in Comp. Science

An Event Structure Based Semantics for High-Level Message Sequence Charts

Loïc Hélouët*, Claude Jard*, Benoît Caillaud**

* *CNRS*, ** *INRIA*

IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

{helouet,jard,bcaillau}@irisa.fr

Received 13 July 2000

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

1. Introduction

1.1. *Scenario-based Languages*

An increasing attention is paid towards the supervision and the formalization of the very first phases of design in software development. The passage from informal to formal descriptions appears a crucial stage. Formal techniques often neglected this aspect, standing in margin of methodological aspects. The appearance of scenario-based languages is a response to this deficiency, in particular in the context of distributed system modeling.

Scenarios consist in 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. Therefore, they help concentrating on causality or independence of events, the two cornerstones of distributed system modeling. Albeit informal, they are precise enough to contain information about processes, message types, ...

Scenarios are widely used in telecommunication standards, where they help documenting a model, and provide a global understanding of distributed protocols. They are also used to present simulation traces or to specify properties, as in the SDL tools SDT (S-DT) or Object-Géode (ObjectGeode), or in the verification tool SPIN (Holzmann97) for instance.

The most prominent formal notation based on scenarios is the "Message Sequence Charts" (MSCs) notation. MSCs are mostly used for describing the activities of communicating entities in distributed systems. They can be found in a closely related formalism in the Unified Modeling Language (OMG97), where they are called Sequence Diagrams (a convergence between these two notations is expected in a near future). A MSC graphically describes communications between processes (called *Instances*). MSCs give a very intuitive and visual representation of systems behaviors. Other scenario based notations have been proposed, among which LSCs (Harel98) allow for the specification of "mandatory" or "optional" scenarios.

Even though, scenarios are not used at all stages of development, and often remain confined 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 which follows this idea is currently being standardized, and is called "High level MSC" or HMSC (Rudolph96). HMSCs define compositions of MSCs with operators such as sequencing, choice, environmental composition, hierarchical composition. A HMSC is a graph, which nodes can be starting nodes, end nodes, choices, MSCs, or references to other HMSCs. In order to keep the essential, this article 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 (Grabowski93; Reniers94; Mauw96; Leue94a; Leue94b; Leue94c; Kosiuczenko97), but the semantics of 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 (Reniers96; Reniers98), and is now considered as the "official" semantics of HMSCs. More recently, (Katoen98; Heymer98) defined partial-order semantics.

One of the main advantage of MSCs is that independence of events is graphically represented. 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 with a dash 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. This is the usual argument for a partial order semantics.

A second aspect concerns the semantics of concatenation (sequential composition, noted $M1; M2$ hereafter) of two MSCs M_1 and M_2. Many tools use a strong sequential composition which, contrarily 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 synchronization barrier between each pattern of behavior 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 using a weaker sequential composition must allow to describe infinite behaviors with possibly unbounded communication queues and asynchrony between events. Problems of this kind

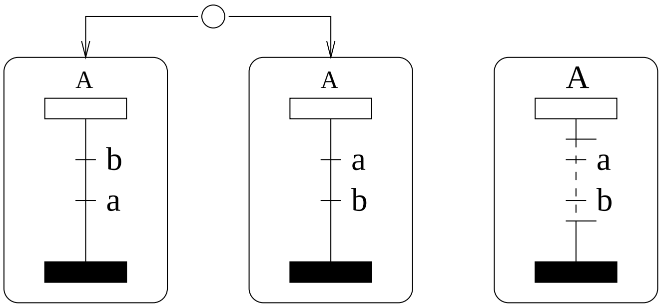


Fig. 1. Example MSCs: a) non-determinism b) potential parallel execution

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.

Semantics of choices is the third point. The representation of system behaviors with 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. As the decision to perform M_2 or M_3 is distributed, this cannot be done without a synchronization 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 detecting such problems, and emphasize choices in the specification and then distinguish between the specifications given in Figure 3-b and Figure 3-b.

Prime event structures (Winskel81) are such a model that both uses a non-interleaving semantics, and can help underlining choices. Of course, these models can be infinite when representing and 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 (Muscholl98). 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 to decide in polynomial time whether two graphs are isomorphic or not in the case of deterministic graph grammars generating finite branching connected graphs (Caucal92). 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 Article*

This article is organized as follows: section 2 introduces the Message Sequence Charts notations, 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 graphically define distributed systems and communications between the components of these systems. The purpose of this section is to provide the reader with the features and vocabulary we will use in the rest of the article rather than providing a complete description of MSCs, which can be found in the ITU standard Z.120 (ITU96) and in (Reniers98).

2.1. *bMSCs*

A bMSC (standing for basic MSC) defines a simple scenario, e.g. an abstraction of a system behavior. Within bMSCs, processes are called *instances*, and are represented by a vertical axis, along which events are put in a top down order. Message exchanges are represented by arrows labeled 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 upper than 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.

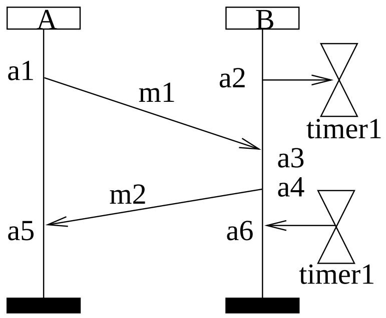


Fig. 2. An example bMSC.

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

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

HMSCs allows 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. A HMSC can also describe an infinite behavior, as in Figure 5.

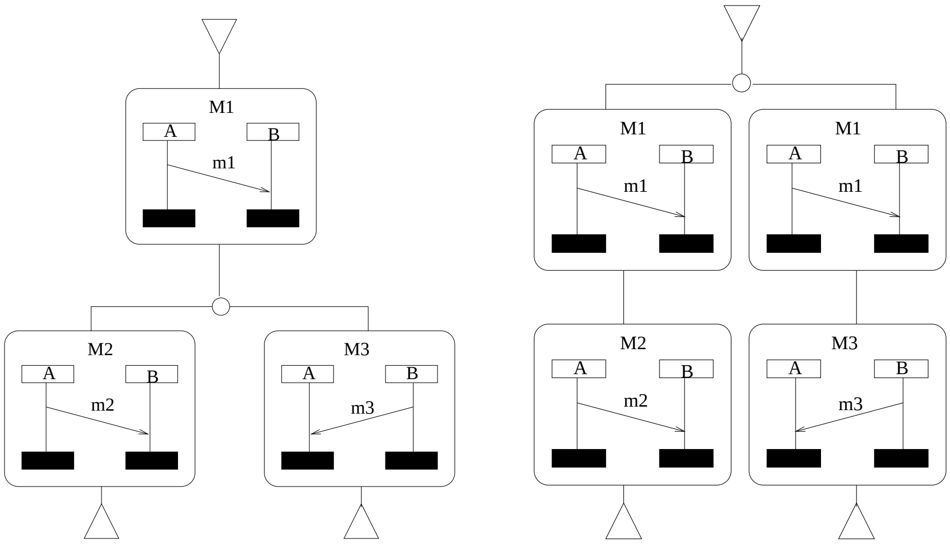


Fig. 3. a) HMSC P_0 (with non-local choice) b) HMSC P_1

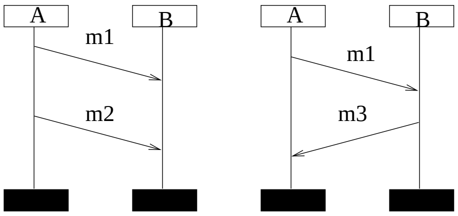


Fig. 4. Scenarios defined by the HMSCs in Figure 3-a and Figure 3-b

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 $M \in \mathcal{M}$ is a bMSC, ; is a sequence operator, ϵ denotes the end of a 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

