Scenario automata : theory and applications

Loïc Hélouët

Rennes 1 University

Document d'habilitation

pour obtenir

l'habilitation diriger des recherches de l' Université de Rennes 1

Spécialité : **Informatique**

préparée au laboratoire **IRISA - INRIA Rennes**

dans le cadre de l'École Doctorale **Matisse**

présentée et soutenue publiquement
par

Loïc Hélouët

le 17 mai - 2013

Automate d'ordres : Théorie et applications:
Scenario Automata: Theory and Applications

Jury

M.	Z,	Président du jury
P.S..	Thiagarajan,	Rapporteur
Paul.	Gastin,	Rapporteur
Roland.	Groz,	Rapporteur
Martin.	Leucker,	Examinateur
Madhavan.	Mukund,	Examinateur
Albert.	Benveniste,	Examinateur
Claude.	Jard,	Examinateur
Sophie.	Pinchinat,	Examinatrice

Résumé

Les automates d'ordres, plus connus sous le nom de Message sequence Charts (MSC), ont connu une énorme popularité depuis les années 1990. Ce succès est à la fois académique et industriel. Les raisons de ce succès sont multiples : le modèle est simple et s'apprend très vite. De plus il possède une puissance d'expression supérieure à celle des automates finis, et pose des problèmes difficiles. L'apparente simplicité des MSCs est en fait trompeuse, et de nombreuses manipulations algorithmiques se révèlent rapidement être des problèmes indécidables.

Dans ce document, nous revenons sur 10 années de recherches sur les Message Sequence Charts, et plus généralement sur les langages de scénarios, et tirons quelques conclusions à partir des travaux effectués. Nous revenons sur les propriétés formelles des Message Sequence charts, leur décidabilité, et les sous-classes du langage permettant la décision de tel ou tel problème. L'approche classique pour traiter un problème sur les MSCs est de trouver la plus grande classe possible sur laquelle ce problème est décidable. Un autre challenge est d'augmenter la puissance d'expression des MSCs sans perdre en décidabilité. Nous proposons plusieurs extensions de ce type, permettant la crétion dynamique de processus, ou la définition de protocoles de type "fenêtre glissante".

Comme tout modèle formel, les MSCs peuvent difficilement dépasser une taille critique au delà de laquelle un utilisateur ne peut plus vraiment comprendre le diagramme qu'il a sous les yeux. Pour pallier à cette limite, une solution est de travailler sur de plus petits modules comportementaux, puis de les assembler pour obtenir des ensembles de comportements plus grands. Nous étudions plusieurs mécanismes permettant de composer des MSCs, et sur la robustesses des sous-classes de scénarios connues à la composition. La conclusion ce cette partie est assez négative: les scénarios se composent difficilement, et lorsqu'une composition est faisable, peu de propriétés des modèles composés sont préservées.

Nous apportons ensuite une contributions à la synthèse automatique de programmes distribués à partir de spécification données sous forme d'automates d'ordres. Cette question répond à un besoin pratique, et permet de situer un role possible des scénarios dans des processus de conception de logiciels distribués. Nous montrons que la synthèse automatique est possible sur un sous ensemble raisonnable des automates d'ordres.

Dans une seconde partie de ce document, nous étudions des applications possibles pour les MSCs. Nous regardons entre autres des algorithmes de model-checking, permettant de découvrir des erreurs au moment de la spécification d'un système distribué par des MSCs. La seconde application considérée est le diagnostic, qui permet d'expliciter à l'aide d'un modèle les comportement d'un système réel instrumenté. Enfin, nous regardons l'utilisation des MSCs pour la recherche de failles de

sécurité dans un système. Ces deux applications montrent des domaines réalistes d'utilisation des scénarios.

Pour finir, nous tirons quelques conclusions sur les scénarios au regard du contenu du document et du travail de ces 10 dernières années. Nous proposons ensuite quelques perspectives de recherche.

Abstract

Partial order automata are more known under their standardized name "'Message Sequence Charts (MSCs). They have met a considerable interest during the last 15 years. This success is both industrial and academic, and has several reasons. First, the model is rather simple and can be learned very easily by engineers. Second, despite its apparent simplicity (MSCs are for instance more expressive than finite state automata), it has an interesting expressive power, and raises many difficult problems. Indeed, many algorithmic applications rapidly turn to be undecidable problems.

In this document, we collect and sumarize a part of the work accomplished on MSCS during the last decade, and draw some conclusions from the obtained results. We first focus on formal properties of MSCs, the decidability of several standard problems, and the definition of subclasses of the language allowing for the decision of some problems when the general case is undecidable. The standard approach to work with MSCs is to find the larger subclass of the language allowing for the decision of a given problem. Another challenge is to increase the expressive power of MSCs without loosing decidability of too many problems. We propose several extensions to the formalism allowing dynamic creation of processes, or allowing for the design of protocols comporting sliding windows behaviors.

As many formal models, MSCs can not exceed a limit size after which a diagram is not understandable for a human designer. A solution is then to build a specifcation in a modular way, and then to assemble the modules to obtain larger sets of behaviors. We propose several mechanisms to compose MSCs, and study the robustness of MSC sub-classes to composition. We then draw some conclusions from the properties of composition mechanisms described in this part of the document. Overall, composition is seldomly effective, and does not preserve formal properties of partial order automata.

In a second part of this document, we study possible applications for MSCs. We consider model checking problems, that can be used to discover design errors during distributed systems specification. The second application considered is diagnosis, which allows to retreive out of a model the explanations of some partial observation of an instrumented system. Last, we consider the applicability of MSCs to the search for security breaches in distributed systems.

To complete the work, we provide some conclusions on scenario models, based on the content of this document and on the experience gained these last 10 years. We then propose future research directions.

Contents

Acknowledgments

This document is a preliminary version, that will be corrected in the next coming weeks, and may be this is too early for acknowledgments. However, I feel indebted to several people. First of all, I would like to thank the reviewers of this document: P.S. Thiagarajan, Paul Gastin, and Roland Groz. The spontaneity with which you accepted to review this work was a real surprise and a real pleasure.

For administrative reasons, Martin Leucker will only be external examiner for this work. I am also very grateful to him, for accepting this role. Many thanks too to Madhavan Mukund for contribution to the jury, and for our scientific collaborations.

Many thanks too to the local members of the jury: Claude Jard, Albert Benveniste, Sophie Pinchinat. Beyond their contribution to this habilitation jury, they all have played an important role at different stages of my scientific career.

Many, many, many thanks to colleagues at IRISA: Blaise, Philippe, Hervé, and to former members of the lab: Akshay, Shaofa. It was a great pleasure to work with you, and [4] was a great experience, demonstrating if needed that scientific collaborations can be maintained over the world (and also around the clock).

I would also be very unfair not to mention the role of my employer, namely INRIA Rennes, first for hiring me, but also for funding several consecutive associated teams. The collaborations that started thanks to these teams had (and still have) huge influence on my work.

Many thanks to the runners of `courir@irisa.fr`. It is always a pleasure to run and talk with you. These relaxing moments are an important part of a subtle equilibrium bewteen work, family life, and leisure.

The list of people to thank does not stop here, and will be completed in the forthcoming issues of this document. But I can not end these acknowledgements without a word about Philippe Darondeau. During all these years, Philippe was an enormous source of knowledge, experience and an exemple of scientific rigor. Philippe's influence goes far beyond the contributions mentionned in this document. I have written many pages thinking that he would have enough time to see and comment them. "Salut" Philipppe, and thank you for all...

How to read this document ?

This document collects many contributions and facts on scenarios, and month after month have become a large document, that might be difficult to read entirely. The reason for including so much material is that I wanted this document to be self-contained. Hence, a part of the material presented at the beginning of the document is not my personal contribution to the scenarios community. However, I felt that it was important to recall at least the basic definitions of scenarios, for several reasons. First of all, many classes of scenarios have been proposed, and sometimes, the same class of language has different names in different papers. It was then important to fix a vocabulary. Second, in our studies of scenarios, we often compare expressiveness and decidability of new classes of scenario formalisms with formerly existing ones.

Readers who are familiar with scenarios and Message Sequence Charts can certainly skip most of the material in the first three chapters. To help readers, an index is provided at the end of the document, and can help finding the formal definition of a particular term.

Still for self-containment purposes, we have provided some proofs of theorems that appear in the document as an appendix.

Introduction

The term "scenario" in the formal methods community refers to many objects, ranging from a simple word $w \in \Sigma^*$ depicting in a sequential way the execution of a system which actions belong to an alphabet Σ, to more complex diagrams including control structures, data manipulation, time, exceptions, etc. Even if the interpretations may differ, the term scenario supposes a *partial and abstract* representation of some behavior. And most of the time, what people have in mind when using the term scenario is a *graphical representation of the behavior of a distributed system.* Many dialects have been proposed to define such collections of behaviors: Message Sequence Charts [81], Live Sequence Charts [38], interworkings [102], UML's sequence diagrams [119],... These formalisms are frequently designated under the very generic term "scenario language". This document addresses the theme of scenarios from a focused point of view, which is that scenario languages are formal models that depict (potentially infinite) sets of partially ordered multisets, obtained by assembling elements from a finite set of finite pomsets. This assembling mechanism can have the form of finite state automata labeled by pomsets (we will then talk about partial order automata), or equivalent models called MSC graphs, or High-level MSCs. More elaborated assembling mechanisms have been proposed (for instance using grammars), and the assembling rules can also be modified to enhance the expressive power of the model.

Scenario languages, and more precisely partial order automata such as High-level Message Sequence Charts have met a considerable interest from the mid 90's to the beginning of the 21^{st} century. This popularity came both from the industrial world and from the academic community. These models have engendered a consequent literature, on properties of partial order automata, their formal manipulation, extensions of the models, etc. In some sense, on may consider that scenarios benefited from a whole corpus of knowledge accumulated around language theory, communicating machines, traces, and many other formal models. As scenarios started to gain popularity in industry, it was the right time to equip them with formal semantics [126, 127] and algorithms, and to highlight which problems were feasible or conversely untractable (see for instance [13, 14, 112, 113, 115]) for scenario descriptions.

