An Effective Equivalence for Sets of Scenarios Represented by HMSCs

Loïc Hélouët, Claude Jard, Benoit Caillaud

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

An effective equivalence for sets of scenarios represented by HMSCs

Loïc Hélouët, Claude Jard, Benoît Caillaud

N ° 3499

Septembre 1998

THÈME 1

ISSN 0249-6399

An effective equivalence for sets of scenarios represented by HMSCs

Loïc Hélouët, Claude Jard, Benoît Caillaud

Thème 1 — Réseaux et systèmes
Projet PAMPA

Rapport de recherche n3499 — Septembre 1998 — 29 pages

Abstract: 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 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.

Key-words: MSC, equivalence, event structure, pomset, graph grammar.

(Résumé : tsvp)

This work was partially supported by Alcatel within the REUTEL action

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Une notion effective d'équivalence pour des familles de scénarios représentés par des HMSC

Résumé : Nous définissons une sémantique d'ordre partiel pour des familles de scénarios représentées par des High-Level Message Sequence Charts (HMSC). Celle-ci permet d'associer à chaque HMSC une structure d'événements engendrable par une grammaire de graphes finie. On peut alors décider effectivement l'équivalence de deux HMSC, ce qui ouvre la voie à la manipulation formelle de scénarios.

Mots-clé : MSC, équivalence, structure d'événements, ordre partiel, grammaire de graphes.

Contents

1 Introduction

System modeling by means of scenari is a common practice. Several formal notations such as "Message Sequence Charts" (MSC) have been proposed. MSCs are mostly used for describing the activities of communicating entities in distributed systems. They can be found in a closely related formalism, called Sequence Diagrams, one of the UML (Unified Modeling Language) views [21]. A MSC graphically describes communications between processes (called *Instances*). MSCs give a very intuitive and visual representation of systems behaviours.

The major drawback of this kind of notation is the number of scenari you have to combine to get a complete system specification. A simple idea is to build families of scenari, using a kind of "scenario automaton". The most elaborate proposal for MSCs is being normalized, and is called "High level MSC" or HMSC [19]. HMSCs define MSC compositions 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, we will only deal in this paper with a subset of HMSCs, limited to choice and sequencing operators.

We aim to provide such a powerful notation with a formal background, allowing :

- property checking : For example, is there a possible process divergence (possibly leading to unbounded message accumulation in a communication medium) in a specification? If we want to make a distributed implementation, are there any non local choices that will require distributed consensus? A first step towards property checking was proposed in [16]. The algorithms defined in this article allow to decide whether a HMSC describing a property "match" another HMSC describing a system or not.
- Automatic program skeleton generation, which will ensure that any execution will respect the order defined in the scenari.

This paper makes a first contribution within this context: the decision of HMSC equivalence.

The question of MSC semantics has been treated in many papers ([3, 13, 14, 10, 11, 12, 9]), but the semantics for HMSCs is still in its infancy. A first interleaving semantics for HMSCs was proposed in [15]. More recently, [8, 6] defined partial-order semantics. Another vision of HMSCs was proposed in [5], where some scenari are "mandatory", and some are "optional".

We consider that a MSC defines a partial order between events, and that a HMSC describes a (possibly infinite) family of partially ordered sets. This family is built from basic MSCs (the HMSC nodes), using composition operators : sequencing (order concatenation) and choice (set theoretic union).

A family of partial orders can be represented by a prime event structure [17]. Within an event structure, orders are represented together, and separated by a conflict relation. This property suggests us that prime event structures are a good model for a canonical representation of HMSCs.

We define HMSC equivalence as the isomorphism of their event structure representations. Therefore, the point is to give a decision procedure for this isomorphism. The potentially

infinite size of prime event structures leads us to consider finite objects representing them. We choose graph grammars, as event structures can be represented by their covering graphs.

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 [2]. 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.

Our partial order representation of HMSCs allows for an equivalence decision. This is a starting point of formal reasoning about MSCs.

The paper is organized as follows: Section 2 introduces the MSC and HMSC formalism. Section 3 defines a partial order semantics for MSC and HMSC. Section 4 recalls basic notions about event structures. Section 5 introduces graph grammars, and Section 6 gives a set of rules for the generation of grammars from a HMSCs. Section 7 is the correctness proof for the rules. Section 8 defines the equivalence between two HMSCs. Section 9 gives an equivalence decision algorithm from graph grammars, based on a normal form computation. Section 10 shows a simple property detection from our representation.

2 bMSCs and HMSCs

Message Sequence Charts (MSCs for short) graphically define distributed systems and communications between the components of this system. 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 norm Z.120 [7]).

2.1 bMSCs

A bMSC (standing for basic MSC) defines a simple scenario, ie 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. 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. Figure 1 summarizes the different kind of events that can be found within a bMSC.

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 *co-regions*. These co-regions are represented by dashed parts of the instance axis. Events situated in a co-region 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 $\mathcal{M}$.

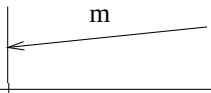
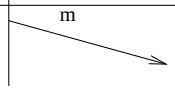
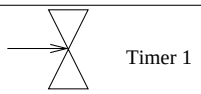
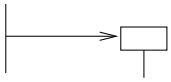
Message events	Receiving	
	Sending	
Timer Events	Set	
	Reset	
	Timeout	
Instance events	Creation	
	Stop	
	Internal action	

Figure 1: Features of bMSCs

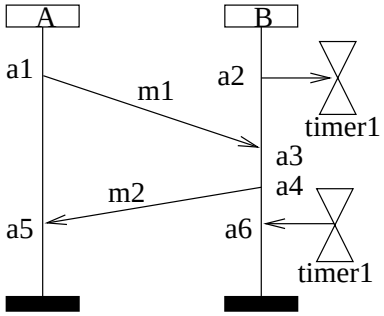


Figure 2: An example bMSC.

2.2 HMSCs

A bMSC defines only one scenario. Extending bMSCs requires a higher-level notation, and allowing for the specification of a sets of scenari, made of combinations of bMSCs. HMSC stands for High-level MSC. A HMSC can be seen as a 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 scenari. HMSC P_0 in Figure 3 defines two possible scenari, represented in Figure 4.

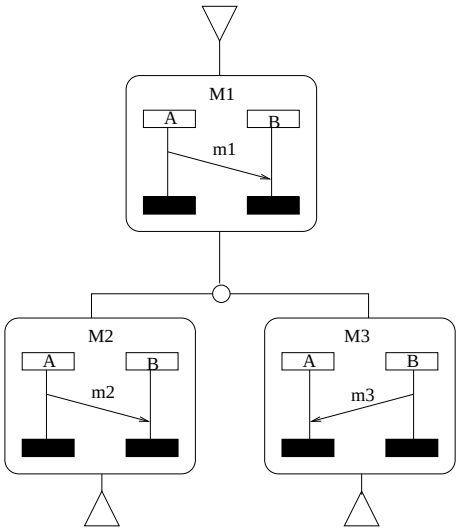


Figure 3: HMSC P_0 (with non-local choice)

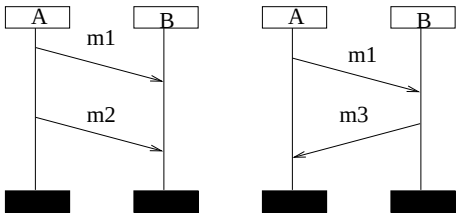


Figure 4: Scenari defined by the HMSC in Figure 3

A HMSC can also describe an infinite behaviour, as in Figure 5.