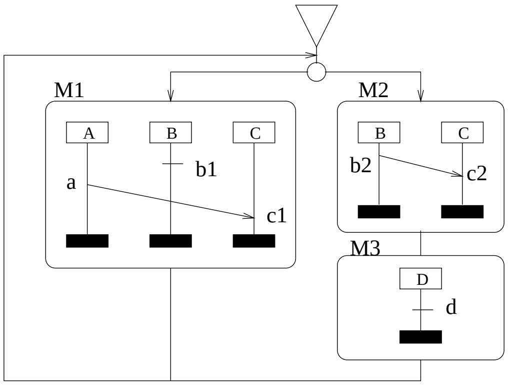


Fig. 5. HMSC P_2

instance is said to be *active* in a branch if it performs at least an action in this branch. The expression associated to the HMSC in Figure 5 is $P_1 ::= recX.(M1; X + M2; M3; X)$.

2.3. *Partial Order Semantics for MSCs*

Let us consider a bMSC M describing the behavior of a finite set of instances, called I hereafter. 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 MSC defines a partial order between events. In Figure 2, a_1 and a_4 are ordered, but a_1 and a_2 are not. The semantics of a bMSC can be formalized by a poset $< E, \leq, \alpha, A, I >$ where E is a set of events, $\leq$ is a partial order relation (reflexive, transitive and antisymmetric binary relation) on E called *causal dependence relation*, A is a set of atomic actions names, and α is a labeling function from E to $A \times I$.

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 $M_1 =< E_1, \leq_1, \alpha_1, A_1, I_1 >$ and $M_2 =< E_2, \leq_2, \alpha_2, A_2, I_2 >$ be two MSCs. The *concatenation* of M_1 and M_2, written $M_1 \circ M_2$ is defined by:

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

$$\begin{aligned} \leq_{M_1 \circ M_2} \quad = \quad & \leq_1 \cup \leq_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \\ & \exists (e'_1, e'_2) \in E_1 \times E_2 \wedge \phi_1(e'_1) = \phi_2(e'_2) \wedge e_1 \leq_1 e'_1 \wedge e'_2 \leq_2 e_2\} \end{aligned}$$

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 (Pratt86)(the same operation has been defined by (Rensink94) for process algebra), and is often referred in the MSC-related literature as the weak sequential composition.

Clearly, a 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) hereafter. POF semantics for HMSCs have been defined by (Heymer98; Katoen98). However, the POF representation abstracts the branching information at choices, which can be a very important characteristics of a HMSC as shown previously. Furthermore, POFs can not be manipulated easily, because they are infinite. We now define a finite representation of 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 a HMSC, and will allow for equivalence decision.

2.4. *An Event Structure semantics for HMSCs*

A prime event structure (ES for short) is a 6-tuple $< E, \leq, \sharp, \alpha, A, I >$ where E, A , I, $\leq$ and α have the same meaning than in section 2.3, and $\sharp$ is a symmetric anti-reflexive binary relation called *conflict relation*, such that:

$\forall e \in E, \forall e' \in E, e \sharp e' \Leftrightarrow \forall e'' \in E, (e' \leq e'' \Rightarrow e \sharp e'')$ (conflicts are inherited through causality relation).

An event structure defines a domain of *configurations*, that 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 downward-closed ($\forall e \in C, e' \leq e \Rightarrow e' \in C$).

A *maximal configuration* of an ES $S =< E, \leq, \sharp, \alpha, A, I >$ is a subset C of E such that: $\forall e \in E - C, \exists e' \in C \mid e' \sharp e$, e.g. $C \cup \{e\}$ is not a configuration.

For further reading on event structures, consult (Winskel81).

The *conflict operation* between two ESs is noted $S_1 \sharp S_2$ and is the ES:
$S_1 \sharp S_2 =< E_1 \cup E_2, \leq_1 \cup \leq_2, \sharp, \alpha_1 \cup \alpha_2 >$, such that $\sharp = (E_1 \times E_2) \cup (E_2 \times E_1) \cup \sharp_1 \cup \sharp_2$.
We generalize the notation to an arbitrary number of ES with $\underset{i \in 1..n}{\sharp} (S_i)$.

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

- $SE_0(P) =< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$
- $SE_k(M; P) = M \circ SE(P)$
- $SE_k(\underset{i \in 1..n}{\Sigma} E_i = \underset{i \in 1..n}{\sharp} (SE_{k-1}(E_i))$
- $SE_k(recX(P)) = SE_{k-1}(P_{[X:=recX.(P)]})$
- $SE_k(\epsilon) =< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$

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

2.5. *Covering graphs*

An event structure can be represented by a graph which associates a vertex to each event, and a typed edge to each conflict and each pair in the order relation. Infinite behaviors lead to infinite graphs. Unfortunately, the resulting graph is not necessarily a regular graph, which means that it can not always be generated by a graph grammar.

A first intuitive solution is to represent only minimal conflicts, and the covering of the causal order $\leq$. The other conflict and causality edges can be deducted from the graph representation, using the conflict inheritance property and the transitivity of the order relation. Unfortunately, an event can remain connected to an infinite number of events via a conflict, and our graph may still not be regular: if we try generating a graph for the event structure representation of the HMSC in Figure 6, we obtain the graph in Figure 7. By restricting this graph to events labeled by a and conflict edges, we can extract the graph in Figure 8, which is a well known irregular graph.

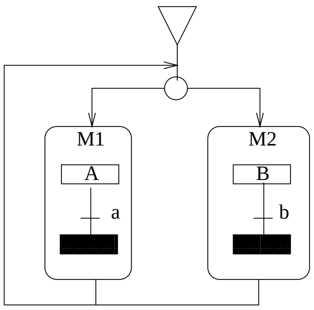


Fig. 6. HMSC with an irregular SE representation

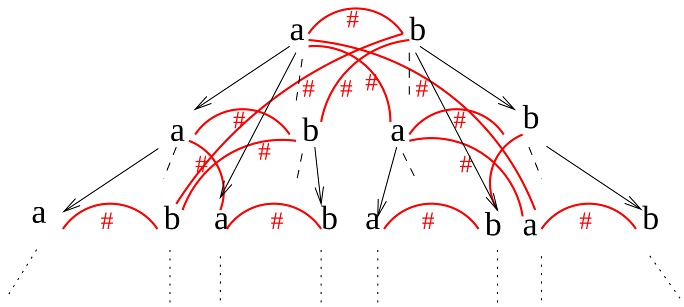


Fig. 7. Irregular SE graph for HMSC of Figure 6

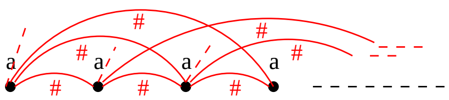


Fig. 8. An irregular subgraph extracted from the graph in Figure 7

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 conflict 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:

— $\longrightarrow = \{(e,e') \in \leq \mid e \neq e' \wedge \not\exists e'' \in E - \{e,e'\}, e \leq e'' \leq e'\}$ is the covering of $\leq$,
— $\rightsquigarrow$ is an conflict inheritance relation ($e_1 \rightsquigarrow e_2 \Rightarrow \forall e' \in E \mid e'\sharp e_1, e'\sharp e_2$),
— $\sharp_c$ is the set of minimal conflicts, i.e. $\{(e,e') \mid \not\exists e'' \wedge (e'' \leq e' \vee e'' \rightsquigarrow e') \wedge e\sharp e''\}$.