From an industrial point of view, scenarios have been originally designed to show execution traces of distributed systems modeled with SDL [83]. This language was called Message Sequence Charts (MSCs). A preliminary version called MSC'92 was published in 1992. It was then standardized by the International Telecommunication Union (ITU) as the Z.120 recommendation. In their early version, MSCs were nothing more than simple chronograms that all computer science engineers draw to explain how components of a distributed protocol interact. Indeed, MSC'92

comport only two graphical elements: processes are represented by vertical lines, and exchange asynchronous messages, that are represented by arrows from the sending process to the receiving process. This basic model was then used as a basis for the TTCN test language, and extended with several composition operators, allowing sequential composition of scenarios, iterations, choices, etc. The enhanced model was then called MSC'96 (due to the fact that a standardized version of the language was published by the ITU in 1996). The latest version of standard Z.120 was published in February 2011 [81]. In parallel, a similar kind of diagram called Sequence Diagrams was developed for the Unified Modeling Language [119].

The appealing and intuitive aspect of Message Sequence Charts is usually the reason given to explain why MSCs gained such a popularity. Indeed, only a few concepts are needed to design a MSC, and a small chronogram is usually understood in no time by any user familiar with distributed protocols. Figure 1-a gives an example of the kind of chronograms used in MSCs to describe interactions among processes. However, as for many other models, intuitivity of the language is questionable when the size of models increases to dozens of interacting processes exchanging hundreds of messages. Furthermore, the composition operators of MSC'96 increase the complexity of the model even if a clear and formalized semantics was written [126]. Despite this complexification of scenarios, MSC'96 were used at large scale at Motorola to discover design errors in the initialization protocols of their cellular phones [18]. One may wonder if this success story is due to the language itself or to the huge modeling effort, and if similar work could have been done with another language. Yet, this remains a success story for the language.

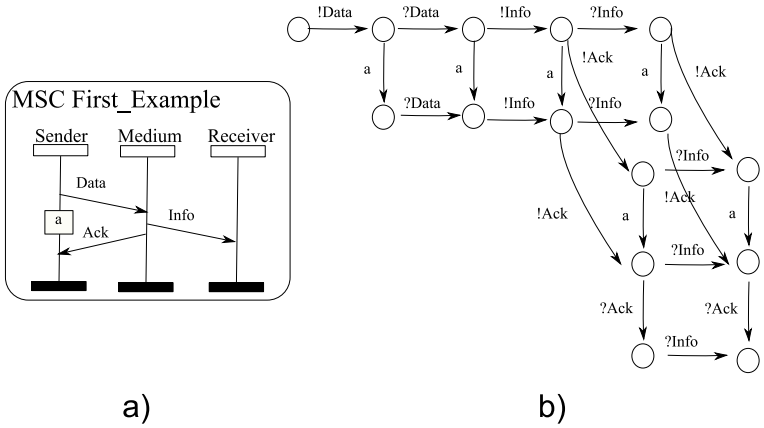


Figure 1: An example MSC a), and an equivalent interleaved representation b)

Another argument in favor of MSCs is the old debate between interleaved and non-interleaved formalisms as a model for concurrency [34]. Indeed, a model such as MSCs use partial orders as basic building construct of the language. It is well known that a partial order representation is more concise than an equivalent interleaved model. Let us compare a description provided by a diagram in the MSC'92 formalism (i.e. a partial order), and an automaton depicting the same set of executions, i.e. in which concurrency is represented by interleaving all transitions symbolizing each event. Then, the size of the automaton can be exponential in the size of the MSC'92 model. The example of Figure 1-a) and its interleaved counterpart in Figure 1-b) illustrate this phenomenon. MSCs can hence be exponentially more concise

than automata. We do not claim here that one way of representing executions is better than the other: modeling with partial orders also has a cost. For instance, checking global behavioral properties of an MSC model (such as model checking an LTL formula) can only be done by considering all possible configurations of the model, that is computing the states of the equivalent interleaved model. Hence, the supposed exponential gain for a partial order representation is clearly lost when the addressed problems need can only be addressed by computing an interleaved model. However, when the properties to check on an MSC are of more local nature, for instance considering sequences of events occurring on a single process, causal dependencies in a specification, or verifying that a sequence of messages occurs in a MSC, there is no need to compute an equivalent interleaved representation. Hence, when some property to consider can be expressed in terms of a property of some partial order, modeling with MSCs results in a high complexity gain. Another advantage in using MSCs is that causality in systems is explicitly represented. This information usually disappears in interleaved models such as automata. We will show in chapter 10 of this document that causality is important to address some information flows problems.

The last argument in favor of MSCS is their expressive power. Standard finite state automata can only represent systems with a finite state space (or an approximation of infinite state systems). Partial order automata can be used to represent systems with infinite state space. Of course, this expressive power has a cost: many verification problems are undecidable for MSCs. This does not mean however that formal analysis of MSCs is always untractable, or that MSCs are not usable as a formal modeling tool. First of all, some problems that are undecidable for models such as Turing Machines, communicating automata, or counter machines have a solution (and even is some cases a trivial solution) in partial order automata. Furthermore, experience shows that many problems have solutions for non-trivial subclasses of partial order automata. This is for instance the case for globally-cooperative HMSCs [56], that allow for some kind of model checking. Interestingly, non-globally cooperative specifications are usually seen as degenerate models, and this observation frequently applies to a given problem and its associated decidable subclass. We address decidability issues and define existing subclasses of partial order automata in chapter 2, and provide decidable verification techniques in chapter 8.

When modeling distributed systems with scenarios, one should keep in mind the incomplete and abstract nature of the model. Scenarios are not a programming language, and it seems difficult to build a complete distributed system only via a partial order automaton definition. This raises several issues: first of all, if a single partial order automaton describes only a partial and abstract set of behaviors, can we cover a larger subset of all the behaviors of a system by composing several automata? Can we slightly extend the model to allow the design of more elaborated behaviors? The second issue is verification: once again, if a scenario specification describes only an abstract view of a system, model checking techniques should be used with care: the model and the property to check should address behaviors at the same abstraction level. Furthermore, if scenarios describe only a subset of all possible behaviors of a system, proving a property of the model does not necessarily implies that this property holds on the real system. The last issue is the relationship between scenario languages and the implementation that they represent. In the

worst situation, the model and the implementation are designed at different moments by different actors, and nothing guarantees coherence of the two objects. However, it seems reasonable to consider that scenarios have been proposed at requirement time, and then used to program the system, or conversely that the running system's execution traces have been used to build the scenario model. In these more favorable cases, scenarios should depict (yet at an abstract level) a reasonable subset of all behaviors of the running system, hence making pertinent some formal analyzes. One way to ensure that a scenario and an implementation have similar behaviors is to perform automatic synthesis of code from scenarios. Yet, all existing techniques address only a restricted subset of specifications. Furthermore, scenarios are not a programming language, and the synthesis step should be seen as a first step in the design process to produce a code skeleton. We address the synthesis problem in chapter 7 of this thesis.

Even if scenarios are seen as a subset of all possible behaviors of a real system, several interesting applications remain. Diagnosis is one of these applications. Diagnosis algorithms are used to recover explanations from a partial observation of a real execution. This technique is useful to provide explanations when a system has crashed, or to detect that a fault has occurred. One can also use diagnosis techniques to detect that no explanation exists for an observed run of a system in a proposed scenario model. Detecting that an observed run of a system is not contained in the scenario model is a crucial point: it helps comparing a model and a running application, and can be used as a way to enrich an existing model. In some sense, scenarios are well adapted to problems that aim at proving the *existence* of good/bad properties of a system. We will also show (chapter 10) that diagnosis techniques can be used as a monitoring tool to detect security breaches.

In this document, we present the research accomplished from 2001 to 2011, which aimed at a better understanding and better usability of partial order automata and close scenario models. This work can be classified into three main research directions:

- increase the expressive power of partial order automata, without loosing too much decidability.
- Solve new problems on partial order automata and their variants, and when these problems are undecidable, find the largest decidable subclass of the model for the considered problem.
- Find new practical and effective applications that go beyond simple protocol modeling.

This document is organized as follows: Chapter 1 is a rapid introduction to partial order automata and to their semantics. Chapter 2 lists several well known problems for partial order automata, such as model checking, etc. In this chapter, we also show that many interesting problems are indeed undecidable, and show for some problems the syntactic subclasses of partial order automata for which some problems become decidable.

Chapter 3 shows that the expressive power of standard HMSCs is not sufficient to model very common protocols that use sliding windows mechanisms. This chapter also shows a way to extend the initial model to overcome this problem, and recalls some results originally proved in [58]. The proposed extension consists in allowing

the sending of a message and the corresponding reception to appear in separate diagrams (which is not allowed in the standard model)

Chapter 4 proposes another way to extend HMSCs to model sliding windows. The extension relies on closure of the behavior of each process by a commutation relation. This extension also solves the expressive power problem highlighted in chapter 3.

Chapter 5 shows several approaches to enrich partial order automata with standard operators that are frequently seen in process algebra : product, shuffle, projections. This chapter assembles the contributions of several papers, and provides a feedback on all these composition schemes. Among several things, this chapter shows that the syntactic subclasses of partial order automata defined in chapter 2 are in general not closed under product, shuffle or projection, and that consistency of views defined as partial order automata is in general undecidable. This hinders the practical use of such operators.

Chapter 6 introduces another extension of HMSCs to allow for the description of protocols involving an arbitrary number of participating threads, created online during execution of the protocol.

Chapter 7 addresses the implementation problem: for a given HMSC, describing the behavior of a fixed number of processes, can we derive by simple projection an implementation (a set of communicating machines) with identical behavior ? In general, the answer to this question is no. This implementation problem is of major importance for practical use of MSCs, as outside the embedded and critical software world, engineers are very often reluctant to design a model when it can not be translated easily into code.

The second part of the document is devoted to more practical aspects of partial order automata, and shows several application domains for the models described in the first part of the thesis.

Chapter 8 addresses verification of partial order automata using partial order logics. The first logic that we consider is MSO for MSC. This model only expresses facts on the shape of the partial orders generated by a partial order automaton; We show that MSO for MSCs is decidable for partial order automata, but also for dynamic MSCs. A practical consequence of this decidability result is that diagnosis, which can be expressed as a model checking problem, is also decidable for dynamic MSCs. However, satisfiability of MSO over MSCs is undecidable, except with restrictions on the shape of MSCs that can be models of a formula. This fixes the limits of partial order assembling, and completes the picture of chapter 5: if one can not decide if a specification is consistent (i.e. described at least a run), then the considered formalism is not adapted to system design. In this chapter, we also propose a partial order logic, which is a kind of LTL with finite orders replacing usual atomic propositions. This logic can be used to express collections of known facts about a system, of the form : "when behavior A occurs, then behavior B always occurs in the future". This model is very expressive, and mimics in some sense the template MSCs that were proposed in [58]. Unsurprisingly, many problems such as diagnosis or existence of a run satisfying a formula are undecidable for specifications given with this partial order logic. However, we show that with a finite horizon, diagnosis becomes decidable.