The representation of system behaviours with MSCs allows for the introduction of undesirable features. For example, HMSC P_0 presented in Figure 3 includes an exclusive

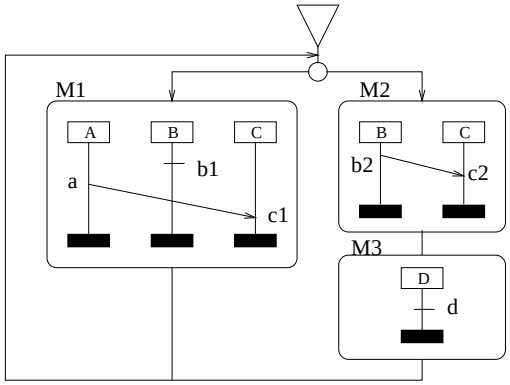


Figure 5: HMSC P_1

distributed choice between two scenari M_2 and M_3. Consequently, the two possible scenari described by P_0 are $M_1; M_2$ and $M1; M3$. To be consistant with the specification, an implementation will have to ensure that only these two scenari can be performed. As the decision to perform M_2 or M_3 is distributed, this can't be done without a synchronization between A and B. This situation will be called a *non-local choice*. Section 10 describes another problem, called *process divergence*. Such bad properties of a system description reinforces the need for formal manipulation of MSCs.

2.3 Regular expressions on MSCs

We now define a convenient notation for manipulating HMSCs. HMSCs define set of scenari that can be defined by 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 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 $P_1 ::= recX.(M1; X + M2; M3; X)$.

3 Partial order semantics for MSCs

3.1 bMSCs and Partial Orders

Let us consider a bMSC M describing the behaviour 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 event e is performed by instance $i \in I$ $(\exists a \in A \mid \alpha(e) = (a, i))$.

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.

3.2 HMSCS and partial order families

Clearly, HMSCs define sets of scenari, that will be called *partial order families* (POF for short) hereafter. We can define inductively a POF of index k, noted $\mathcal{F}_k$, associated to any HMSC $P \in (\mathcal{M}, ;, \Sigma, rec, \epsilon)$:

- $\forall P, \mathcal{F}_0(P) = \{\emptyset\}$
- $\mathcal{F}_k(\epsilon) = \{\emptyset\}$
- $\mathcal{F}_k(M; P) = \{M \circ f \mid f \in \mathcal{F}_{k-1}(P)\}$
- $\mathcal{F}_k(\sum_{i \in 1..K} M_i; P_i) = \bigcup_{i \in 1..K} (\{M_i \circ f_i \mid f_i \in \mathcal{F}_{k-1}(P_i)\})$
- $\mathcal{F}_k(recX(P)) = \mathcal{F}_{k-1}(P_{[X:=recX.(P)]})$

Let $\mathcal{W}$ denote the set of well founded partial order families. Let $\mathcal{F} \in \mathcal{W}$ be a partial order family and f be an order of $\mathcal{F}$. The *rank* of an event e of f is the upper bound of the length of every strictly decreasing sequence starting from e.

We note $\mathcal{R}_l(f)$, for $l \geq 0$ the subset of f made of events of rank less or equal than l.

Let f and g be to well-founded posets. We define the distance between f and g as :

$$d(f, g) = 2^{-min\{l \mid R_l(f) \not\equiv R_l(g)\}}$$

Let $\mathcal{F} \in \mathcal{W}$ and $\mathcal{J} \in \mathcal{W}$ be two POF. The distance between $\mathcal{F}$ and $\mathcal{J}$ is debined by :

$$d(\mathcal{F}, \mathcal{J}) = \min_{\rho \subseteq \mathcal{F} \times \mathcal{J}} (max(d(f, g) \mid (f, g) \in \rho))$$

As $d(\mathcal{F},\mathcal{J}) + d(\mathcal{J},\mathcal{H}) \geq d(\mathcal{F},\mathcal{H})$, $d(\mathcal{F},\mathcal{J}) = d(\mathcal{J},\mathcal{F})$, and $d(\mathcal{F},\mathcal{J}) = 0 \iff \mathcal{F} = \mathcal{J}$, d is a distance and $(\mathcal{W}, d)$ is a complete metric space.

$\forall P \in (\mathcal{M}, ;, , \Sigma, rec, \epsilon), (\mathcal{F}_k(P))_{k\geq 0}$ is a Cauchy sequence :

$(\forall \epsilon > 0, \exists N \geq 0 \mid \forall n, m \geq N, d(\mathcal{F}_n(P), \mathcal{F}_m(P)) < \epsilon)$.

Therefore, $(\mathcal{F}_k(P))_{k\geq 0}$ converges, and its limit is noted $\mathcal{F}_w(P)$. The partial order family associated to a HMSC P is $\mathcal{F}_w(P)$.

4 Event Structures

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 3.1, and $\sharp$ is a symmetric anti-reflexive binary relation called *conflict relation*, such that :

> $\forall e \in E, \forall e' \in E, e\sharp e' \Leftrightarrow \forall e'' \in E, (e' \leq e'' \Rightarrow e\sharp e'')$ (the 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 ($\forall e \in C, e' \leq e \Rightarrow e' \in C$). For further reading on event structures, consult [17].

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 $\underset{i\in 1..n}{\sharp}(S_i)$.

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 behaviours 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 causality order. 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.

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 dependant on e.

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

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

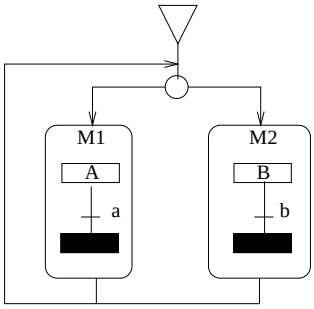


Figure 6: HMSC with an irregular SE representation

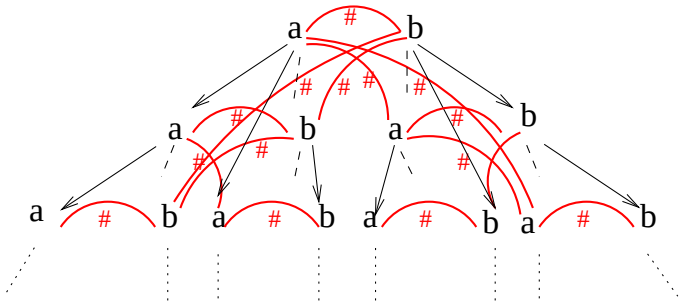


Figure 7: Irregular SE graph for HMSC of Figure 6

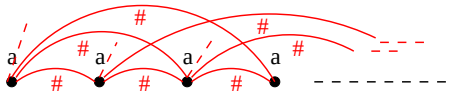


Figure 8: An irregular subgraph extracted from the graph in Figure 7

- $\rightsquigarrow$ 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. $\{(e, e') \mid \nexists e'' \wedge (e'' \leq e' \vee e'' \rightsquigarrow e') \wedge e \sharp e''\}$.