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. $\longrightarrow$ is represented by simple arrows, $\leadsto$ is represented by dotted arrows, and $\sharp$ is represented by an edge labeled by $\sharp$.

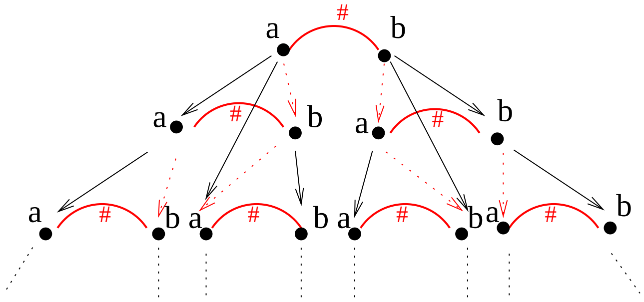


Fig. 9. Regular covering graph for HMSC Figure 6

3. Graph Grammar Generation from a HMSC

This section details how the infinite event structure semantics of a Message Sequence Charts can be expressed finitely by an hyperedge replacement graph grammar. First, a short introduction to hyperedge replacement grammars is given.

3.1. *Graph Grammars*

This section details the notation that will be used within the rest of the paper for graph grammars. For further reading, (Habel89) may be consulted.

An *hypergraph* G is a pair $(T(G), H(G))$, where $T(G)$ is a finite graph, and $H(G)$ is a set of hyperarcs. An *hyperarc* is a word $ls_1..s_n$, which label l belongs to an alphabet L, and where $s_1..s_n$ are vertices of $T(G)$. On the hypergraph represented in Figure 10-a, the vertices s_4, s_5, s_6 are linked by a hyperarc labeled by A. Such a hyperarc will then be noted $As_5s_6s_4$.

A *graph grammar* consists in a hypergraph G_0 called the *axiom* of the grammar, and of a set $\mathcal{R}$ of rewriting rules. A rewriting rule is a pair $r = (X, H)$, where X is a hyperarc called the *left part* of r, and H is the hypergraph rewritten from X's vertices, called the

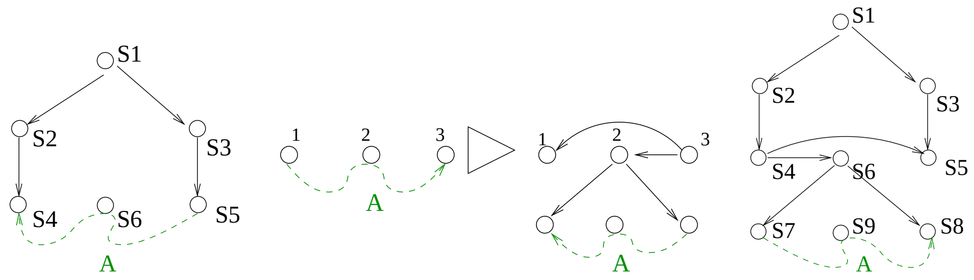


Fig. 10. a) Hypergraph H b) rule R c) unfolding of H by R

right part of r. We shall often write this rule $X \triangleright H$. An example of rule is provided Figure 10-b. For a hypergraph M, we note $M \underset{\mathcal{G},X}{\longrightarrow} N$ if M can be rewritten into N by replacing X by H in M.

A *direct derivation* of a hypergraph H by a rule $(X,R) \in \mathcal{R}$ is a hypergraph H' such that $H' = H_{[X:=R]}$. We write $H \underset{\mathcal{G},(X,R)}{\longrightarrow} H'$. A sequence of direct derivations $H \underset{\mathcal{G},(X_1,R_1)}{\longrightarrow} H_1 \underset{\mathcal{G},(X_2,R_2)}{\longrightarrow} \dots \underset{\mathcal{G},(X_n,R_n)}{\longrightarrow} H_n$ is called a *derivation of length n*. Figure 10-c shows the hypergraph obtained by derivation of hypergraph H Figure 10-a by rule R Figure 10-b.

Parallel derivation generalizes direct derivation, by enabling simultaneous hyperarc replacements. We write $M \underset{\mathcal{G}}{\Longrightarrow} N$ (N is a parallel derivation of M by $\mathcal{G}$) iff there exists a set of rules $P = (X_1,R_1)...(X_n,R_n)$ of $\mathcal{G}$, M comprises exactly n hyperarcs, and for any permutation π on $\{1..n\}$, $M \underset{\mathcal{G},(X_{\pi(1)},R_{\pi(1)})}{\longrightarrow} \dots \underset{\mathcal{G},(X_{\pi(n)},R_{\pi(n)})}{\longrightarrow} N$.

Let us note $[M]$ the projection of a hypergraph M on its terminal edges. $\mathcal{G}^w(M)$ denotes the possibly infinite graph generated from M, defined by $\mathcal{G}^w(M) = \bigcup_n [\mathcal{G}^n(M)]$, where $\mathcal{G}^0(M) = M$ and $\mathcal{G}^n(M) \underset{\mathcal{G}}{\Longrightarrow} \mathcal{G}^{n+1}(M)$.

3.2. *Operations on Covering Graphs*

Let I be a set of instances, $M \in \mathcal{M}$ a MSC, and $\mathcal{G}r(M) =< S, \longrightarrow, \rightsquigarrow, \sharp, \alpha, A, I >$ its covering graph. We define the following operations:

- For all $i \in I, S_i = \{s \in S \mid \phi(s) = i\}$.
- $Inf(\mathcal{G}r(M)) = \{s \in S \mid \forall s' \in S, (s',s) \not\in \longrightarrow\}$ is the set of minimal events of $\mathcal{G}r(M)$ w.r.t $\longrightarrow$.
- $Sup(\mathcal{G}r(M)) = \{s \in S \mid \forall s' \in S, (s,s') \not\in \longrightarrow\}$ is the set of maximal events of $\mathcal{G}r(M)$ w.r.t $\longrightarrow$.
- For all $i \in I, Inf_i(\mathcal{G}r(M)) = \{s \in S_i \mid \forall s' \in S_i, (s',s) \not\in \longrightarrow\}$ is the set of minimal events on instance i.
- For all $i \in I, Sup_i(\mathcal{G}r(M)) = \{s \in S_i \mid \forall s' \in S_i, (s,s') \not\in \longrightarrow\}$ is the set of maximal events on instance i.
- $Act(\mathcal{G}r(M)) = \{i \in I \mid \exists s \in S, \phi(s) = i\}$: set of instances that perform at least an event within M.

We will also write $S_{\mathcal{G}r(M)}$ and $\underset{\mathcal{G}r(M)}{\longrightarrow}$ for the vertices and causality edges of $\mathcal{G}r(M)$.