Chapter 9 shows how to use HMSCs to perform diagnosis of distributed imple-

mentations. Considering a HMSC as a faithful model of the running application, the question addressed in this work is how to recover all explanations for a failure of the system from a partial observation collected before a fault. This problem is decidable in HMSCs, and the solution can be expressed as another partial order automaton, opening the way to compositional diagnosis.

The last application for scenarios is security, and is considered in chapter 10. We consider two main applications, for which scenarios seem to be well adapted. The first one is covert channels detection. A covert channel is a mean to transfer information from one user to another in a system among parties that should not communicate otherwise (or should only communicate via specified and monitored means). Scenarios are well adapted to find such security leaks, as causality helps highlighting intentional information passing. The second security application considered is anomaly detection. First of all, as scenarios are a partial specifications of usual behaviors of a system, comparing the actual behavior of a running system with partial order models provides a way to detect when the system is used in an abnormal way. We show in this chapter that anomaly detection form scenarios can be brought back to a simplified instance of diagnosis.

The last chapter of this document draws several conclusions on the model from the experience gained during this decade. We also sketch some perspectives on applications and formal development for the model, and propose future research directions.

Chapter 1

Preliminaries

Là, tout n'est qu'ordre et beauté, luxe, calme et volupté.
Here, everything is beauty and order, luxury, quietness and delight.
[Charles Baudelaire, l'initiation au voyage]

1 Introduction

In this chapter, we recall all basic models and notations that will be used throughout the document. We present a formal definition for partial order automata, and several semantics for this language, defined in terms of pomsets, words, and transition systems. The formalism is usually defined using two specification layers. At the lowest layer, partially ordered multisets describe finite interactions of a finite set of processes. These diagrams are often called basic MSCs (or bMSCs for short), but throughput the document, we will simply call them MSCs. MSCs can be composed by the second layer of the formalism, namely partial order automata. Partial order automata are simply finite automata labeled by MSCs. In the Z.120 standard, these partial order automata are called High-level MSCs (HMSCs for short), and we will use interchangeably the terms partial order automata and HMSCs.

This chapter first defines MSCs and their semantics, and then considers some formal properties and definitions attached to these basic diagrams that will be used later in the document. We then give a formal definition of partial order automata and their semantics.

2 Message Sequence Charts

Message Sequence Charts is a language standardized by the International Telecommunication Union (ITU) [82]. The language is composed of simple and intuitive elements that are combined to define the behavior of processes that communicate asynchronously. In a MSC diagram, processes are represented by vertical lines, and message exchanges by horizontal arrows from the sending process to the receiving process. Local actions executed by a process are represented by boxes, localized on the executing process, and labeled by the name of the action. Each process has its own clock, and is independent from all others. All communications among processes are performed via asynchronous message exchanges. The actions and communication events localized on the same process are ordered from top to bottom. Figure 1.1

shows an example of MSC in which three processes *Sender*, *Medium* and *Receiver* exchange messages of type *Data*, $Info$ and *Ack*. This MSC also contains an atomic action a.

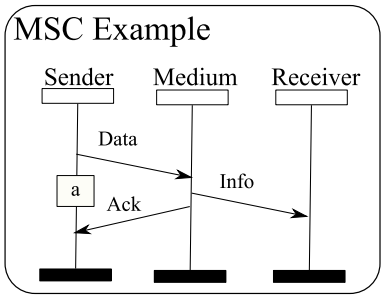


Figure 1.1: An example MSC

As one can easily see from the above diagram, MSCs describe the covering relation of some pomset. In the following definition, we however differentiate the ordering originating from local ordering of events on a process, and the ordering due to message exchanges.

Definition 1 *A* Message Sequence Chart *over a set of processes $\mathcal{P}$ is a tuple $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \phi)$, in which E represents a set of events. $E = \biguplus E_p = E_S \uplus E_R \uplus E_A$ can be partitioned according to the process that executes each event ($E_p = \{e \in E \mid \phi(e) = p\}$, or according to the type of event considered : message sending, reception, internal action (also called* atomic action*). We then have $E_S = \{e \in E \mid \exists f, (e, f) \in \mu\}$, $E_R = \{e \in e \mid \exists f, (f, e) \in \mu\}$ and $E_A = E \setminus (E_S \cup E_R)$.*

For every process $p \in \mathcal{P}$, the relation $<_p \subseteq E_p \times E_p$ is a total ordering on events located on process p. The mapping $\mu : E_S \longrightarrow E_R$ associates a sending event with the corresponding reception.

$\alpha : E \longrightarrow \Sigma$ is a function that associates a label to each event, $\phi : E \longrightarrow \mathcal{P}$ is a function that localizes each event on a process (i.e. $\phi(e)$ is the process that executes e). The labels attached to events also denote the type of an event. We will write $\alpha(e) = p!q(m)$ iff $e \in E_S$ is the sending of message m by process p to process q, $\alpha(e) = q?p(m)$ iff $e \in E_S$ is a reception of a message m sent by process p on process q, and $\alpha(e) = p(a)$ iff e is the execution of an atomic action a by process p. The labeling of an event must be coherent with the message mapping, that is if $f = \mu(e)$ then $\alpha(e) = p!q(m)$ and $\alpha(f) = q?p(m)$ for some p, q, m.

Usually, several additional assumptions are done on MSCs. From the definition, every local ordering on processes $<_p$ must be a total order. However, we also impose that $\leq = \left(\bigcup <_p \cup \mu\right)^*$, a relation called the *causal order relation* of M is a *partial order* over E. In this document, we will also use the *covering relation* of $\leq$, that is $<= \{(e, f) \in \leq \mid \nexists x, x \neq e, x \neq f, e \leq x \leq f\}$. For a MSC M, we will denote by $Min(M) = \{e \in E | \forall e' \in E,\, e' \leq e \Rightarrow e' = e\}$ the set of minimal events for the causal order relation, i.e the set of events that have no causal predecessor. Similarly, we will denote by $Max(M) = \{e \in E \mid \forall e' \in E, e' \geq e \Rightarrow e' = e\}$ the set of event that have no causal successor in M. We will also denote by $Min_p(M)$ the unique event on process p (if it exists) that has no predecessor on p, and by $Max_p(M)$ the

unique event on process p (if it exists) that has no successor on p. We will denote by $|M|$ the size of M, that is the number of events in M.

The Z.120 standard does not impose anything on communications, but simply indicates that messages are asynchronous. It is also frequently assumed that communication between processes are implemented via FIFO buffers. Communications between a pair of processes are then FIFO (there is one single FIFO buffer from p to q, and all messages from p to q are received in the order they were sent) or weak-FIFO (there is one FIFO buffer per type of message). In a FIFO setting, message overtaking is not allowed, and in the weak-FIFO setting, overtaking of messages of the same kind is forbidden.

- FIFO : $\forall e, f, e', f'$ such that $e, f \in E_p, e', f' \in E_q$ and $e' = \mu(e), f' = \mu(f)$ $e <_p f \iff e' <_q f'$.
- Weak-FIFO : $\forall e, f, e', f'$ such that $\lambda(e) = \lambda(f) = p!q(m), \lambda(e') = \lambda(f') = q?p(m)$ $e, f \in E_p, e', f' \in E_q$et $e' = \mu(e), f' = \mu(f)$, $e <_p f \iff e' <_q f'$

The most used hypothesis is the FIFO one. In the rest of this document, we will denote by M_ϵ the empty MSC (i.e. that contains no event).

As one can figure from the example of Figure 1.1, MSCs have a visual and intuitive aspect. As suggested by the formal definition, a MSC M can be seen as a labeled pomset $(E, \leq, \lambda)$. These pomsets can be linearized to obtain the set of executions (sequences of actions) that are compatible with the causal order $\leq$.

Definition 2 *A* linear extension *of a MSC $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \phi)$ with n events is a sequence $e_{i_1}.e_{i_2}\dots e_{i_n}$ of events of M such that for every $j > k$ $e_{i_j} \not\leq e_{i_k}$. A* linearization *of a MSC M is a word w of Σ^* such that there exists a linear extension $e_{i_1}.e_{i_2}\dots e_{i_n}$ of M and $w = \alpha(e_{i_1}).\alpha(e_{i_2})\dots\alpha(e_{i_n})$. The* language *defined by a MSC is the language $\mathcal{L}(M)$ of all linearizations of M.*

3 Formal Properties of MSCs

3.1 Bounds

Bounds on the contents of communication channels is a major question to address for distributed systems. The intuitive meaning of an action $p!q(m)$ is that a new message is sent from p to q. This message will transit through a network, and will be buffered by q before consumption. Hence, every time a new message is sent, the size of a reception buffer may increase. A desirable property is that a system can run with a finite amount of memory dedicated to communication buffers: if no bound is guaranteed at execution time, the communication buffers can be overloaded. When this situation occurs, messages can be lost, but buffers overload can also lead to machines failure, or even be exploited to create security breaches. In this setting, the property to ensure is that no run of a system can let a buffer content exceed a maximal size (we hence talk about universal bounds on runs).

Bounds are also important for implementation reasons. Ensuring that a (potentially infinite) set of scenarios X *can be* executed with some maximal bound on

buffer contents is one of the first properties to ensure to guarantee that the specification is safely implementable. Even if no universal bound exists for the whole set X, it is still interesting to guarantee that every MSC in X can be executed without exceeding fixed maximal buffers sizes. Existence of such bound ensures that buffer overloading is not *enforced* by some run of a MSC in X. Within this setting, we hence talk about *existential bound*.

For a single finite execution, and for a finite set of finite MSCs, a maximal bound always exists, and one can easily compute the maximal size of communication buffers during an execution. Note however that existence of universal or existential bounds is not always a decidable property for infinite state systems. For instance, deciding if a reset Petri net is (universally) bounded is undecidable [45]. In this document, we will show that boundedness problems are often decidable properties of scenario models.