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

A covering graph of the event structure of Figure 7 is represented in Figure 9. $\longrightarrow$ is represented by simple arrows, $\rightsquigarrow$ is represented by dotted arrows, and $\sharp$ is represented by an edge labelled by $\sharp$.

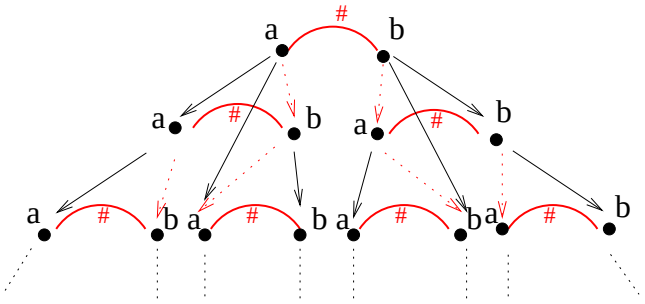


Figure 9: Regular covering graph for HMSC Figure 6

5 Graph grammars

Only a short introduction to graph grammars is given. For further reading, [4] may be consulted.

5.1 Hypergraphs

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 $\{s_i\}$ are vertices of $T(G)$. On the hypergraph represented in Figure 10, the vertices s_4, s_5, s_6 are linked by an hyperarc labeled by A. Such an hyperarc will then be noted $As_5s_6s_4$.

5.2 Graph grammars

A *graph grammar* consists in an hypergraph G_0 called the *axiom* of the grammar, and of a set $\mathcal{R}$ of rewriting rules. A rewriting rule is a pair (X, H), where X is an hyperarc, and H is the hypergraph rewritten from X's vertices. We will often write this rule $X \triangleright H$. For an 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.

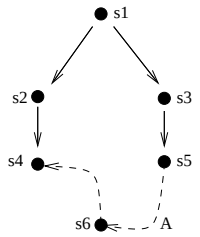


Figure 10: Hypergraph

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

Parallel derivation generalizes direct derivation, by enabling simultaneous hyperarcs 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 $\{1..n\}$, $M \underset{\mathcal{G},(X_{\pi(1)},R_{\pi(1)})}{\longrightarrow} ... \underset{\mathcal{G},(X_{\pi(n)},R_{\pi(n)})}{\longrightarrow} N$.

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

6 Graph grammar generation from an HMSC

6.1 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 :

- $S_i = \{s \in S \mid \phi(s) = i\}$ $(i \in I)$.
- $Inf(\mathcal{G}r(M)) = \{s \in S \mid \forall s' \in S, (s', s) \notin \longrightarrow\}$: minimal events of $\mathcal{G}r(M)$ w.r.t $\longrightarrow$.
- $Sup(\mathcal{G}r(M)) = \{s \in S \mid \forall s' \in S, (s, s') \notin \longrightarrow\}$: maximal events of $\mathcal{G}r(M)$ w.r.t $\longrightarrow$.
- $Inf_i(\mathcal{G}r(M)) = \{s \in S_i \mid \forall s' \in S_i, (s', s) \notin \longrightarrow\}$ $(i \in I)$: minimal event on instance i (if this event exists).
- $Sup_i(\mathcal{G}r(M)) = \{s \in S_i \mid \forall s' \in S_i, (s, s') \notin \longrightarrow\}$ $(i \in I)$: maximal event on instance i (if this event exists).

- $Act(\mathcal{G}r(M)) = \{i \in I \mid \exists s \in S \wedge \phi(s) = i\}$: set of instances that perform at least an event within M.

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

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

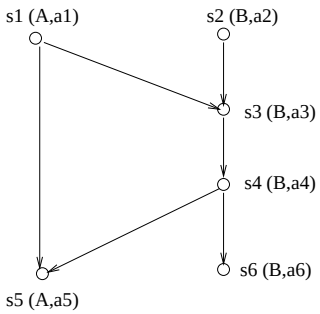


Figure 11: Covering graph $\mathcal{G}r(M_1)$

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

6.2 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. We associate a graph grammar 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*, i.e. the set of active instances, the branch of the choice we are evaluating, and so on. Therefore, the rule generated from an expression won't 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.

We define the function $Rules(P, S, LE, Her, Br, Var) = \{(H, R)\}$, that gives 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, an hyperarc labelled by P rewrites into an hypergraph C including hyperarcs of the form $P'_i.S'_i$. For the sake of clarity, we neither define the computation of A, I, or α nor we indicate the context on the labels of the rules. The rules associated to an 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 labelled by ϵ.

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

  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.

- $Rules(\sum\limits_{k \in 1..K} M_k; P_k, S, LE, Her, Br, Var) =$
  $$\sum_{k \in 1..K} M_k; P_k.S \triangleright \left(\langle S_\Sigma, \longrightarrow_\Sigma, \rightsquigarrow_\Sigma, \sharp_\Sigma, \alpha, A, I\rangle,\ \bigcup_{k \in 1..K} P_k.S_k\right)$$
  $\cup \bigcup\limits_{k \in 1..K} Rules\left(P_k, S'_k, LE'_k, Her'_k, Act(\mathcal{G}r(M_k)), Var\right)$ where :
  - $S_\Sigma = S \cup \bigcup\limits_{k \in 1..K} S_{\mathcal{G}r(M_k)}$
  - $\longrightarrow_\Sigma = \bigcup\limits_{k \in 1..K} \Big(\{(s, Inf_i(\mathcal{G}r(M_k))) \mid s \in LE(S) \wedge \phi(s) = i \wedge i \in Act(\mathcal{G}r(M_k))\} \cup \underset{\mathcal{G}r(M_{k'})}{\longrightarrow} \Big)$

- $\rightsquigarrow_{\sum} = \{(s, Inf_i(\mathcal{G}r(M_{k'})) \mid s \in S \wedge Her(s) \wedge i \in Act(\mathcal{G}r(M_k)\}$
- $\sharp_{\sum} = \{(s, s') \mid s \in Inf(\mathcal{G}r(M_m)) \wedge s' \in Inf(\mathcal{G}r(M_n)) \wedge m \neq n\}$
- $S'_k = \{s \in LE(S) \mid \phi(s) \notin Act(\mathcal{G}r(M_k))\}$
  $\cup\{Sup_i(\mathcal{G}r(M_k)) \mid i \in Act(\mathcal{G}r(M_k))\} \cup \{Inf(\mathcal{G}r(M_k))\}$
- $Her'_k = Inf(\mathcal{G}r(M_k))$
- $LE'_k = LE \cap S'_k \cup \{Sup_i(\mathcal{G}r(M_k)) \mid i \in I\}$

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

- $Rules(recX.(P), S, LE, Her, BR, Var) = recX(P).S \triangleright \left(\langle S, \emptyset, \emptyset, \alpha, A, I\rangle,\ P.S\right)$
  $\cup\ Rules(P, S, LE, Her, Br, Var \cup \{(X, P)\})$
- $Rules(X, S, LE, Her, Br, Var) = X.S \triangleright \left(\langle S, \emptyset, \emptyset, \alpha, A, I\rangle,\ P.S \mid (X, P) \in Var\right)$
  $\cup\ Rules(P, S, LE, Her, Br, Var)$

Let P be a HMSC. The graph grammar generating the covering graph for P is : $\mathcal{G}_P = \left(\ P, Rules(P, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)\ \right)$.