Graph $\mathcal{G}r(M_1)$ in Figure 11 is the covering graph of the bMSC defined in Figure 2. For this particular graph, we have the following sets:

- $S_A(\mathcal{G}r(M_1)) = \{s_1, s_5\}$,$S_B(\mathcal{G}r(M_1)) = \{s_2, s_3, s_4, s_6\}$,
- $Inf(\mathcal{G}r(M_1)) = \{s_1, s_2\}$,
- $Sup(\mathcal{G}r(M_1)) = \{s_5, s_6\}$,
- $Act(\mathcal{G}r(M_1)) = \{A, B\}$
- $Inf_A(\mathcal{G}r(M_1)) = \{s_1\}$, $Inf_B(\mathcal{G}r(M_1)) = \{s_2\}$
- $Sup_A(\mathcal{G}r(M_1)) = \{s_5\}$, $Sup_B(\mathcal{G}r(M_1)) = \{s_6\}$

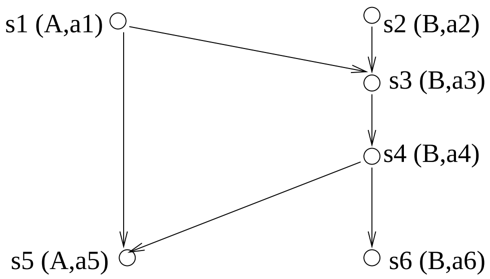


Fig. 11. Covering graph $\mathcal{G}r(M_1)$

3.3. *Deriving Graph Grammars from a HMSC*

Due to recursive definitions, infinite event structures can be defined. However, the covering graph of a HMSC can be represented finitely by a graph grammar. A graph grammar is associated to any HMSC defined by its regular expression P. The axiom of the grammar is the starting expression P, without vertex. Derivations of rules have to take into account expressions, but also a *context*, e.g. the set of active instances, the branch of the choice that is evaluated, and so on. Therefore, the rule generated from an expression will not be the same depending on the history of the system. The alphabet L of non-terminal hyperarcs is composed of sub-expressions of the axiom and contexts.

Let $P \in (\mathcal{M}, ;, \Sigma, rec, \epsilon)$ be a HMSC, S be a set of vertices, $LE(S)$ a predicate indicating if a vertex s corresponds to the last event performed by instance $\phi(s)$, Her a predicate on S 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 Br be a list of instances that performed at least an action in the branch of the HMSC currently studied.

Let us define the function $Rules(P, S, LE, Her, Br, Var) = \{(H, R)\}$, that returns the rules associated to the HMSC P according to a specific context $\{LE, Her, Br, Var\}$. Rules are of the form $P_{Con}.S \triangleright (C, \{P'_i.S'_i\})$, meaning that in the context Con, a hyperarc labeled by P rewrites into an hypergraph C including hyperarcs of the form $P'_i.S'_i$. For the sake of clarity, the computation of A, I, or α is not defined, and the context is not indicated on the labels of the rules. An illustration of grammar rules associated to any kind of subexpresions of a HMSC is provided Figure 12. The rules associated to a HMSC are computed in the following way:

— $Rules(\epsilon, S, LE, Her, Br, Var) = \epsilon.S \triangleright (< S, \emptyset, \emptyset, \emptyset, \alpha, A, I >, \emptyset)$
  This rule suppresses hyperarcs labeled by ϵ.

— $Rules(M;P, S, LE, Her, Br, Var) \;=\; M;P.S \triangleright \big(\langle S_{M;P}, \underset{M;P}{\longrightarrow}, \underset{M;P}{\rightsquigarrow}, \emptyset, \alpha, A, I, \rangle,\; P.S'\big)$
  $\cup\ Rules(P, S', LE', Her', Br \cup Act(\mathcal{G}r(M)), Var)$ where:
  - $S_{M;P} = \quad S \cup S_{\mathcal{G}r(M)}$
  - $\underset{M;P}{\longrightarrow} \;=\; \{(s, Inf_i(\mathcal{G}r(M)) \mid s \in LE(S) \wedge \phi(s) = i \wedge i \in Act(\mathcal{G}r(M))\} \cup \underset{\mathcal{G}r(M)}{\longrightarrow}$
  - $\underset{M;P}{\rightsquigarrow} \;=\; \{(s, s') \mid s \in S \wedge Her(s) \wedge s' \in Inf(\mathcal{G}r(M)) \wedge \phi(s') = i \wedge i \notin Br\}$
  - $LE' \;=\; LE \cap S' \cup \{Sup_i(\mathcal{G}r(M)) \mid i \in Act(\mathcal{G}r(M))\}$

- $S' \;=\; \{s \in LE(S) \mid \phi(s) \notin Act(\mathcal{G}r(M))\} \cup \{Sup_i(\mathcal{G}r(M)) \mid i \in Act(\mathcal{G}r(M))\}$
  $\cup \; \{s \in S \cup S_{\mathcal{G}r(M)} \mid s \in Her'\}$
- $Her' = Her \cup \{s \in Inf(\mathcal{G}r(M)) \mid \phi(s) = i \wedge i \notin Br\}$

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.

— $Rules(\sum_{k\in 1..K} M_k; P_k, S, LE, Her, Br, Var) =$

$$\sum_{k\in 1..K} M_k; P_k.S \triangleright \big(\langle S_{\sum}, \longrightarrow_{\sum}, \rightsquigarrow_{\sum}, \sharp_{\sum}, \alpha, A, I\rangle, \bigcup_{k\in 1..K} P_k.S'_k\;\big)$$

$\cup \bigcup_{k\in 1..K} Rules\big(P_k, S'_k, LE'_k, Her'_k, Act(\mathcal{G}r(M_k)), Var\big)$ where:

- $S_{\sum} \;=\; S \cup \bigcup_{k\in 1..K} S_{\mathcal{G}r(M_k)}$
- $\longrightarrow_{\sum} = \bigcup_{k\in 1..K} \big(\; \{(s, Inf_i(\mathcal{G}r(M_k))) \mid$
  $s \in LE(S) \wedge \phi(s) = i \wedge i \in Act(\mathcal{G}r(M_k))\} \cup \underset{\mathcal{G}r(M_k)}{\longrightarrow}\big)$
- $\rightsquigarrow_{\sum} \;=\; \bigcup_{k\in 1..K} \{(s, s') \mid s \in S \wedge Her(s) \wedge s' \in Inf(\mathcal{G}r(M_k)) \wedge \phi(s) \neq \phi(s')\}$
- $\sharp_{\sum} \;=\; \{(s, s') \mid s \in Inf(\mathcal{G}r(M_m)) \wedge s' \in Inf(\mathcal{G}r(M_n)) \wedge m \neq n\}$
- $S'_k \;=\; \{s \in LE(S) \mid \phi(s) \notin Act(\mathcal{G}r(M_k))\}$
  $\cup\{Sup_i(\mathcal{G}r(M_k)) \mid i \in Act(\mathcal{G}r(M_k))\} \cup \quad \{Inf(\mathcal{G}r(M_k))\}$
- $Her'_k = \; Inf(\mathcal{G}r(M_k))$
- $LE'_k \;=\; LE \cap S'_k \cup \{Sup_i(\mathcal{G}r(M_k)) \mid i \in I\}$

This rule produces the conflicts due to a choice in a HMSC. Note that predicate Her is set to $Inf(\mathcal{G}r(M_k))$, e.g. the minimal events of each branch. This ensures that no infinite branching will be generated by inheritance edges.