Definition 3 *Let w be a word of Σ^*, and $p, q \in \mathcal{P}$ be two processes. Let us denote by $|w|_{p!q}$ the number of messages sent from p to q in word w, and by $|w|_{p?q}$ the number of messages received on p and sent by q in w. A* linearization *of Σ^* is a word $w \in \Sigma^*$ such that for any prefix v of w, and every pair of process p, q, $|v|_{q?p} \leq |v|_{p!q}$ (there is no more reception of messages than sendings). A linearization $w \in \Sigma^*$ is b-bounded if and only if for every pair of processes $p, q \in \mathcal{P}$ and any prefix v of w, $|v|_{p!q} - |v|_{q?p} \leq b$. A MSC M is existentially b-bounded if and only if there exists a b-bounded linearization of M. A MSC M is universally b-bounded if and only if every linearization of M is b-bounded. A MSC is existentially bounded (resp. universally bounded) if and only if there exists some $b \in \mathbb{N}$ such that M is existentially (resp. universally) b-bounded.*

Definitions of existential and universal boundedness extends to sets of MSCs the obvious way: a set of MSCs $\mathcal{X}$ is existentially *b-bounded* (resp. universally b-bounded) if all MSCs $X \in \mathcal{X}$ are existentially b-bounded (resp universally b-bounded); $\mathcal{X}$ is *existentially (resp. universally) bounded* if it is existentially (resp. universally) b-bounded for some b.

Theorem 1 *[96] Checking if a MSC M is existentially b-bounded can be done in linear time. Checking that a MSC M is universally b-bounded can be done in $O(|M|^2)$*

The intuition for the first part of the theorem is that any b-bounded prefix of a linearization of a b-bounded MSC can be extended to a b-bounded linearization. Hence it is sufficient to linearize M while enforcing bound b on every channel. The second result comes from the fact that enforcing a bound on all executions is equivalent to requiring that on a channel, the $n + b^{th}$ sending must occur after the n^{th} reception. If adding such dependency makes execution impossible, then M is not universally b-bounded. This property is checked by detecting cycles in the extended dependency relation, whence the quadratic complexity.

Definition 4 *Given a MSC $M = (E, (<_p)_{p \in \mathcal{P}}, \alpha, \mu, \phi)$ and a non-negative integer b, let Rev_b be the binary relation on E such that $e\, Rev_b\, e'$ if and only if, for some p and q in $\mathcal{P}$ and $m \in \mathcal{M}$, e is the i-th event on process p with the label $\lambda(e) = p\,?\,q\,(m)$ and e' is the $i + b$-th event on process q with the label $\lambda(e') = q\,!\,p\,(m)$. We also define $Rev_{\geq b} = \cup_{b' \geq b} Rev_{b'}$.*

Proposition 1 (lemma 2 in [96]) *A MSC M is $\exists$-B-bounded if and only if the relation $< \cup Rev_B$ is acyclic, if and only if the relation $< \cup Rev_{\geq B}$ is acyclic.*

If M is $\exists$-b-bounded then M is $\exists$-b'-bounded for all $b' \geq b$, because $Rev_{b'}$ is included in the least order relation containing Rev_b and $\bigcup_{p\in\mathcal{P}} <_p$. Let us consider a MSC M with linearization $p!q(m_1)^b\,(q!p(m_2)\,p?q(m_2))^b\,q?p(m_1)^b)$ for some fixed b. Let (s_i, r_i) denote the i^{th} pair of events $(p!q(m_1), q?p(m_1))$ and (s'_i, r'_i) the i^{th} pair of events $(q!p(m_2), p?q(m_2))$, then $s_b <_p r_1\, Rev_{(b-1)}\, s'_b <_q r_1\, Rev_{(b-1)}\, s_b$ is a cycle, and one can conclude that M is not existentially b-bounded.

This construction of the Rev_b relation will be used in chapter 5 to check if a composition of HMSCs remains existentially bounded. In the next chapter, we will also show that under some restrictions, it suffices to consider linearizations of MSC up to some bound on channels to check some properties. To this extent, we define the notion of bounded restriction of a language.

Definition 5 *Let M be a MSC, with linearization language $\mathcal{L}(M)$. The restriction of $\mathcal{L}(M)$ to bound $n \in \mathbb{N}$ is denoted by $\mathcal{L}^b(M)$ and is the restriction of $\mathcal{L}(M)$ to $b-$bounded words.*

3.2 Concatenation, atoms

So far, we have mainly addressed properties of finite MSCs. To obtain more elaborated sets of scenarios, one need to add to MSCs usual composition operators that are frequently seen is process algebra: choices, sequence, iteration, ... The first useful operator is *sequence* of MSCs, also called *weak sequential composition* in the literature. This operation allows for the merging of two diagrams, and is the basic construct for HMSCs, MSC graphs and all their variants. All other operators will be obtained with partial order automata.

Definition 6 *Let $M_1 = (E_1, (<_{1,p})_{p\in\mathcal{P}}, \alpha_1, \mu_1, \phi_1)$ and $M_2(E_2, (<_{2,p})_{p\in\mathcal{P}}, \alpha_2, \mu_2, \phi_2)$ be two MSCs defined over disjoint sets of events. The* sequential composition *of M_1 and M_2, denoted $M_1 \circ M_2$ is the MSC $M_1 \circ M_2 = (E_1 \cup E_2, (\leq_{1\circ 2,p})_{p\in\mathcal{P}}, \alpha_1 \cup \alpha_2, \mu_1 \cup \mu_2, \phi_1 \cup \phi_2)$, in which $\leq_{1\circ 2,p} = (\leq_1 \cup \leq_2 \cup\{(e_1, e_2) \in E_1 \times E_2 \mid \phi(e_1) = \phi(e_2)\})^*$, where $f_1 \cup f_2$ denotes a function defined over $Dom(f_1) \cup Dom(f_2)$, that associates $f_1(x)$ to every $x \in Dom(f_1)$ and $f_2(x)$ to every $x \in Dom(f_2)$.*

Let us underline the fact that $\mathcal{L}(M_1).\mathcal{L}(M_2) \subseteq \mathcal{L}(M_1 \circ M_2)$, but that in general, both languages are not equal. Indeed, in $M_1 \circ M_2$, events of M_2 can be executed before all events of M_1 are executed.

The sequential composition of MSCs is defined for MSCs over disjoint sets of events. However, a MSC over a set of events E can be considered as a representant for an infinite class of isomorphic MSCs over other arbitrary sets of events. For two MSCs M and M', we will write $M = M'$ when M and M' are isomorphic, that is when there exists an injective map h from E to E' that preserves locality, local ordering, messages, and labeling. If we define events as integers, and if we fix an arbitrary order on processes, we can define a canonical representant for each isomorphism class of MSCs. Similarly, slightly abusing the definition of sequential composition, we will write $M \circ M'$ without checking that M and M' are defined

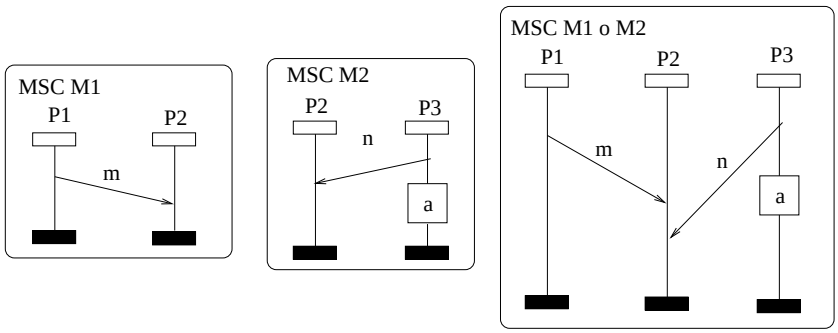


Figure 1.2: An example of sequential composition

over disjoint sets of events, that is we consider that in $M \circ M'$, the MSC M' is systematically an isomorphic copy of M' defined over event that do not appear in M. This will allow us to write expressions of the form $M \circ M$. Notice that many MSCs can be expressed indifferently as a single diagram, or as a concatenation of several smaller MSCs. This immediately raises the question of whether a given MSC can be decomposed into smaller *factors*.

Definition 7 *A MSC $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \phi)$ is an* atom *if an only if, for every pair of MSCs N, N' such that $M = N \circ N'$, we have $N = M$ and $N' = M_\epsilon$ or $N = M_\epsilon$ and $N' = M$. An* atomic partition *of M is a set of MSCs $\{N_1, \ldots, N_k\}$ defined over sets of events $\{E_1, \ldots E_k\}$ such that $E = \bigcup_{i\in 1..k} E_i$, and there exists $i_1, i_2, \ldots, i_k, M = N_{i_1} \circ \cdots \circ N_{i_k}$. MSCs in a partition of M are called* atoms *of M.*

Definition 8 *Let $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \phi)$ be an MSC over set of events E. The* connection graph *$Conn(M) = (E, \rightarrow_{Conn})$ of M is defined by $v_1 \rightarrow_{Conn} v_2$ if either $\phi(v_1) = \phi(v_2)$ and $v_1 \lessdot v_2$, or one of (v_1, v_2), (v_2, v_1) is a message in M (edges are added from receives to associated sends).*

Proposition 2 *[71] An MSC M is atomic if and only if the connection graph $Conn(M)$ is strongly connected.*

This property is rather straightforward: a MSC is an atom if it can not be separated into two (or more) non-trivial pieces without cutting a message. It will be important later in chapter 5 to test whether a chosen linearization of a projection of a MSC is an atom. Indeed, we will also show that a connection graph can be maintained online while unfolding a HMSC, hence allowing for online construction of atoms. Furthermore, it gives us the following result:

Theorem 2 *[59, 71] Let $M = (E, \leq, \lambda, \mu, \phi)$ be a MSC. Checking whether M is an atom and finding an atomic partition of M can be done in $O(|E| + 2.| \leq |)$.*

The proof for this theorem is rather simple: checking atomicity or finding a partition resumes to finding connected components in the connection graph of M. It can be done using Tarjan's algorithm [133]. We do not detail this algorithm, and refer interested readers to [71]. Figure 1.3 shows a decomposition of a non-atomic MSC into atoms (denoted by dashed lines).