The graph grammar generated for HMSC in Figure 5 is represented in Figure 12. 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.

7 Grammar correctness

The question addressed in this section is whether the set of scenari defined by an HMSC and by its graph grammars are the same or not. Said in another way, given a HMSC P, can we show that the graph generated from the graph grammar calculated from P and the event structure semantics of P define the same set of scenari?

7.1 Properties of the rules

The graph grammar computed from a regular expression on MSCs produces a covering graph $\mathcal{C} =< E_{\mathcal{C}}, \longrightarrow_{\mathcal{C}}, \rightsquigarrow_{\mathcal{C}}, \sharp_{c_{\mathcal{C}}}, \alpha_{\mathcal{C}}, A_{\mathcal{C}}, I_{\mathcal{C}} >$

For a single bMSC, we obtain the graph $\mathcal{G}r(M) =< E_M, \longrightarrow_M, \rightsquigarrow_M, \sharp_{c_M}, \alpha_M, A_M, I_M >$, with $\sharp_{c_M} = \emptyset$ and $\rightsquigarrow_M = \emptyset$. $\dashrightarrow$ denotes conflict inheritance or causal dependency : $e_1 \dashrightarrow e_2 = (e_1 \longrightarrow e_2 \vee e_1 \rightsquigarrow e_2)$.

Let $\mathcal{C}$ be a covering graph. The event structure associated to $\mathcal{C}$ is : $ES(\mathcal{C}) =< E_{\mathcal{C}}, \leq, \sharp, \alpha_{\mathcal{C}}, A_{\mathcal{C}}, I_{\mathcal{C}} >$,where

$\leq = \{(e_1, e_2) \in E^2_{\mathcal{C}} \mid e_1 \stackrel{*}{\longrightarrow} e_2\}$

$\sharp = \sharp_c \cup \{(e_1, e_2) \in E^2_{\mathcal{C}} \mid \exists e' \in E_{\mathcal{C}}, \exists e' \in E_{\mathcal{C}} \wedge e \stackrel{*}{\dashrightarrow} e_1 \wedge (e' \stackrel{*}{\dashrightarrow} e_2 \wedge e' \sharp_c e\}$

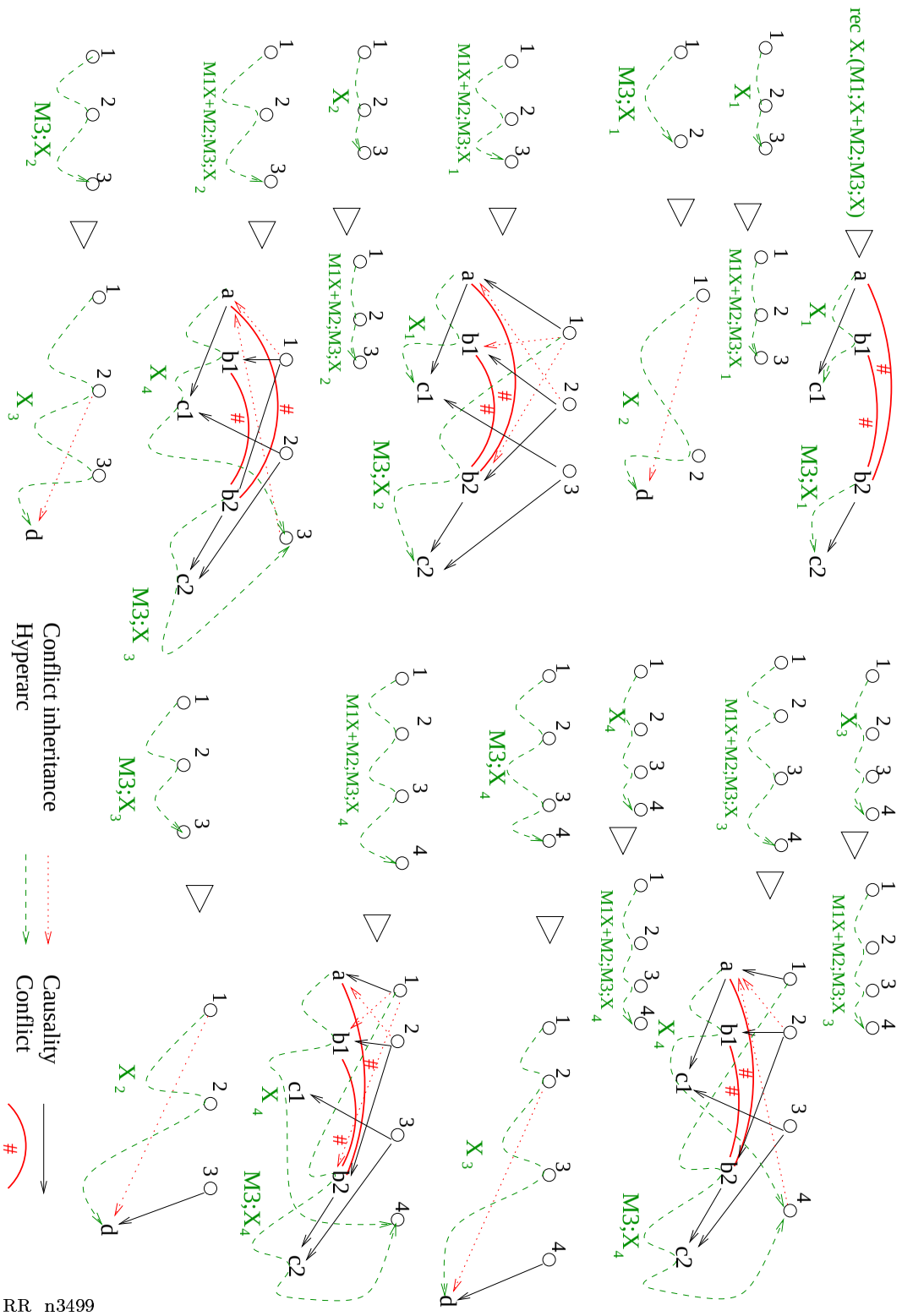


Figure 12: Graph grammar calculated from HMSC P1 in Figure 5

Let $M \in \mathcal{M}$ be a MSC, and $\mathcal{G}r(M)$ be its covering graph. Concatenation between $\mathcal{G}r(M)$ and any covering graph $\mathcal{C}$ is defined by :

$$\begin{aligned}\mathcal{G}r(M) \circ \mathcal{C} = \quad & < E_M \cup E_{\mathcal{C}}, \\ & \longrightarrow_M \cup \longrightarrow_{\mathcal{C}} \cup \{(e,e') \mid \exists i \in I_M \cap I_{\mathcal{C}} \wedge e \in max_i(\mathcal{G}r(M)) \wedge e' \in min_i(\mathcal{C})\}, \\ & \rightsquigarrow_{\mathcal{C}}, \sharp_{c_{\mathcal{C}}}, \alpha_M \cup \alpha_{\mathcal{C}}, A_M \cup A_{\mathcal{C}}, I_M \cup I_{\mathcal{C}} >\end{aligned}$$