— $Rules(recX.(P), S, LE, Her, BR, Var) = recX(P).S \triangleright \big(\langle S, \emptyset, \emptyset, \alpha, A, I\rangle, P.S\big)$
$\cup\; Rules(P, S, LE, Her, Br, Var \cup \{(X, P)\})$

— $Rules(X, S, LE, Her, Br, Var) = X.S \triangleright \big(\langle S, \emptyset, \emptyset, \alpha, A, I\rangle, P.S \mid (X, P) \in Var\big)$
$\cup\; Rules(P, S, LE, Her, Br, Var)$

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

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

We now have 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 article defines an equivalence calculus, based on covering graphs isomorphism.

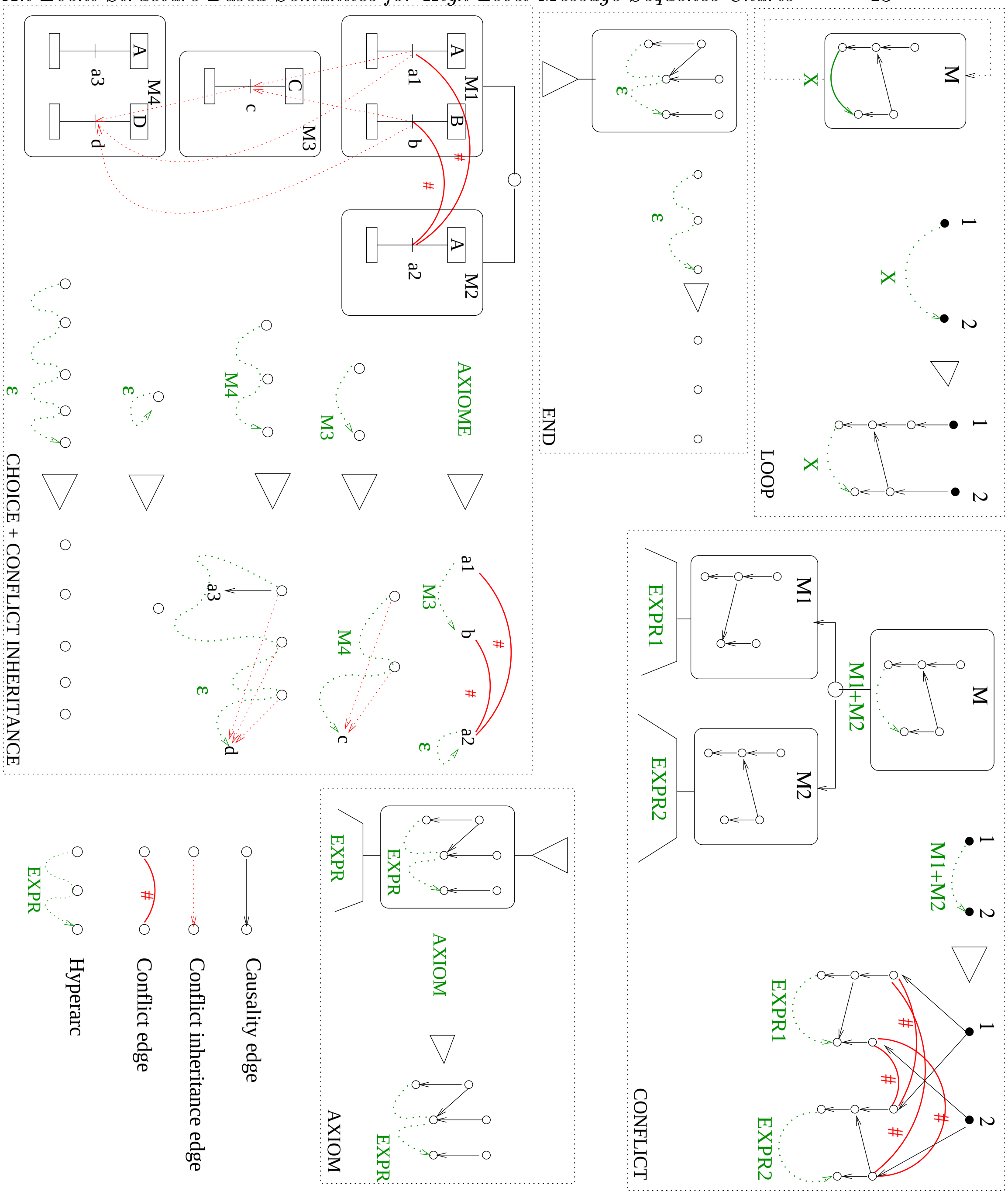


Fig. 12. Illustration of graph grammar calculus from HMSCs

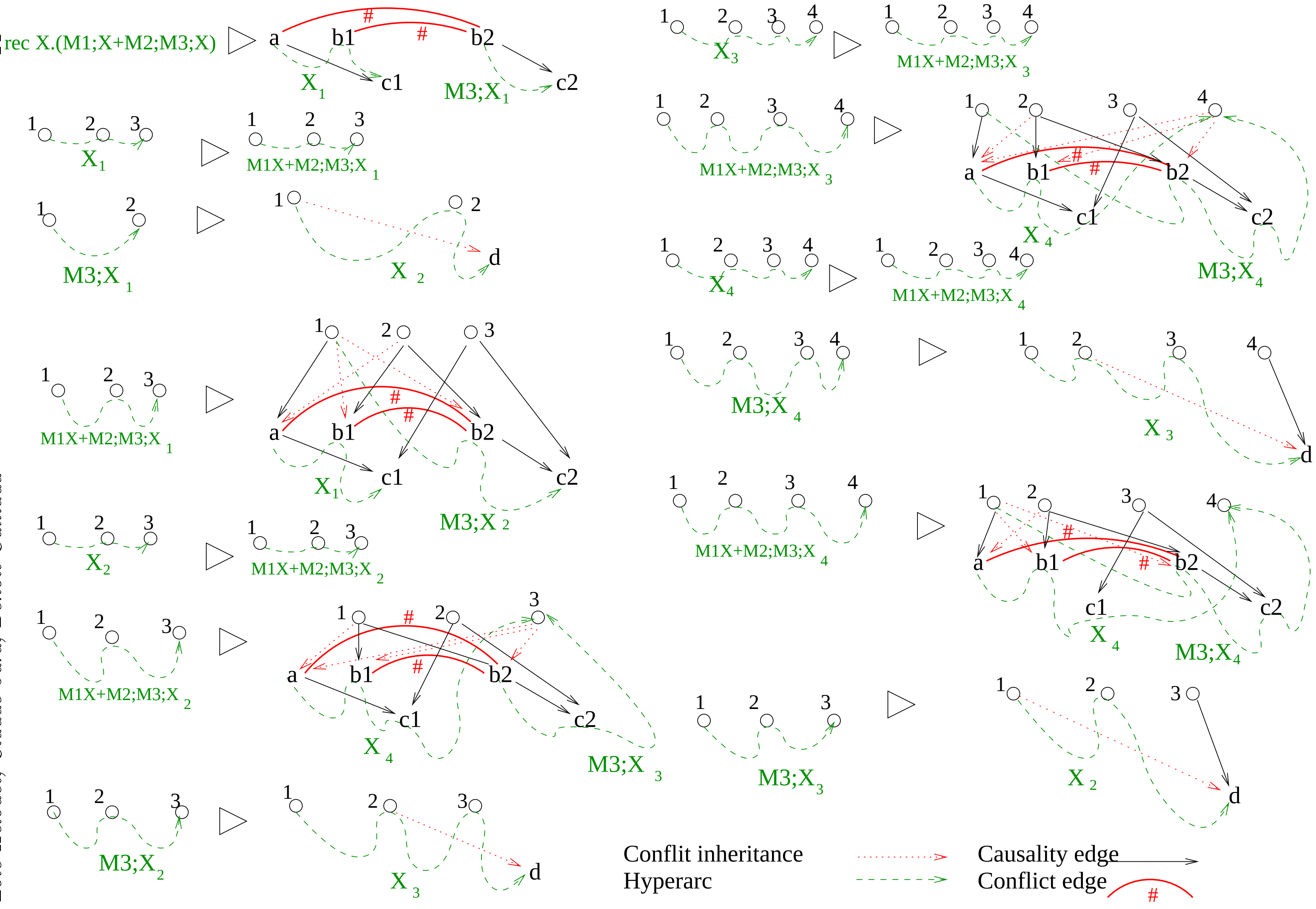


Fig. 13. Graph grammar calculated from HMSC $P2$ in Figure 5

4. HMSCs Equivalence

It is shown in (Caucal92) 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 of 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 isomorphic.