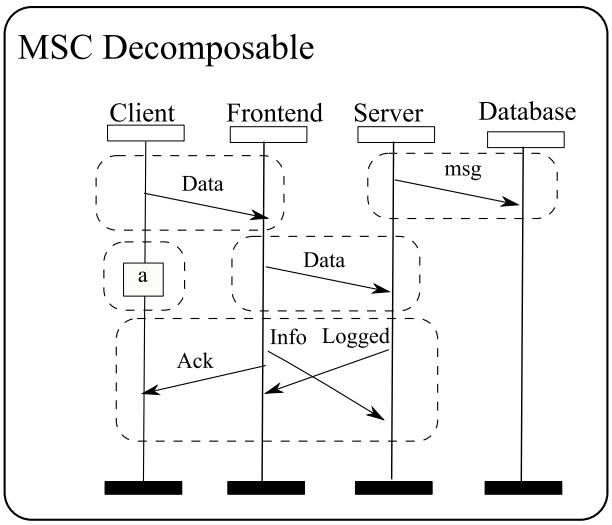


Figure 1.3: Decomposition of a MSC into atomic factors

Note that an atomic partition may contain isomorphic atoms. For a MSC M, we will denote by $Atoms(M)$ the quotient of the partition of M modulo isomorphism. Slightly abusing the notation, we will consider that each element of $Atoms(M)$ is a MSC, and is a canonical representative of it isomorphism class.

Definition 9 *Let M be a non atomic MSC, and let $Atoms(M) = \{M_1, \ldots, M_k\}$. The* atomic language *of M is the language $\mathcal{L}^{at}(M)$ composed of words $M_{i1} \ldots M_{ik} \subseteq Atoms(M)^*$ such that $M_{i1} \circ \cdots \circ M_{ik} = M$.*

Note that the atomic language of a MSC may contain several words. We will say that two MSCs M and M' are independent if they are defined over distinct sets of processes. When two MSCs M and N are independent, their order of concatenation does not change the obtained MSC : $M \circ N = N \circ M$. Reusing the vocabulary from traces, we will say that M and N *commute*, and write $M||N$. Of course, these definitions apply to atoms. We will say that two sequences of atoms are equivalent if they generate the same MSC. Using this commutation relation, one can rewrite a sequence of atoms $M_1 \ldots M_i.M_{i+1} \ldots M_k$ into an equivalent sequence $M_1 \ldots M_{i+1}.M_i \ldots M_k$ as soon as $M_i||M_{i+1}$. The equivalence class $[M_1 \ldots M_i.M_{i+1} \ldots M_k]_{||}$ is defined as the closure of $\{M_1 \ldots M_i.M_{i+1} \ldots M_k\}$ by the rewriting rule up to commutation.

3.3 An useful abstraction: the communication graph

Sometimes, reasoning on MSCs does not imply studying linearizations nor a partial order, but only who communicates with whom. This information can be represented as a communication graph, i.e. a directed graph which edges are processes, and which vertices symbolize the fact that a process communicates with another process. We will show in the next chapter that interesting subclasses of partial order automata are defined from properties of communication graphs.

Definition 10 *Let $M = (E, \leq, \lambda, \mu, \phi)$ be a MSC over a set of processes $\mathcal{P}$. The* communication graph *of M is a graph $CG_M = (\mathcal{P}, V)$, where $V \subseteq \mathcal{P}^2$, and $(p, q) \in V$ if and only if there exists $(e, f) \in \mu \cap E_p \times E_q$.*

Figure 1.4 shows the communication graph for the MSC of Figure 1.1. It contains the three processes *Sender*, *Medium*, *Receiver*, and edges from *Medium* to *Receiver*, from *Sender* to *Medium*, and from *Medium* to *Sender*.

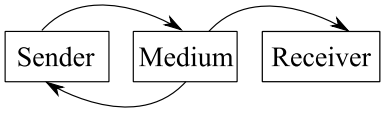


Figure 1.4: An example of communication graph

Communication graphs have interesting properties with respect to concatenation. Indeed, we have

$$CG_{M_1 \circ M_2} = CG_{M_1} \cup CG_{M_2} = CG_{M_1 \circ M_2}$$

4 Partial order automata, HMSCs

As indicated in previous section, MSCs alone are not sufficient to express interesting MSC languages, and a designer rapidly needs control structures such as loops, choices, etc. to design requirements. The usual way to obtain infinite MSC languages comporting orders of arbitrary sizes is to use partial order automata (or a close variant called MSC graphs), often called High-level MSCs (or HMSCs for short). Other higher-level structures such as *inline expressions* have also been proposed to extend MSCs with adequate control structures. In the rest of this chapter, we will only focus on partial order automata, and more precisely on HMSCs.

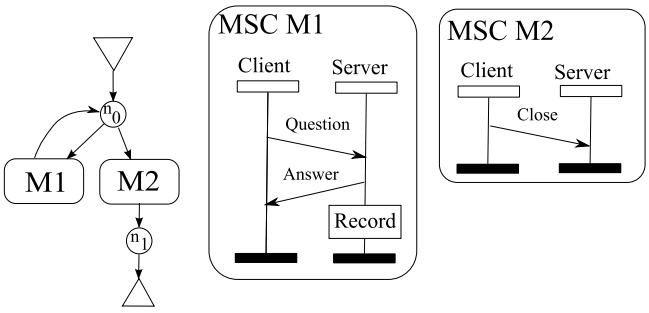


Figure 1.5: An example HMSC

Definition 11 *A* High-level MSC *(HMSC) is a tuple* $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$*, where* N *is a set of nodes,* $\longrightarrow \subseteq N \times \mathcal{M} \times N$ *is a transition relation,* $n_0 \in N$ *is the initial node of* H*, and* F *is a set of accepting nodes.*

Roughly speaking, HMSCs are automata labeled by MSCs. As for any kind of automaton, we can define concepts such as transitions, paths, languages, etc.

Definition 12 *Let* $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$ *be a HMSC. A* path ρ *of* H *is a sequence of transitions* $t_1.t_2 \ldots t_k$ *such that for every* i *in* $1..k$*,* $t_i \in \longrightarrow$ *is of the form* $t_i = (n_i, M_i, n'_i)$*, and for every* i *in* $1..k-1$*,* $n'_i = n_{i+1}$*. A path* ρ *is* initial *if and only if it*

starts from node n_0 *(i.e.,* $t_1 = (n_0, M_1, n'_1)$*), and it is an* accepting path *if and only if it terminates on a node of* F *(i.e.,* $t_k = (n_{k-1}, M_k, n_k)$ *for some* $n_k \in F$*). A path is a* cycle *if and only if it starts and ends on the same node. A path* ρ *is a prefix of another path* ρ' *if and only if* ρ' *is of the form* $\rho' = \rho.\rho''$ *for some* ρ''*. A path is* maximal *if and only if it starts from the initial node of* H*, and it is not the prefix of another path (a maximal path can be finite or infinite).*

Definition 13 *Let* $\rho = t_1.t_2 \dots t_k$ *be a path of a HMSC* H*. The MSC associated to path* ρ *is the MSC* $M_\rho = \big(\dots(M_1 \circ \phi_2(M_2) \cdots \circ \phi_k(M_k)\big)$ *where each* ϕ_i *is an isomorphism that guarantees* $\phi_i(E_i) \cap \bigcup_{j \in 1..i-1} E_j = \emptyset$*.*

More intuitively, the MSC associated to a path is obtained by concatenating MSCs encountered along this path. In the sequel, to simplify notations, we will forget the isomorphisms used at concatenation time, and we will simply write $M_\rho = M_1 \circ M_2 \circ \cdots \circ M_k$, even if some MSCs M_i and M_j are occurrences of the same MSC diagram. The automaton structure as well as the concatenation operator allows for the definition of : a set of path, a set of MSCs, and a set of linearizations.

Definition 14 *A HMSC* $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$ *defines:*

- *A set of* accepting paths*, denoted* $\mathbb{P}_H$
- *A set of sequences of MSCs* $L(H) = \{M_1.M_2 \dots M_k \mid \exists \rho = t_1 \dots . t_k \in \mathbb{P}_H \wedge \forall i \in 1..k, t_i$ *is of the form* $(n, M_i, n')\}$
- *A* partial order family*, or MSC language:* $\mathcal{F}_H = \{M_\rho \mid \rho \in \mathbb{P}_H\}$
- *A linearization language:* $\mathcal{L}(H) = \bigcup_{M \in \mathcal{F}_H} \mathcal{L}(M)$

One may immediately notice that the linearization language of a HMSC is not necessarily regular. For instance, the HMSC of Figure 1.6 describes behaviors of the form $Client!Server(Data)^n.Client!Server(Disconnect).Server?Client(Data)^n.Server?Client(D$ Note also that none of the languages defined in definition 14 take into account the control structure of the automaton. We will however see that this control structure is a central element to define syntactic subclasses of HMSCs.

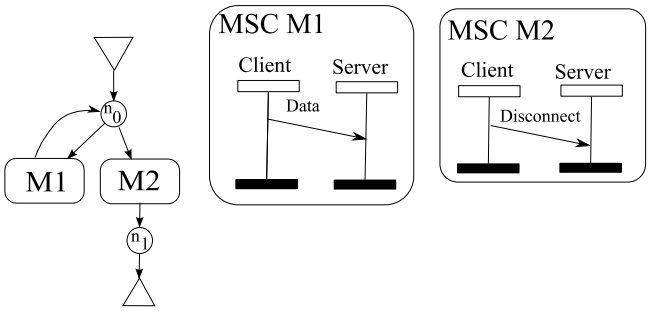


Figure 1.6: A HMSC with non-regular set of linearizations.

Definition 15 *Let $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$ be a HMSC. A node $c \in N$ is a choice node if and only if there exists two distinct path ρ_1, ρ_2 starting from c.*

Choice nodes in HMSCS allow for the definition of alternatives. Note however that choices can not always be interpreted as a local branching in the control flow of one single process. Indeed, choosing a scenario or another involves all processes that carry minimal events in one branch of the choice. Consider for instance the HMSC $H1$ of Figure 1.7. Node n_0 is a choice node. However, processes A, B and C, D do not communicate. To realize H_1, these processes have to *decide globally* the branch that they want to follow. This situation is called *non-local choice*, and will be discussed more in detail in the next chapter. This remark highlight the fact that even if HMSCs are automata, their control structure should not be strictly interpreted as a control flow of some program. For instance, the semantics of a path $\rho = (n, M, n')(n', N, n'')$ is $M_\rho = M \circ N$. Hence, in the behavior described by ρ, some events of N might be executed before M has completely been executed. And when M and N are independent, N might be completely executed before M. Hence, sequence can not be interpreted as M is executed before N (even if this behavior is allowed).