Let C_1 and C_2 be two covering graphs. We define the conflict operation between two covering graphs :

$$\begin{aligned}C_1 \sharp C_2 \quad = \quad & < E_{C_1} \cup E_{C_2}, \longrightarrow_{C_1} \cup \longrightarrow_{C_2}, \rightsquigarrow_{C_1} \cup \rightsquigarrow_{C_2}, \\ & min(C_1) \times min(C_2) \cup min(C_2) \times min(C_1) \cup \sharp_{C_C 1} \cup \sharp_{C_C 2}, \\ & \alpha_{C_1} \cup \alpha_{C_2}, A_1 \cup A_2, I_1 \cup I_2 >\end{aligned}$$

We note that for any covering graph $\mathcal{C}$, $M \circ SE(\mathcal{C}) = SE(Gr(M) \circ \mathcal{C})$.

We can give some properties of the graph grammars obtained from an expression P. We will note $\mathcal{G}_P$ the graph grammar calculated from P. $\mathcal{G}_P^k(P')$ is the graph obtained by k parallel derivations of P' by the grammar $\mathcal{G}_P$.

- $\forall P, \forall P', \mathcal{G}_P^0(P') =< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$
- $\forall k, \forall P, \mathcal{G}_P^k(\epsilon) =< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$
- $\forall M \in \mathcal{M}, \forall P \mathcal{G}_{M;P}^{k+1}(M;P) = \mathcal{G}r(M) \circ \mathcal{G}_P^k(P)$

  According to the rules of Section 6.2,

  $$\begin{aligned}Rules(M;P,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset) = \quad & \Big(\mathcal{G}r(M), P.max(\mathcal{G}r(M)\Big) \\ & \cup Rules\Big(P, \{max_i(\mathcal{G}r(M)) \mid i \in Act(\mathcal{G}r(M))\}, \\ & \qquad \{max_i(\mathcal{G}r(M)) \mid i \in Act(\mathcal{G}r(M))\}, \emptyset, Act(\mathcal{G}r(M), \emptyset\Big).\end{aligned}$$

  So, any first event performed by an instance i in P will be attached to the last event performed by i in M. This is exactly what is done by the concatenation. So, we have : $\mathcal{G}_{M;P}^{k+1}(M;P) = \mathcal{G}r(M) \circ \mathcal{G}_P^k(P)$.

- $\forall M_i \in \mathcal{M}, \forall P_i, \mathcal{G}_{\Sigma M_i;P_i}^{k+1}(\Sigma M_i; P_i) = \sharp\big(\mathcal{G}r(M_i) \circ \mathcal{G}_{P_i}^k(P_i)\big)$
- $\mathcal{G}_{recX.(P)}^{k+1}(recX.(P)) = \mathcal{G}_{recX.(P_{[X:=P]})}^{k+1}(recX.(P)) = \mathcal{G}_{P_{[X:=recX.(P)]}}^{k}(P_{[X:=recX.(P)]})$

7.2 Event structures and partial order families

An event structure defines a set of possible orders between events. Of course, two conflicting events can not appear in the same order. This leads to an intuitive definition of a POF calculus from an event structure.

A *maximal configuration* of an ES $S =< E, \leq, \sharp, \alpha, A, I >$ is a subset C of E such that :

$\forall e \in E - C, \exists e' \in C \mid e' \sharp e$, ie $C \cup \{e\}$ is not a configuration.

The family of maximal orders described by S is noted by $Ord(S)$ and is the set of projections of S on its maximal configurations.

$Ord(S) = \{< \pi_c(E), \pi_c(\leq), \pi_c(\alpha), \pi_c(A), \pi_c(I) >|$ c is a maximal configuration of S $\}$.

The POF for SE of Figure 7 is represented in Figure 13. It is made of a single infinite partial order.

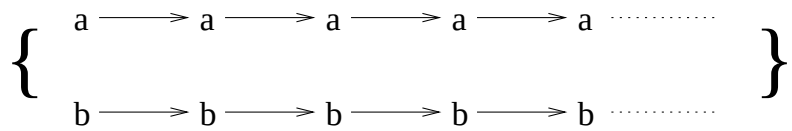


Figure 13: Partial order family

Notice that $Ord(\underset{i\in 1..K}{\sharp}(S_i)) = \underset{i\in 1..K}{\cup} Ord(S_i)$ (two events from conflicting ESs are in conflict, so $\sharp$ doesn't create new maximal configurations).

7.3 Rule correctness

Let us show that the event structure generated by the finite grammar encodes the semantics of the HMSC, i.e. that $\forall P \in (\mathcal{M}, ;, \Sigma, rec, \epsilon)$, $\mathcal{F}_w(P) \equiv Ord(SE(\mathcal{G}^w_E(E)))$. This is shown by induction on the expression's structure and on the length of derivations. For any HMSC $P \in (\mathcal{M}, ;, \Sigma, rec, \epsilon)$, $\forall k$, $\mathcal{F}_k(P) = Ord(SE(\mathcal{G}^k_P(P)))$.

- For k=0, $\forall P, \mathcal{F}_0(P) = Ord(SE(\mathcal{G}^0_P(P))) = \{< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >\}$
- Let us assume the property is true for any n $\leq$ k, and let us show it implies it is also true for k+1.
  - if $P = \epsilon$, $\mathcal{F}_{k+1}(\epsilon) = Ord\big(SE(\mathcal{G}^{k+1}_P(\epsilon)\big) = \{< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >\}$
  - if $P = M; P'$, then

$$\begin{aligned} \mathcal{F}_{k+1}(M;P') &= \{M \circ f \mid f \in \mathcal{F}_k(P')\} \\ &= \{M \circ f \mid f \in Ord(SE(\mathcal{G}^k_{P'}(P')))\} \\ &= Ord\big(SE(\mathcal{G}r(M) \circ \mathcal{G}^k_{P'}(P'))\big) \\ &= Ord\big(SE(\mathcal{G}^{k+1}_{M;P'}(M;P'))\big) \end{aligned}$$

  - if $P = \underset{i\in 1..K}{\Sigma} M_i; P_i$

$$\begin{aligned}
\mathcal{F}_{k+1}(\underset{i\in 1..K}{\Sigma} M_i;P_i) &= \bigcup_{i\in 1..K} \mathcal{F}_{k+1}(M_i;P_i)\\
&= \bigcup_{i\in 1..K} Ord\big(SE(\mathcal{G}^{k+1}_{M_i;P_i}(M_i;P_i))\big)\\
&= Ord\big(\underset{i\in 1..K}{\sharp}(SE(\mathcal{G}^{k+1}_{M_i;P_i}(M_i;P_i)))\big)\\
&= Ord\big(SE(\mathcal{G}^{k+1}_{\Sigma M_i;P_i}(\underset{i\in i..K}{\Sigma} M_i;P_i))\big)
\end{aligned}$$

– if $P = recX.(P')$

$$\begin{aligned}
\mathcal{F}_{k+1}(recX.(P')) &= \mathcal{F}_k(P'_{[X:=recX.(P')]})\\
&= Ord\big(SE(\mathcal{G}^k_{P'_{[X:=recX(P')]}}(P'_{[X:=recX.(P')]}))\big)\\
&= Ord\big(SE(\mathcal{G}^{k+1}(recX.(P')))\big)
\end{aligned}$$

Hence, $\mathcal{F}_w(P) = Ord(SE(\mathcal{G}^w_P(P)))$ holds true.