Section 3 has shown how to express infinite ESs described by HMSCs by the means of 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 characterize the cases when infinite branching is generated, and then shows that the isomorphism decision procedure can be brought back to isomorphism of connected graphs of finite degree.

4.1. *Infinite Branching*

The covering graph of a 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 $M_n; P_n$.

An example of such a specification is given in Figure 5 (instance A is active in M_1, but not in $M_2; M_3$). A graph grammar calculated from a HMSC can be modified in order to obtain finite branching covering graphs. For any rule (X, H), 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, H) is said to *loop* if a hyperarc labeled by X can be found after applying a finite length derivation to H $(\exists (X, red) \in App_X)$.

4.2. *Redundancy Elimination*

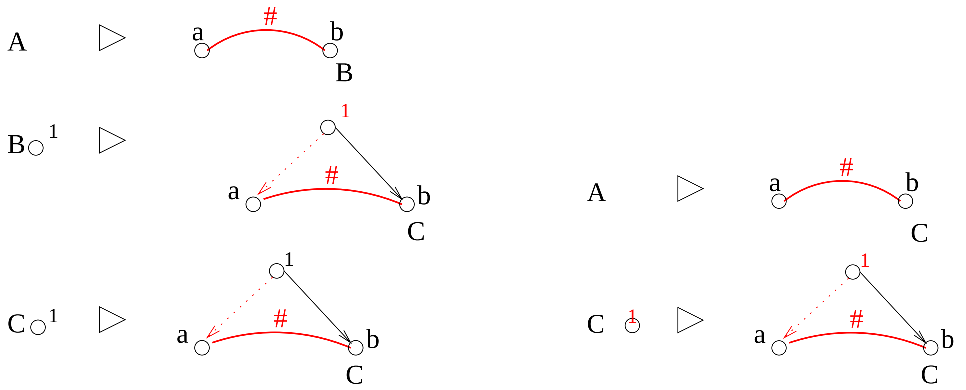
Unfolding of loops shall now be eliminated. (X, H) is said *redundant* with $(X', H') \in \mathcal{G}$ if and only if:

$$\exists m, n \in I\!N, [\mathcal{G}^m(H)] = [\mathcal{G}^n(H')] \wedge X' \in \mathcal{G}^m(H) \wedge X' \text{ loops } \wedge X = X' \text{ (up to renaming of rule)}$$

We suppress (X, H) redundant with (X', H') by removing (X, H) 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.

4.3. *Removing Infinite Branching*

A grammar $\mathcal{FBG}_P$ generating a graph with finite branching can be computed from a grammar $\mathcal{G}_P$ in the following way:



a) Grammar with redundancy b) Equivalent grammar without redundancy

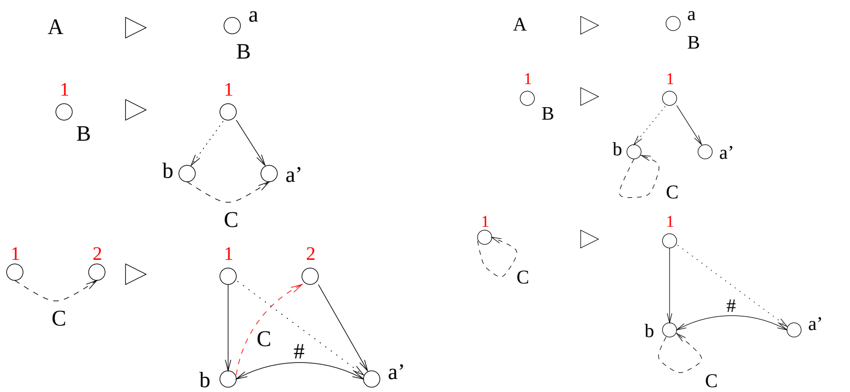
Fig. 14. Redundancy elimination

For any looping rule (X, H) of $\mathcal{G}_P$, if there is a vertex v kept all along the successive rewritings of H ($\exists (X, Red) \in App_X \mid red \neq \emptyset$), then any edge originating from v creates an infinite branching. Therefore, edges going out of v must be deleted from R, and vertex v can be removed from any hyperarc labeled 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, and the grammar of Figure 15-b is obtained.



a) Grammar with infinite branching b) Grammar without infinite branching

Fig. 15. Removing infinite branching

Theorem 4.1.

Removing all ib-vertices for a grammar $\mathcal{G}$ suppresses all infinite branchings in the covering graph $\mathcal{G}^w$.

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 if an instance can be either active or not in different branches of the choice. As long as an instance performs no action, the vertex corresponding to the last action is kept by rewritings, and so $\exists (X, red) \in App_X \mid red \neq \emptyset$. □

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

Proof. From the construction method, inheritance edges do not generate infinite branching (a vertex can be used only once as the origin of inheritance edges). So, edges connected to an ib-vertex are only causalities. □

4.4. *Removing End-vertices from Hyperarcs*

An *end-vertex* in a rule $r = (X, H)$ is a vertex v such that: $\forall n \in I\!N, \nexists (v, v') \in \mathcal{G}^n(X)$. And 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 $r = (L.ctxt.E, G) \in \mathcal{G}$, let E' be a set of end-vertices of E. A rule $r' = (L.ctxt.E', G_{/E-E'})$ 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 $\mathcal{RE}(\mathcal{G})^w = \mathcal{G}^w$.

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, $\{s'_{o_i}\}, i \in I\!N$, located on the same instance ($\forall i \in I\!N, \phi(s_o) = \phi(s'_{o_i})$).

The causality relation between s_o and one of its successors s'_{o_i} can't be due to a message. Effectively, standard Z.120 specifies that a message must be sent and received within the same bMSC (except when constructs such as gates are used, which case is not considered within our approach). Consequently, we can't have a message sent only once 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 inheritance relation, and any event e such that $\phi(d_i) \neq \phi(e)$ which 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 between c_i and d_i, and there is a causality 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 remain 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.