Similarly a too operational interpretation of HMSCs choices supposes that all involved processes reach an agreement on which branch to execute as soon as one of the processes executes the first event in one branch. Implicitly, this supposes that there exists a global memory mechanism that lets all processes know which scenario was chosen. However, here is no bijection between the set of paths $Paths(H)$ of H and the partial order family $\mathcal{F}_H$ that H generates, and the same MSC can be obtained as a concatenation of distinct sequences of MSC concatenated along distinct branches. Consider for instance the HMSC H_2 of Figure 1.7. The path $\rho_1 = (n_0, M1, n_1)(n_1, M_4, n_2))$ and $\rho_2 = (n_0, M_2, n_1)(n_1, M_3, n_2))$ generate the same behavior containing two occurrences of message msg and of message $query$. Hence considering a choice in the higher level structure is not always a safer interpretation. A safe interpretation is then to see **HMSCs as generators for MSC languages**.

4.1 Bounds and atoms in HMSCs

One can immediately notice that HMSCs are more expressive than finite automata: the linearization language of a HMSC is not necessarily regular. This is for instance the case for the example of Figure 1.6, which linearizations can not be recognized by a finite automaton. However, a HMSC with a single process and labeled by MSCs that contain atomic actions only is an automaton.

Definition 16 *A HMSC H is universally bounded by some integer b iff $\mathcal{L}(H)$ is universally b-bounded. H is existentially b-bounded iff for every M in $\mathcal{F}_H$, there exists a b-bounded linearization of M.*

Note that a HMSC $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$ is necessarily existentially bounded for some value b_H that is the maximal value among minimal existential bounds for MSCS in $\mathcal{M}$. Indeed, for sequence of MSCs $M = M_1 \circ M_2 \circ \cdots \circ M_k$ allowed by H, the word $u_1.u_2 \ldots u_k$, where each u_i is a linearization of M_i is a linearization of M. In particular, one can chose each u_i as a word that minimizes the buffer contents. We will call this value b_H the *existential bound* of H.

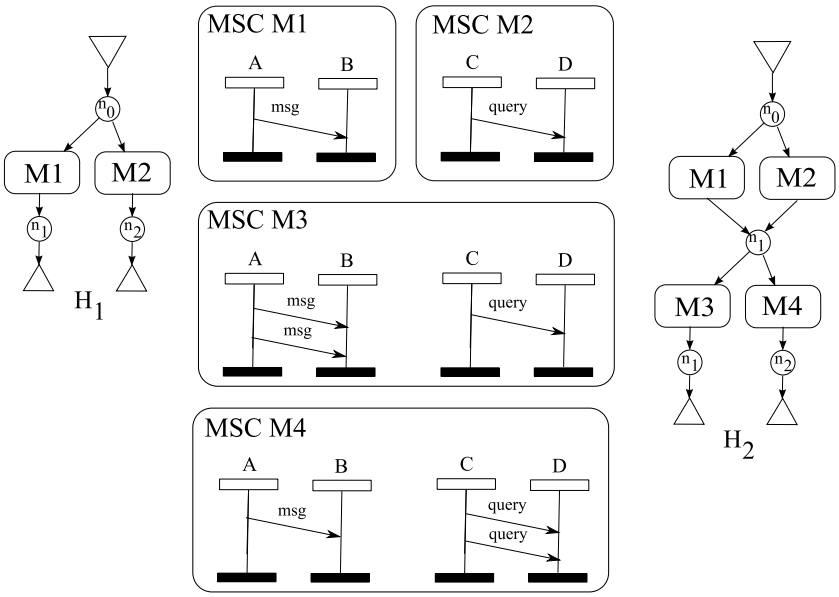


Figure 1.7: Choice nodes in two HMSCs H_1 and H_2

Definition 17 *Let $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$ be a HMSC. For every $b \in \mathbb{N}$ we will denote by $\mathcal{L}^b(H)$ the restrictions $\mathcal{L}(H)$ to b-bounded linearizations. We will denote by $Atoms(H) = \bigcup_{M \in \mathcal{M}} Atoms(M)$ the set of atoms appearing in some MSC of H. We will call the* atomic language of H *and denote by $\mathcal{L}^{at}(H) = \bigcup_{M \in \mathcal{F}_H} \mathcal{L}^{at}(M)$ the set of all sequences of atoms in $Atoms(H)^*$ which concatenation is a MSC in $\mathcal{F}_H$.*

Note that in general, $\mathcal{L}^b(H)$ and $\mathcal{L}^{at}(H)$ need not be regular languages, even when considering that b is the existential bound of H. Let us emphasize that the non-regularity of HMSCs linearizations is not simply due to the fact that communication buffers are not always universally bounded. The simple example of figure 1.8 below shows a situation in which only one communication between two processes p and q occurs (an hence communication buffers are universally bounded by one). The linearization language of this example is $L = \{u.p!q(m).v.p?q(m) \mid |u.v|_a = |u.v|_b\}$, which is not regular, but universally 1-bounded. Considering the atomic language does not yield a regular language either.

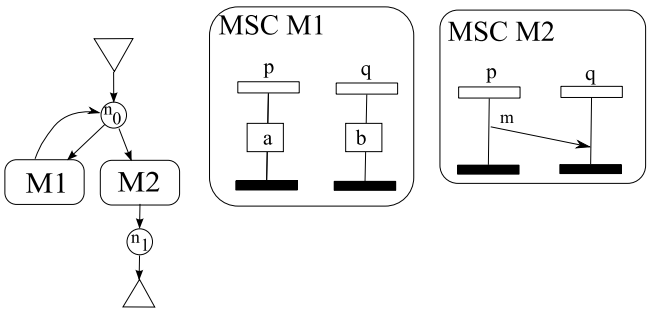


Figure 1.8: A non-regular and 1-bounded HMSC

4.2 A transition system recognizing $\mathcal{L}(H)$

HMSCs linearizations can be recognized by an infinite transition system, which states are *configurations of the HMSC*. Configurations memorize a part of a path in a HMSC, and the events that have not yet been executed in the MSCs labeling transitions of the followed path. This information is kept as a sequence of *segments*.

Definition 18 *Let H be a HMSC. A* segment *is a couple $(T, Done_T)$ where: $T = t_1 \dots t_k$ is a path of H, which transitions are respectively labeled by $M_1, \dots, M_k$, and $Done_T : \mathcal{P} \times |T| \to \mathbb{N}$ is a function that associates to every pair (p, i) an integer that memorizes the rank of the last event executed on process $p \in \mathcal{P}$ in transition t_i of T. We furthermore require that:*

- *Each process that has executed an event of a MSC M_i has completed all its actions in preceding MSCs of the segment, i.e., $\forall p \in \mathcal{P}, \forall i \in 1..k, Done_T(p, i) > 0 \Longrightarrow \forall j < i, Done(j, p) = |\{e \in E_j \mid \phi(e) = p\}|$*
- *The restriction of every $M_i, i \in 1..k$ to already executed events is closed by causal precedence, i.e. calling $e_{M_i,j,p}$ the j^{th} event on process p, then $\forall q, \forall p, |\downarrow (e_{M_i,Done(i,p),p}) \cap \{e \in E_i \mid \phi(e) = q\}| \leq Done_T(i, q)$*
- *any transition in T comports at least one event that is not yet executed: $\forall i \in 1..k, \exists p, Done_T(i, p) < |\{e \in E_i \mid \phi(e) = p\}|$*

A configuration can be seen as a memory containing all events that remain to be executed so that a started linearization belongs to $\mathcal{L}(H)$. Configuration are defined as sequences of segments, plus the last node visited along a path of H.

Definition 19 *A* configuration *of a HMSC is a pair $C = (S_1 \dots S_k, n)$, where $S_1 \dots S_k$ is a sequence of segments, and n is a node of H. Let $C = (S_1 \dots S_k, n)$ and $C' = (S'_1 \dots S'_{k'}, n')$ be two configurations. We will say that a move from C' to C' with action a exists and write $C \xrightarrow{a} C'$ when one of the following cases holds:*

- *there exists a segment $S_i = (T_i, Done_i)$ of C that contains a transition $t_{i,j}$ is which* **more that one event** *remains to be executed. There exists an event $e = e_{M_i,Done_i(j,p)+1,p}$ such that $\alpha(e) = a$ and all events on process p have already been executed in segments $S_1 \dots S_{i-1}$, as well as in all transitions preceding $t_{i,j}$ in T_i. C' is the configuration C in which $Done'_i(j, p) = Done_i(j, p) + 1$. In particular, $(T_i, Done_i)$ remains a segment.*
- *there exists a segment $S_i = (T_i, Done_i)$ of C that contains a transition $t_{i,j}$ in which $e = e_{M_i,Done_i(j,p),p}$ is* **the last event to execute** *in M_j, $\alpha(e) = a$, and such that all events on process p have been executed in segments $T_1 \dots T_{i-1}$ as well as in all transitions preceding t_{ij} in T_i. C' is the configuration $(S_1 \dots S_{i-1}.S.S_{i+1} \dots S_k, n)$, where S is the segment obtained by removing $t_{i,j}$ from T_i (this needs recomputing $Done_i$).*
- *There exists a path ρ starting from n, ending in a node n', and an event e, minimal in M_ρ such that:*

$\alpha(e) = a$, and no prefix ρ' of ρ contains e. All event on $\phi(e)$ have been executed in $S_1 \dots S_k$. $C' = (S_1 \dots S_k.(\rho, Done_{k+1})$,n') where $Done_{k+1}(j,p) = 0$ for every process $p \neq \phi(e)$ or if M_j does not contain event e, and $Done_{k+1}(j, \phi(e)) = 1$ if M_j contains e.

A configuration is *final* if and only if it is of the form (ϵ, n) with $n \in F$: intuitively, all events along a final path of H have been executed. A transition system recognizing linearizations of H is obtained by building inductively the set of configurations $\mathcal{C}_H$ and the transitions $\delta_H \subseteq \mathcal{C}_H \times \Sigma \times \mathcal{C}_H$ between these configurations, starting from the initial configuration (ϵ, n_0). Obviously, a word $w \in \Sigma^*$ is a word of $\mathcal{L}(H)$ if and only if allows a sequence of moves from the initial configuration (ϵ, n_0) to a final configuration. One can notice that the transition system defined this way can be infinite, non-deterministic, and that the set of configurations that can be reached in one move from a given configuration C can also be infinite.

5 Conclusion

In this chapter, we have defined the main objects that will be used throughout this document: MSCs, HMSCs and their languages, atoms,... In the next chapter, we will focus on classical problems frequently addressed on formal models, such as model checking, minimality of a model, existence of bounds, etc.