8 HMSCs equivalence

It is shown in [2] 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.

We have 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. We first characterize the cases when infinite branching is generated. Then we show that the isomorphism decision procedure can be brought back to isomorphism of connected graphs of finite degree.

8.1 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 $C = \underset{i=1..K}{\Sigma} M_i;P_i$,
- 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 2.
Our grammars are modified in order to obtain finite branching covering graphs.

For any rule (X, R), we 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, R) is said to *loop* if an hyperarc labeled by X can be found after applying a finite length derivation to R ($\exists (X, red) \in App_X$).

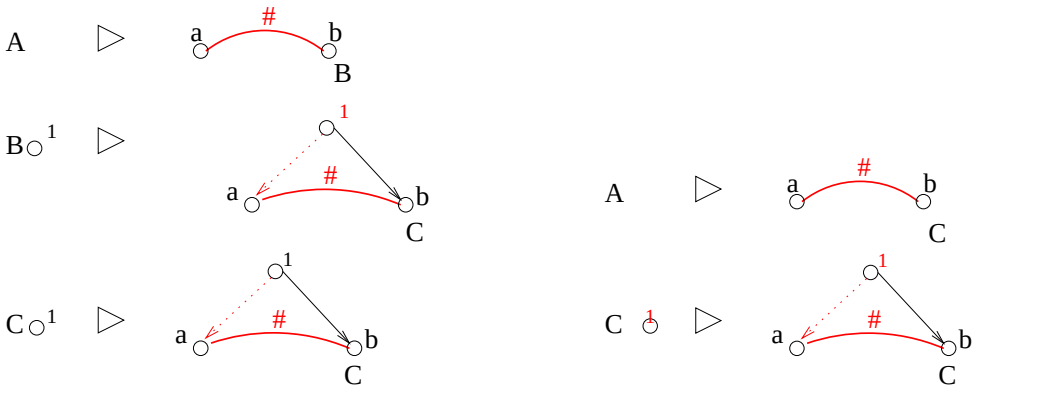
8.1.1 Redundancy elimination

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

$$\exists m, n \mid [\mathcal{G}^m(R)] = [\mathcal{G}^n(R')] \wedge X' \in \mathcal{G}^m(R) \wedge X' \text{ loops } \wedge X = X' \text{ (up to renaming)}$$

.

We suppress (X, R) redundant with (X', R') by removing (X, R) 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) Grammar with redundancy b) Equivalent grammar without redundancy

Figure 14: Redundancy elimination

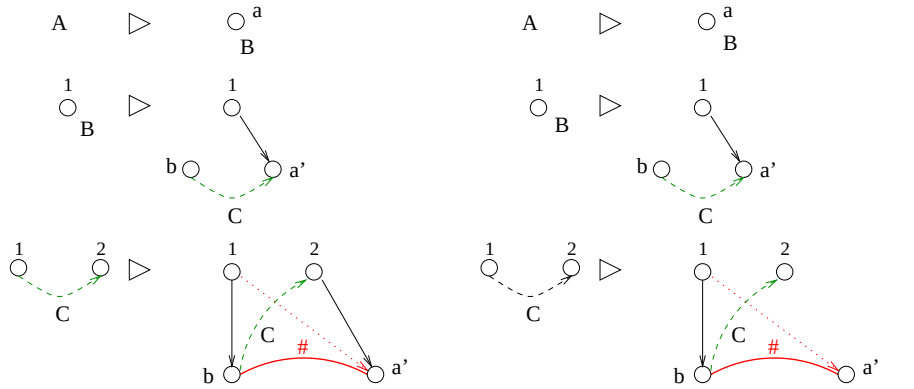
8.1.2 Removing infinite branching

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

For any looping rule (X, R) of $\mathcal{G}_P$, if there is a vertex kept all along the successive rewritings of R ($\exists (X, Red) \in App_X \mid red \neq \emptyset$), and a causality edge starts from this vertex, then there is an infinite branching, and the causality edge is removed from R.

For the example in Figure 15-a, we have :

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



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

Figure 15: Removing infinite branching

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.

This modification suppresses all infinite branchings. From the definitions of Section 8.1, 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 $\exists (X, red) \in App_X \mid red \neq \emptyset$.

8.2 Connections preservation

Removing infinite branching doesn't 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. Consequently, we can't have a message sent only once with multiple receptions.

Any vertex s_i involved in an infinite branching is in conflict with another vertex, say b_i, or is connected by a causality or an inheritance edge to a vertex, say c_i, in conflict with b_i (See Figure 16 for an illustration). As this infinite branching is caused by an iteration over a choice, at the occurrence $i-1$ of the choice, three cases may happen :

- Event b_i is chosen, there is an inheritance edge between b_{i-1} and c_i, so c_i and s_i are still connected if we remove edges (c_{i-1}, c_i) and (s_{i-1}, s_i).

- Event c_{i-1} is chosen, and two causality edges, from c_{i-1} to c_i, and from s_{i-1} to s_i will be removed. But since there is an inheritance edge from c_{i-1} to b_i, and c_i is connected to b_i via a conflict, events c_i and s_i remain connected.
- Any other action can be chosen, say d_{i-1}, and as there is an inheritance edge from d_{i-1} to c_i, vertices c_i and s_i remain connected.

Thanks to inheritance 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.

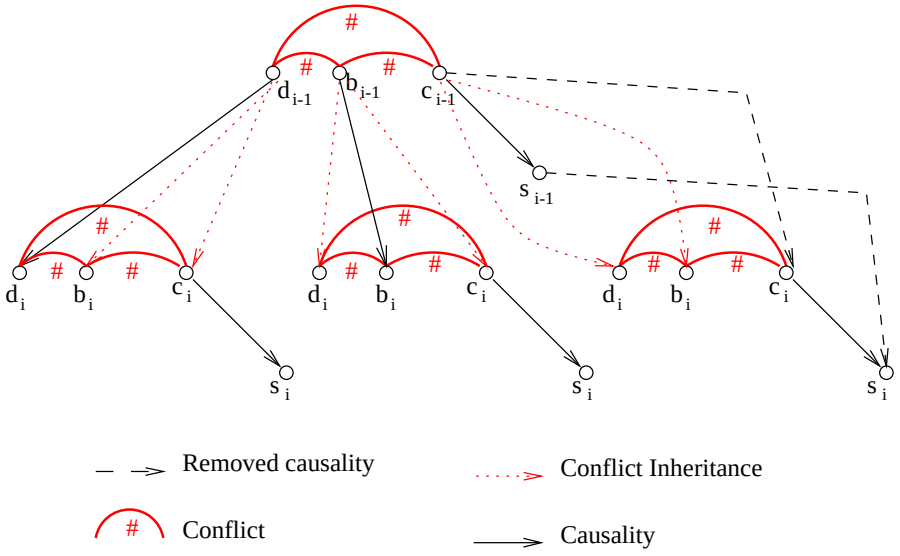


Figure 16: Connexity preservation

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 $\{(X_i, R_i)\}$, 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 $\{(X_i, R_i)\}$ 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.