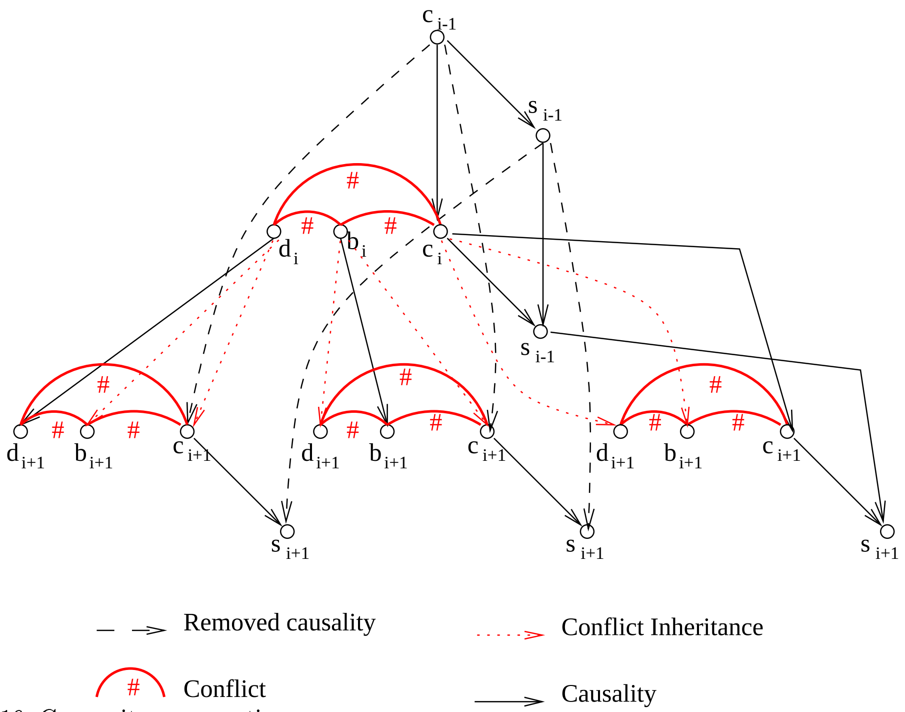


Fig. 16. Connexity preservation

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 for the reasons 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 $\{(X_i, H_i)\}$, without inheritance edges (see for example rule B on Figure 15). The next occurrences of that choice are expressed by a set of looping rules $\{(B_i, S_i)\}$, that might be modified. As the rules $\{(X_i, H_i)\}$ don't loop, they won't 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 a HMSC contains redundant information. Thanks to the inheritance and conflict edges, the structure of the HMSC can still be deducted after removing infinite branching. Therefore, the full covering graph can be reconstructed.

Let $CG =< E, \longrightarrow, \rightsquigarrow, \sharp >$ 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: $\forall (i \neq j) e_i \neq e_j \wedge (e_i, e_{i+1}) \in (\longrightarrow \cup \rightsquigarrow \cup \sharp)$.

More intuitively, a link is a directed path of the covering graph CG without cycle (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 $\{\{\longrightarrow + \rightsquigarrow\}^+ \{\sharp\}^*\}^*$

Theorem 4.3. Let $CG =< E, \longrightarrow, \rightsquigarrow, \sharp >$ 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 $e \overset{*}{\rightsquigarrow} e'$, then e, e' and any event e_i situated between them are connected by a causality or inheritance edge belong to a common scenario. if $e \sharp e'$, the conflict extends to successors but predecessors of e and e' and not in conflict either with e and its successors nor with e' and its successors. Furthermore, if for each e_i, $i \in 1..n-1, \phi(e_i) = \phi(e_0) \Longrightarrow e_i \sharp e_{i+1}$, then e_n is the first event contained in the same scenario than 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 from e between two choices.
$Brsucc(e) = \{e' \mid e = e_1 \longrightarrow e_2 ... \longrightarrow e_n = e' \wedge \forall i \in 2..n, \nexists x, x \sharp e_i\}$

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

Proof. 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', 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, then 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).

— 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 5, we know that $\leq$ can be reconstructed, and therefore $\longrightarrow$ can also be reconstructed.

□

4.7. *Isomorphism Preservation*

Let us show that removing infinite-branching vertices does not affect isomorphism decision.

Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two graph grammars without redundancies calculated from HMSCs. We want to show that $\mathcal{G}_1^w \equiv \mathcal{G}_2^w \iff \mathcal{FBG}_1^w \equiv \mathcal{FBG}_2^w$.

— $\mathcal{G}_1^w \equiv \mathcal{G}_2^w \Longrightarrow \mathcal{FBG}_1^w \equiv \mathcal{FBG}_2^w$?
We know that infinite branching suppression does not affect vertices, conflicts, nor inheritance edges on a covering graph. So, if $\mathcal{FBG}_1^w \not\equiv \mathcal{FBG}_2^w$, then there exists a causality edge within $\mathcal{FBG}_1^w$ that does not belong to $\mathcal{FBG}_2^w$ (or conversely). Such a causality edge is generated by a rule (X_1, H_1) that does not loop in $\mathcal{G}_1$, and by a rule (X_2, H_2) that loops within $\mathcal{G}_2$. As $\mathcal{G}_1^w \equiv \mathcal{G}_2^w$, and as (X_2, H_2) loops, we know that (X_1, H_1) and (X_2, H_2) represent the same part of a recursion in an HMSC. But as (X_1, H_1) does not loop, this rule adds a new active instance within the HMSC, or it is the first occurrence of a choice.

  - if (X_1, H_1) adds a new active instance to the HMSC, then, as (X_2, H_2) loops, no instance is added to the HMSC at this time, so $\mathcal{G}_1^w \equiv \mathcal{G}_2^w$ cannot hold.
  - if (X_1, H_1) is the first occurrence of a choice, then the next occurrences of this choice will have to include inheritance edges. As (X_2, H_2) loops, it can not represent the same pattern than (X_1, H_1), and we can not have $\mathcal{G}_1^w \equiv \mathcal{G}_2^w$.

Which proves $\mathcal{G}_1^w \equiv \mathcal{G}_2^w \Longrightarrow \mathcal{FBG}_1^w \equiv \mathcal{FBG}_2^w$.

— $\mathcal{G}_1^w \equiv \mathcal{G}_2^w \Longleftarrow \mathcal{FBG}_1^w \equiv \mathcal{FBG}_2^w$?
From Theorem 5, we know that any removed causality edge can be reconstructed using causal links. It follows that $\mathcal{FBG}_1^w \equiv \mathcal{FBG}_2^w \Longrightarrow \mathcal{G}_1^w \equiv \mathcal{G}_2^w$.

□

4.8. *A Normal Form for HMSCs*

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

4.9. *Standard Form Grammars*

The normalization 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 a HMSC can be transformed into a standard form grammar by unfolding rules.

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

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

A grammar $\mathcal{G}$ is *separated* if: $\forall (X,H) \in \mathcal{R}, \forall h_1, h_2$ hyperarc of H, h_1 and h_2 have no common vertices.

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

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

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

Proof. Obvious from the construction method, every vertex added is a terminal one. □

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

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,H) of $\mathcal{G}$, there is a minimal number of unfoldings n such that $(X, \mathcal{G}^n(H))$ is a normal rule. □

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

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 kept all along a loop are removed (infinite branching and end vertices are discarded), such a vertex kept can not be a looping vertex, and therefore must have a successor after a finite number of rewritings. □

A computation of a standard form for our grammar is straightforward: it consists in unfolding rules as long as they are not normal and separated. Due to the tree-like structure of our covering graphs, the algorithm is more simple than Caucal's algorithm ((Caucal92)). As the standard form calculus is performed by rewritings, it is obvious that the standard grammar generates the same covering graph than the non-standard one.