An important thing to remember at this stage of the document is that a HMSC is a finite state automaton, labeled by partial orders, and that can be seen as the *generator* of a (potentially infinite) partial order family $\mathcal{F}_H$ that may contain partial orders of arbitrary size. This partial order family can also be seen as a compact representation for the language $\mathcal{L}(H)$, that contains all linearization of partial orders in $\mathcal{F}_H$, and can be of exponentially greater size. This language can be recognized by a potentially infinite and infinitely branching transition system.

The problems addressed on HMSCs will usually find efficient solutions when they can be solved by checking a syntactic property of the considered HMSC, or solved on a reasonably small subset of orders in $\mathcal{F}_H$, or words in $\mathcal{L}(H)$. If a question needs to consider the whole interleaved language $\mathcal{L}(H)$, or the underlying transition system, then the problem will often be undecidable (this is for instance the case for model checking of global temporal logics). Even in the decidable cases, the complexity due to considering all interleavings is rapidly redhibitory.

Chapter 2

Standard problems for partial order automata

Il n'y a pas de problèmes ; il n'y a que des solutions. L'esprit de l'homme invente ensuite le problème. Il voit des problèmes partout.
There is no problem. There are only solutions. Then man's mind invents a problem. It sees problems everywhere.
[André Gide]

1 Introduction

In this chapter, we address standard problems for formal models: comparison of two models, comparison of a model with regular languages, search for a canonical representation,....

We first recall several well known results that show that many problems are undecidable for partial order automata. These negative results are mainly due to the expressive power of the formalism, that can encode rational relations or Mazurkiewicz traces. All these negative results could definitely hinder practical use of partial order automata. However, we also show in this chapter that several reasonably expressive and decidable subclasses of the language allow for the decision of most of the addressed problems.

2 Negative results

2.1 HMSC Comparison

Comparing the semantics of two distinct models is an natural question. This question is even more crucial for partial order automata, as we have seen in chapter 1 that a non-atomic MSC can be generated by concatenations of different sequences of MSCs. Even if HMSCs are a graphical model, finding that two models are equivalent is not always straightforward. A question that is hence frequently addressed is whether, for two specifications H_1, H_2 we can say that H_2 is a *refinement* of H_1 (i.e. $F_{H_1} \subseteq \mathcal{F}_{H_2}$). Another usual question is whether $F_{H_1} \cap \mathcal{F}_{H_2} = \emptyset$. Intuitively, one may consider H_1 as a set of behaviors that must be avoided, and H_2 as a specification

under study. Answering the problem $F_{H_1} \cap \mathcal{F}_{H_2} = \emptyset$? means checking if H_2 contains none of the bad behaviors collected in H_1.

Theorem 3 *Let H_1, H_2 be two HMSCs, and let R be a regular language. The following problems are undecidable:*

Orders	*Linearizations*	*Regularity*
i) $\mathcal{F}_{H_1} \cap \mathcal{F}_{H_2} = \emptyset$? *[115]*	*iv)* $\mathcal{L}(H_1) \cap \mathcal{L}(H_2) = \emptyset$?	*vii)* $R \cap \mathcal{L}(H_1) = \emptyset$? *[14]*
ii) $\mathcal{F}_{H_1} = \mathcal{F}_{H_2}$? *[58]*	*v)* $\mathcal{L}(H_1) \subseteq \mathcal{L}(H_2)$?	*viii)* $\mathcal{L}(H_1) \subseteq R$? *[33]*
iii) $\mathcal{F}_{H_1} \subseteq \mathcal{F}_{H_2}$?	*vi)* $\mathcal{L}(H_1) = \mathcal{L}(H_2)$?	*ix)* $R \subseteq \mathcal{L}(H_1)$? *[33]*
		x) *is* $\mathcal{L}(H_1)$ *regular? [33, 74]*

Most of the above mentioned undecidability results come from the relation between HMSCs and rational traces [43]. We refer interested readers to the references given for items $i), ii) vii), x)$ for more information. The decidability of *iii)* would imply that *ii)* is also decidable, so *iii)* is undecidable. Items $iv), v), vi)$ are direct consequences of $i), ii), iii)$ as the existence of a MSC in $\mathcal{F}_{H_1} \cap \mathcal{F}_{H_2}$ implies the existence of a word in $\mathcal{L}(H_1) \cap \mathcal{L}(H_2)$, and the converse also holds if we assume FIFO MSCs. These undecidability properties have also been demonstrated in [33], by reduction from undecidable problems on rational relations. Indeed, a HMSC labeled by MSCs that comport two processes and atomic actions can encode very easily a rational relation in $X^* \times Y^*$. The first process carries actions from alphabet X and the second process actions from alphabet Y. Each MSC hence encodes a relation between a word of X^* and a word of Y^*. Decidability of item *viii)* would imply that on can decide whether the intersection of $\Sigma^* \setminus R$ and $\mathcal{L}(H_1)$ is empty or not, which would imply that problem *vii)* is decidable. Last, *ix)* can be used to encode a universality problem for rational trace languages.

The proofs of some items in Theorem 3 refer to rational traces, rational relations, etc. Undecidability in HMSCs can however be brought back to undecidability of well known problems. For instance, $\mathcal{F}_{H_1} \cap \mathcal{F}_{H_2} = \emptyset$ can be used to encode an occurrence of the well know Post's correspondence problem (PCP). We give a reduction from PCP to this intersection problem in appendix. The reason why we highlight this proof in particular is that undecidability in HMSCs can frequently be brought back to this encoding.

2.2 Confluence

The confluence problem is a standard question for several models such as traces. The main question addressed by confluence is whether two distinct choices from a given configuration depicting independent behaviors describe the beginning of maximal runs that should be considered as equivalent. We will write $M \sqsubseteq M'$ if M is a prefix of M'.

Definition 20 *Let H be a HMSC. H is said* confluent *if and only if, for every pair ρ, ρ' of maximal paths of H such that M_ρ et $M_{\rho'}$ are independent, there exists a path γ of H such that $M_\rho \sqsubseteq M_\gamma$ et $M_{\rho'} \sqsubseteq M_\gamma$*

The intuition behind confluence is that when a choice between independent behaviors exist (M_ρ and M'_ρ in the definition), then performing one behavior should not prevent the other one to occur, and there is a behavior including M_ρ and M'_ρ.

Theorem 4 *[113] Checking whether a HMSC is confluent is undecidable.*

This problem can easily be brought back to an intersection problem (and hence encode a PCP) as we are looking for a common behavior M_γ. The confluence problem is tightly connected to the interpretation of choices in HMSCs. Very often in specification languages, an alternative between two behaviors is considered as a control point in a system where some decision is taken. However, this is exactly the meaning of choices in HMSCs. Considering HMSCs as generators for partial orders, choices are means to append more than one scenario to an already generated prefix. Considering a choice as a control point is sometimes dangerous, as the decision to behave according to a scenario or according to another is a global decision involving several processes. This global decisions is not necessarily distributable, as we shall see in chapter 7.

2.3 Useless Branches

As we have seen in previous chapter, there is more than one way to obtain the same MSC by concatenation of atoms (or even of non-atomic MSCs). A natural question to address is whether a HMSC contains distinct runs generating the same partial order. This can help, for instance reducing the size of the model, or even generating a canonical version of a HMSC. We will show however that this problem is again undecidable.

Definition 21 *Let H be a HMSC, and c be a choice node of H. Let ρ be a finite path of H starting from node c. Path ρ is a* useless branch *if and only if, for every path ρ' such that $\rho.\rho'$ is an accepting path starting from c, there exists another accepting path γ that does not have ρ as prefix, and such that $M_\gamma = M_{\rho.\rho'}$.*

Obviously, detecting a useless branch of a HMSC resumes to testing inclusions of the set of MSCs $\mathcal{F}_1 = \{M_{\rho.\rho'} \mid \rho.\rho' \text{ maximal path starting at node } c\}$ into the set of MSCs $\mathcal{F}_2 = \{M_\gamma \mid \gamma \text{ maximal path starting from } c \wedge \gamma \neq \rho.\gamma'\}$. This problem is hence clearly undecidable. This means in particular that when a new branch is appended at a choice node in a HMSC, one can not decide if this new branch adds some behavior to the original specification.

2.4 Races

Race is often described as a semantical weakness of MSCs [13, 113]. The usual interpretation of the ordering along a process line is that all events depicted in a certain order on a process **must** occur in that order in any execution of the considered MSC. Muscholl et al [113] remark that this assumption implicitly means that receiving a message is in fact a consumption of this message from a buffer, that is decided by the receiving process according to the total ordering described in the MSC. However, some systems may work without buffering mechanisms, and reception of a message hence means the "physical arrival" of the message on an agent. If the message is not consumed immediately, it might be lost. Hence, strict ordering of two receptions of messages coming from different agents may not be implementable. For this reason, Muscholl et al [113] propose two interpretations of

ordering in MSCs. The *visual order* on events, which is the standard interpretation of MSCs, and the *causal order*, which does not impose ordering of receptions for messages originating from distinct processes. An MSC contains a race when both orders are not equal. One can consider that races are ambiguities in models that may lead to wrong interpretations at implementation time, and hence should be avoided, or at least documented.

For HMSCs, the definition of races is more involved. A HMSC H may define a behavior M in which two receptions $e = p?q(m)$ and $f = p?r(n)$ are visually, but not causally ordered. This will not be considered as a race if H contains another MSC M' that differs from M only by reordering $f \leq e$. The main intuition is that if a visual ordering is ambiguous, but the additional behaviors that are allowed by the causal ordering are visually represented in another path of H, then the visual ordering can not be misinterpreted.

Definition 22 *Let M be a MSC. The* visual order *of M is the order $\leq = (\bigcup_{p\in\mathcal{P}} <_p \cup \mu)^*$. The* causal order of M *is the ordering $\prec = (\mu \cup \bigcup_{p\in\mathcal{P}} <_p \setminus \{(e,f) \in E_p^2 \cap (E_S \times E_R \cup E_R \times E_R)\} \cup \{(e,f) \in E \cap E_R \mid \phi(e) = \phi(f) \wedge \exists e' <_q f', e = \mu(e'), f = \mu(f')\})^*$. M contains a* race *if and only if $\leq \neq \leq$.*

Despite the apparently complex definition of visual ordering, the intuition is rather simple: the visual ordering relaxes the order among sendings and receptions, and among receptions on the same process, except when the received messages originate from the same sending process. In such case, Muscholl et al consider that FIFO ordering holds, and can help receiving messages in their order of emission. Checking if a MSC M contains a race can be done in $O(|M|^2)$. There has been several definitions for races [13, 15, 113, 125], but all definitions highlight a discrepancy between the visual ordering that orders all event sequentially along processes, and a semantic variant of this interpretation of MSCs that relaxes ordering on receptions.