8.3 Equivalence preservation

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

- $\mathcal{G}_1^w \equiv \mathcal{G}_2^w \Longrightarrow \mathcal{FBG}_1^w \equiv \mathcal{FBG}_2^w$?

  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, R_1) that does not loop in $\mathcal{G}_1$, and by a rule (X_2, R_2) that loops within $\mathcal{G}_2$. As $\mathcal{G}_1^w \equiv \mathcal{G}_2^w$, and as (X_2, R_2) loops, we know that (X_1, R_1) and (X_2, R_2) represent the same part of a recursion in an HMSC. But as (X_1, R_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, R_1) adds a new active instance to the HMSC, then, as (X_2, R_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, R_1) is the first occurrence of a choice, then the next occurrences of this choice will have to include inheritance edges. As (X_2, R_2) loops, it can not represent the same pattern than (X_1, R_1), and we can not have $\mathcal{G}_1^w \equiv \mathcal{G}_2^w$.

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

- $\mathcal{G}_1^w \equiv \mathcal{G}_2^w \Longleftarrow \mathcal{FBG}_1^w \equiv \mathcal{FBG}_2^w$?

  Let us suppose that $\mathcal{FBG}_1^w \equiv \mathcal{FBG}_2^w \wedge \mathcal{G}_1^w \not\equiv \mathcal{G}_2^w$, then there is a edge (v_x, v_r) of $\mathcal{G}_1^w$ that is not in $\mathcal{G}_2^w$. If (v_x, v_r) is in $\mathcal{G}_1^w$ but not in $\mathcal{G}_2^w$, then it has been deleted from $\mathcal{G}_1^w$ by the infinite branching elimination, so this edge is not unique, and there is an infinity of vertices v_{r_i} such that $\forall i, (v_x, v_{r_i}) \in \mathcal{G}_1$ (because we have eliminated all redundancies). Therefore, we have an infinite branching starting from a looping rule X, and $\forall i, v_{r_i}$ is in conflict with a set of vertex $\{b_i\}$, which are the minimal vertices of a branch $M; P$ of the currently examinated choice, and such that the instance $\phi(v_x)$ does not perform any action on this branch ($\phi(v_x) \notin \mathit{Act}(M; P_{[X:=\epsilon]})$).

  So, as the calculus of a finite branching form does not suppress any vertex or any conflict, there is also a repetition of a choice between a branch containing a v_{r_i}, and another where $\phi(v_x)$ does not perform an action within $\mathcal{G}_2^w$. Consequently, edges (v_x, v_{r_i}) belong to $\mathcal{G}_2^w$, which leads us to a contradiction.

  Consequently, $\mathcal{G}_1^w \equiv \mathcal{G}_2^w \Longleftarrow \mathcal{FBG}_1^w \equiv \mathcal{FBG}_2^w$ holds.

□

9 A decision procedure for HMSC equivalence

We first have seen that it is possible to generate a graph grammar defining an unique event structure from a HMSC. This graph grammar may comprise infinite branchings. We have shown that a modification of this grammar allows to eliminate infinite branching while preserving the decision procedure for isomorphism. 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 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 $\mathcal{FBG}_{P_2}$, using the algorithm defined in [2].

If P_1 and P_2 have the same normal forms up to a renaming, then they are equivalent.

Let us compare HMSC P_2 in Figure 17 to HMSC P_1 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_2 is isomorphic to P_1.

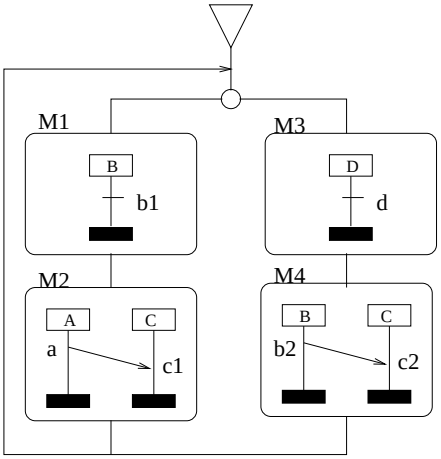


Figure 17: HMSC P_2

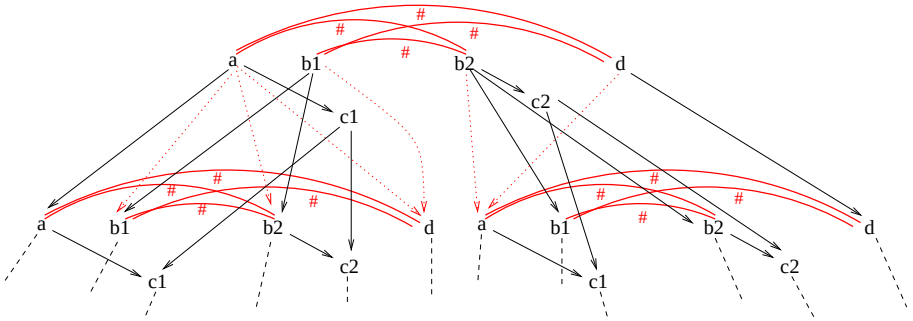


Figure 18: Covering graph generated by the normal form

10 Process divergence detection

As we already metioned, HMSCs allow for the definition of undesirable features. One of them is called *process divergence*. This section describes an adaptation of the algorithm proposed in [1] for detection of process divergence.

10.1 Process Divergence

A system contains a process divergence when an instance A can send an unbounded number of messages to an instance B without ever receiving an acknowledgment.

We can now adapt this definition to our graph grammar definition. There will be a process divergence whenever a rule X loops (this is a consequence of the infinite behaviour requirement), and when the derived pattern leading from X to X contains a communication between an instance A and B, and no event of A is causally dependent of an event of B.

10.2 Algorithm

A first step towards process divergence detection is cycle detection. For this purpose, we introduce a *derivation graph*, calculated from a graph grammar.

Let $\mathcal{G} = (G_0, \mathcal{R})$ be a graph grammar. The *derivation graph* $DG(\mathcal{G})$ of $\mathcal{G}$ is a directed graph indicating for each rule R of $\mathcal{G}$ which rules can be applied to the left part of R.