4.10. *Normalized Grammars*

A grammar $\mathcal{G} = (Axiom, \mathcal{R})$ is said to be *normalized* if and only if:

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

An uniformisation algorithm is provided in (Caucal92), and proposes to generate uniform rules from connected sets situated at the same distance. Let us define a metrics on the covering graph $\mathcal{G}^w(Axiom)$. 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. Let us note G_n the covering graph restricted to vertices of distance strictly less than n. $G_n = \mathcal{G}^w(Axiom)_{/\{s|dist(s)<n\}}$

For a given set S of vertices, let $Conn(S, G)$ be set of vertices connected to S in G. $Conn(S, G) = \{v \in G | \exists v' \in S \wedge \big((v, v') \in (\longrightarrow \cup \dashrightarrow \cup \sharp) \vee \exists (v', v) \in (\longrightarrow \cup \dashrightarrow \cup \sharp)\big)\}$

From a standard form deterministic and finite branching grammar $\mathcal{G}$, a normalized grammar can be computed. The normalization algorithm consists in generating from $\mathcal{G}$ an uniform grammar $\mathcal{G}' = (root, \mathcal{R}')$ i.e a grammar such that $\mathcal{R}' = \{r_i = (X_i, R_i)\}$, and:

— $\forall i, X_i$ is a set of vertices situated at the same distance n, and connected in $\mathcal{G}^w(Axiom) - G_n$
— $\forall i, R_i$ is the restriction of $\mathcal{G}^w(Axiom) - G_n$ to $X_i \cup Conn(X_i, \mathcal{G}^w(Axiom) - G_n)$
— $\mathcal{G}'^w(root) = \mathcal{G}^w(axiom)$

The main idea of the algorithm is to find the set $\{X_i\}$ on a limited unfolding of the grammar, as $\{R_i\}$ is calculated from $\{X_i\}$. The complete uniformisation algorithm and a termination proof can be found in (Caucal92).

4.11. *A Decision Procedure for HMSC Equivalence*

Section 3 has shown that it is possible to generate a graph grammar defining a unique event structure from a HMSC. This graph grammar may comprise infinite branchings. Section 4 has shown that a modification of this grammar allows to eliminate infinite branching while preserving isomorphism. Section 4.8 has 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 $\mathcal{FBG}_1$ and $\mathcal{FBG}_2$ by eliminating infinite branchings and end- vertices within $\mathcal{G}''_{P_1}$ and $\mathcal{G}''_{P_2}$.
— compute $\mathcal{FNG}_{P_1}$ and $\mathcal{FNG}_{P_2}$, the normal forms of $\mathcal{FBG}_{P_1}$ and $\mathcal{FBG}_{P_2}$, by unfolding rules.
— compute $\mathcal{NG}_{P_1}$ and $\mathcal{NG}_{P_2}$, the normalized grammars for $\mathcal{FNG}_{P_1}$ and $\mathcal{FNG}_{P_2}$,

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

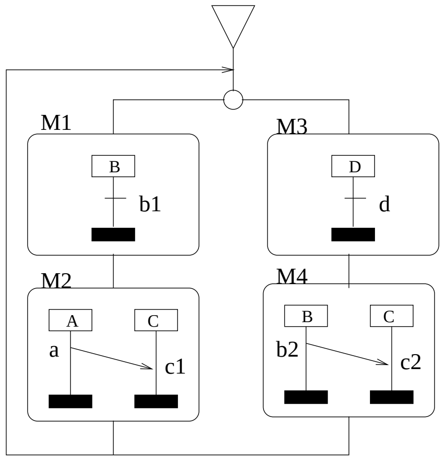


Fig. 17. HMSC P_3

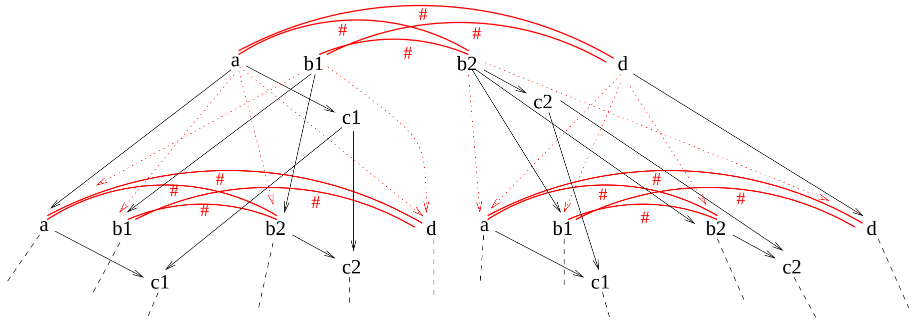


Fig. 18. Covering graph generated by the normal form

5. Conclusion

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

This formal representation of HMSCs exhibits several advantages. First of all, it preserves the partial order semantics described in specifications (event independence, as well as weak sequential composition of bMSCs), which appears as the main advantage of the notation. On the contrary, an interleaving semantics would fail preserving event independence. Then, the semantics defined in this paper emphasizes on conflicts. Choices in a HMSC define important control points in the specification, and should be implemented as a consensus or a synchronization when a distributed choice appears. Finally, using a graph semantics for HMSCs allows to keep a visual and intuitive representation. Furthermore, it makes available a set of well known algorithms on graphs for HMSCs, among which graph isomorphism decision for regular graphs.

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 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 this operation, information about the situation of choices is not considered, and the e-quivalence corresponds to the isomorphism of partial order families. The identification of classes of MSCs that can be determinised, and of determinisation algorithm based on rule unfolding and conflict merging is currently under study.

The representation of HMSC by means of graph grammars and event structures made possible a formal manipulation from informal specifications of distributed systems. It can be considered as another argument towards the use of formal methods even on incomplete informal models at early stages of specification.

References

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

Caucal.D. (1992) On the regular structure of prefix rewriting. *Theoretical Computer Science*, Vol. 106, No 1, pp. 61-86.

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

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

Harel.D, 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, Vol. 23, No. 5, pp. 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*, pp. 281-290.

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

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

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

Leue.S, Ladkin.P (1995) Interpreting message flow graphs. *Formal Aspects of Computing*, Vol 7, No. 5, pp. 473-509.

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

Mauw.S. (1996) The formalization of message sequence charts, Computer Networks and ISDN Systems, Vol 28, No. 12, pp. 1643-1657.

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

Object Geode, http://www.cse.fau.edu/resources/sw/ObjectGeode/Ana_uc.htm

OMG (1997) Unified Modeling Language 1.1.

Pratt.V (1986), Modeling Concurrency with Partial Orders, International journal of Parallel Programming, Vol. 15, No. 1, pp. 33-71.

Reniers.M, Mauw.S. (1994) An algebraic semantics for basic message sequence charts, The Computer Journal", Vol. 37, No. 4, pp. 269-277.

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

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

Rensink.M, Wehrheim.H (1994), Weak Sequential Composition in Process Algebras CONCUR '94: Concurrency Theory, 5th International Conference, Lecture Notes in Computer Science, Vol. 836, Jonsson.B and Parrow.J editors, Springer-Verlag publisher, pp. 226-241.

Rudolph.E, Graubmann.P, Grabowski.J (1996), Tutorial on Message Sequence Charts (msc'96), Computer Networks and ISDN Systems, Vol. 28, No. 12, pp. 1629-1641.

SDT, http://www.telelogic.se/solution/language/msc.asp

Winskel.G, Nielsen.M, Plotkin.G (1981) Petri nets, event structures and domains, part 1. *Theoretical Computer Science*, Vol. 13, No 1, pp. 85-108.