$DG(\mathcal{G}) =< \mathcal{H}(\mathcal{R}), T >$, where

- $\mathcal{H}(\mathcal{R})$ is the set of hyperarcs appearing in the left parts of the rules of $\mathcal{R}$.
- $H_1, H_2 \in T$ if H_1 is an hyperarc appearing in the left part of a rule of $\mathcal{R}$, and H_2 appears in the right part of the same rule.

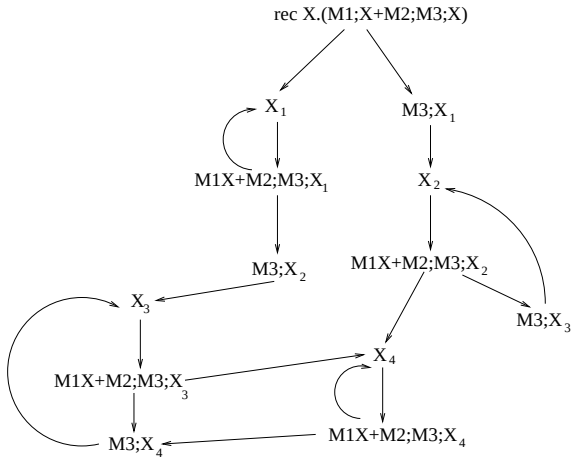


Figure 19: Derivation graph

The derivation graph of grammar in Figure 12 is represented in Figure 19.

A cycle in the derivation graph indicates that all the rules involved in the cycle loop. Each cycle of the derivation graph represents an infinite behaviour. A cycle basis of the derivation graph can be detected by a depth first traversal algorithm.

A basic cycle of the derivation graph is a word $C = r_1 r_2 ... r_n \in \mathcal{H}(\mathcal{R})^n$ with $n < |\mathcal{R}|$, such that $(r_n, r_1) \in T$ and $\forall i, j < n \mid i \neq j, r_i \neq r_j$.

For any couple of rules r_i, r_{i+1}, we define a communication graph $CG(r_i, r_{i+1})$ defining the communications in the right part of r_i leading from vertices of r_i to vertices of r_{i+1}. $\forall r_i, r_{i+1} \in C, CG(r_i, r_{i+1}) = \{(A, B) \in I^2 \mid A$ sends a message to B in the right part of rule r_i and the sending and receiving events are predecessors of the vertices contained in the hyperarc $r_{i+1}\}$.

$$CG(r_i, r_{i+1}) = \{(A, B) \in I^2 \mid \exists e_1 \in \mathcal{G}^1(r_i) \wedge \exists e_2 \in \mathcal{G}^1(r_i) \wedge \phi(e_1) = A \wedge \phi(e_2) = B \wedge A \neq B \wedge e_1 \longrightarrow e_2 \wedge (\exists s_A, s_B \in r_{i+1} \wedge e_1 \xrightarrow{*} s_A \wedge e_2 \xrightarrow{*} s_B)\}$$

For a cycle C, we compute the communication graph of C, which is the union of all the communication graphs.

$CG(C) = \bigcup_{i \in 1..n-1} CG(r_i, r_{i+1})$

If the communication graph of a cycle C contains no communication between strongly connected components, then there is no process divergence generated within C. On the contrary, if there is a communication from a strongly connected component $SC_1 \subset I$ to another strongly connected component $SC_2 \subset I$ then it means that the communication media used for this message could be overloaded.

A computation of strongly connected components can be performed by Tarjan's method [20]. Then, for each couple of communicating instances, we have to check whether the sender and the receiver are in the same component.

$X_3.(M1 + M2; M3; X)_3.X_4.(M1 + M2; M3; X)_4.(M3; X)_4$ is a basic cycle of the graph grammar in Figure 12. The communication graph associated to this cycle is the graph in figure 20.

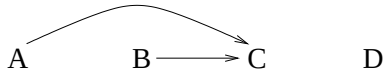


Figure 20: Communication graph for a basic cycle

This communication graph contains 4 strongly connected components { {A}, {B},{C},{D}}, and there are 2 inter-component communications { (A,B), (B,C)}. Though, the HMSC defined Figure 5 contains a process divergence.

11 Conclusion

First, a prime event structure semantics to HMSCs has been given, and then rules for computing a graph grammar that generates a covering graph of this structure have been defined. This allows us to decide the equivalence of HMSCs as the isomorphism of their covering graphs. This first step gives us a background for formal manipulations of HMSCs that we plan to develop within an UML environment.

References

[1] Ben-Abdallah.H,Leue.S

Syntactic Detection of Process Divergence and non-Local Choice in Message Sequence Charts in: E. Brinksma (ed.), Proceedings of the Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems TACAS'97, Enschede, The Netherlands, April 1997, Lecture Notes in Computer Science, Vol. 1217, p. 259-274, Springer-Verlag, 1997.

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

[3] Grabowski.J, Graubman.P, Rudolph.E. Towards a petri net based semantics definition for message sequence charts. Technical report, SDL'93:Using Objects,179-190, 1993.

[4] Habel.A. Hyperedge replacement: grammars and graphs. *Lecture Notes in Computer Science*, (643), 1989.

[5] Harel.D. LSCs: breathing life into message sequence charts *Weizmann Institute Tech. Report CS98-09*, April 1998.

[6] Heymer.S A non-interleaving semantics for MSC. *SAM98:1st conference on SDL and MSC*,281-290, 1998.

[7] ITU-T Message Sequence Chart (MSC) *ITU-T Recommendation Z120*, October 1996.

[8] Katoen.J.P, Lambert.L, Pomsets for message sequence charts. *SAM98:1st conference on SDL and MSC*,281-290, 1998.

[9] Kosiuczenko.P. Formalizing MSC'96: Inline expressions. Technical report, Insitut fur Informatik, Munich, 1997.

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

[11] Leue.S, Ladkin.P. Interpreting message flow graphs. *Formal Aspects of Computing*,7(5):473-509, 1995.

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

[13] Reniers.A, Mauw.S. An algebraic semantics for basic message sequence charts. Technical report, Departement of Mathematics and Computing Science, Eindhoven University of Technology, 1994.

[14] Mauw.S. The formalization of message sequence charts. Technical report, Eindhoven University of Technology, 1995.

[15] Reniers.A, Mauw.S. High-level message sequence charts. Technical report, Heindoven University of Technology, 1996.

[16] Muscholl.A, Peled.D, Su.Z Deciding properties for message sequence charts. *Lecture Notes in Computer Science*, (1378), 1998.

[17] Winskel.G, Nielsen.M, Plotkin.G. Petri nets, event structures and domains, part 1. *Theoretical Computer Science*, 13, 1981.

[18] Rudolph.E, Graubmann.P, Grabowski.J. Tutorial On Message Sequence Charts Computer Networks and ISDN Systems 28 (1996), 1629-1641

[19] Graubmann.P, Rudolph.E, Grabowski.J. Tutorial on message sequence charts (msc'96). FORTE/PSTV'96, october 1996.

[20] Tarjan.R Depth-First search and linear graph algorithms SIAM J. Comput, Vol. 1, No 2, June 1972

[21] OMG Unified Modelling Language 1.1, september 1997.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399