As the ordering among events changes depending on whether the interpretation for ordering is causal or visual order, we will distinguish two sets of linearizations for a MSC, and denote by $Lin^{\leq}(M)$ the set of linearizations of the visual order of M. Recall that this is the standard semantics of MSCs, and when no ambiguity can arise, we will simply write $Lin(M)$ instead of $Lin^{\leq}(M)$. Similarly, we will denote by $Lin^{\prec}(M)$ the set of linearizations of $\prec$. Note that by definition, for any MSC M, we have $\prec \subseteq \leq$.

Definition 23 *Let H be a HMSC. H is* race-free *if and only if $Lin^{\prec}(H) = Lin^{\leq}(H)$*

Theorem 5 *Deciding is a HMSC H is race-free is undecidable.*

The proof of this theorem comes from the fact that checking equality of $Lin^{\prec}(H)$ and $Lin^{\leq}(H)$ can encode an equivalence problem for traces with two distinct commutation relation, which is know undecidable. Details of the proof can be found in [113].

3 Positive results

All the negative results listed in previous section could be seen as a huge drawback for order automata, and mean that HMSCs are in fact not adapted as a specification language. Fortunately, there are several decidable syntactic subclasses of HMSC for which most of the properties listed in this chapter are decidable. Before reviewing some of these classes, we show some positive results that apply to HMSCs without restriction.

3.1 The message problem

Let us define a trivial problem for HMSCs, that consists in deciding whether a message of a chosen type is sent an received in some behavior of a specification given as a HMSC H.

Definition 24 *Let $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$ be a HMSC, m be a message type, and $p, q \in \mathcal{P}$ two processes of H. The* message problem *consists in deciding whether there exists a MSC $M \in \mathcal{F}_H$ that contains a sending of m by p and the corresponding reception on process q.*

A solution for this problem is trivial, and simply consists in exploring all MSCs in $\mathcal{M}$, and check if one of them contains a message m from p to q. This problem alone has little interest. However, one may notice that this problem is already undecidable for Communicating Finite State Machines [31]. This means that communicating finite state machines and HMSCs are distinct languages with a priori different expressive powers and distinct decidability results. Furthermore, we will show in chapter 9 that undecidability of the message problem implies undecidability of diagnosis in general.

3.2 Divergence

The divergence problem is a standard question in a distributed system. Processes composing a system can be of very different nature, and run on different machines. Consequently, some processes can be faster than others. In producer/consumer situations, where a group of processes $\mathcal{P}$ continuously sends messages to another group of processes $\mathcal{P}'$ and no acknowledgment is sent from $\mathcal{P}'$ to $\mathcal{P}$, if the processes in $\mathcal{P}'$ are slower than those in $\mathcal{P}$, then the communication buffers can be rapidly overflown. This situation can result in message losses or memory overflows, and should be avoided.

Definition 25 *A HMSC H is* divergent *if and only if there exists no universal bound on the contents of communication buffers in $\mathcal{L}(H)$.*

Theorem 6 *[23] A HMSC H is divergent if and only if for every simple loop ρ of H, the transitive closure of the communication graph CG_{M_ρ} is symmetric.*

Theorem 7 *[14] Checking that a HMSC is not divergent is a co-NP complete problem.*

A proof of this theorem can be found in appendix. Though we do not want to include all existing results on HMSCs, this proof is representative of standard proof techniques for co-NP completeness in partial order automata, and can be reused to prove that deciding if a HMSC belongs to the class of regular or globally cooperative HMSCs is also in Co-NP. Indeed, one can easily encode a SAT problem as connectivity question on the communication graph of some path in a HMSC H built from the SAT problem.

Note that non-divergent HMSCs need not have regular linearization languages, as shown by the example of Figure 1.8.

4 Syntactic subclasses of HMSCs

4.1 Regular HMSCs

The term *regular HMSC* is not very well chosen, as it lets reader think that an HMSC is a regular HMSC if and only if its linearization language is regular (which is undecidable in general, as shown by [33, 74]). We will show in this section that this term designates a decidable sub-class of HMSCs (i.e. it is decidable whether a HMSC belongs to this syntactic sub-class of the whole formalism), and not an undecidable property of linearization languages. Regular HMSC are also called *bounded HMSCs* [14], *loop connected HMSCs* or locally synchronized [113].

Definition 26 *Let H be a HMSC. H is a* regular HMSC *if and only if, for every cycle ρ of H, the communication graph of M_ρ is strongly connected.*

Theorem 8 *[14, 113] Let H be a regular HMSC. Then, $\mathcal{L}(H)$ is a regular language. Furthermore, $\mathcal{L}(H)$ is recognized by an automaton with at most $2^{(|\mathcal{P}|-1).m}.(m.n.|\mathcal{P}|)^{|\mathcal{P}|}$ states, where m is the number of transitions in H and n is the maximal number of events in a MSC of H.*

Theorem 9 *[14, 113] Checking if a HMSC is regular can be done in $O(m.n.2^|P|)$ and is a co-NP-complete problem.*

The Co-NP completeness proof uses the same argument as for divergence: A SAT problem can be encoded as a search for a cycle with a strongly connected communication graph.

The algorithm given in [14] is as follows:

Chose a partition of $\mathcal{P}$ into two sets P, Q of processes. Remove from H all transitions labeled by HMSCs with communications from a process in P to a process in Q. In the resulting graph, find all strongly connected components. If one of these components C involves processes from both P and Q, then any cycle of C has a communication graph that is not strongly connected.

One can alternatively chose a subset X of $\mathcal{M}$, check that the union of communication graphs of MSCs in X is not strongly connected. Then, restrict H to transitions labeled by a MSC in X, and check if the resulting graph contains cycles. This algorithm was initially proposed in [53], and yields a complexity in $O(m.n.2^{|\mathcal{M}|})$.

Of course, regular HMSCs enjoy all good properties of regular languages. For two regular HMSCs H_1, H_2 and a regular language R, one can decide if $\mathcal{F}_{H_1} \subseteq \mathcal{F}_{H_2}$,

$\mathcal{F}_{H_1} \cap \mathcal{F}_{H_2} = \emptyset$, $\mathcal{L}(H_1) \subseteq \mathcal{L}(H_2)$, $\mathcal{L}(H_1) \cap \mathcal{L}(H_2) = \emptyset$, $\mathcal{L}(H_1) \subseteq R$, $R \subseteq \mathcal{L}(H_2)$, $\mathcal{L}(H_1) \cap R = \emptyset$. Note that some HMSCs with regular linearization languages do not belong to the class of regular HMSCs. Consider for instance the HMSC of Figure 2.1. Its linearization language is $(a + b)^+$, which is a regular linearization language. However, from definition 26, it is not a regular HMSC. As deciding is a HMSC has a regular linearization language is undecidable, one can not either decide if there exists a way to transform a HMSC into an equivalent regular one, nor design an effective procedure to compute such transformation.

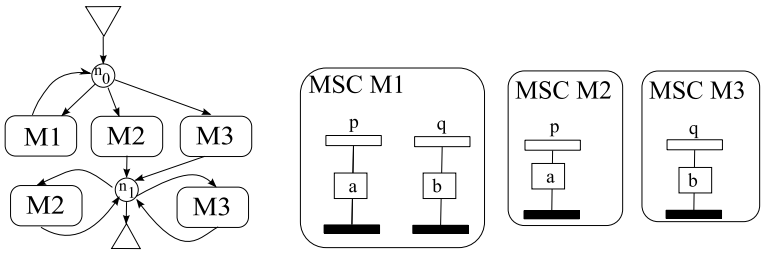


Figure 2.1: A HMSC with regular linearization language

4.2 Globally cooperative HMSCs

The class of regular HMSCs is quite restrictive: first of all it is limited to regular specifications only. Then, it is only a syntactic subclass of HMSCs, and does not capture the whole class of HMSC with a regular linearization language. A strictly greater and more interesting class of HMSC is the class of *globally cooperative* HMSCs, that was defined in [56, 109].

Definition 27 *A HMSC H is* globally cooperative *if and only if, for every cycle ρ of H, the communication graph of M_ρ is a connected graph.*

Unsurprisingly, verifying that a HMSC is globally cooperative is a $Co - NP$ complete problem. The proof (available in appendix) reuses the SAT encoding used for the divergence problem. The class of globally cooperative HMSCs enjoys some useful decidability results:

Definition 28 *Let H be a HMSC, and let $X \subseteq \mathcal{L}(H)$. We will say that X is a set of* representatives *for H if and only if for every M in $\mathcal{F}(H)$ we have $L(M) \cap X \neq \emptyset$.*

Theorem 10 *[55] Let H be a globally cooperative HMSC. Then there exists an integer b such that $\mathcal{L}^b(H)$ is a regular set of representatives.*

In fact, the result given by [55](Thm 4.1) is more general, and addresses any existentially bounded MSC language. Indeed is a set of MSCs $\mathcal{M}$ is existentially b-bounded, then it is equivalent for $\mathcal{M}$ to be the language of a CFM, to be the language of some globally cooperative HMSC, and to have a regular set of representatives $\mathcal{L}^b(\mathcal{M})$.

Theorem 11 *[56] Let H be a globally cooperative HMSC. Then, $\mathcal{L}^{at}(H)$ is recognized by a non-deterministic automaton of size at most $2^{O(n.k.p)}$, where k is the maximal size of some MSC in $\mathcal{M}$, n is the number of transitions in H, and p the number of processes.*

Corollary 1 *Let H_1,H_2 be two HMSC, and let H_2 be globally cooperative. Then checking whether $\mathcal{F}_{H_1} \subseteq \mathcal{F}_{H_2}$ and $\mathcal{F}_{H_1} \cap \mathcal{F}_{H_2} \neq \emptyset$ is decidable, and are respectively $EXPSPACE$ and $PSPACE$ complete.*

Similar results hold for checking $\mathcal{L}(H_1) \subseteq \mathcal{L}(H_2)$ and $\mathcal{L}(H_1) \cap \mathcal{L}(H_2) \neq \emptyset$. The complexity comes from the fact that checking $\mathcal{F}_{H_1} \subseteq \mathcal{F}_{H2}$ amounts to checking that $\mathcal{L}^{at}(H_1) \subseteq \mathcal{L}^{at}(H_2)$. Furthermore, it is sufficient to consider a set of representants X_1 such that $[X_1]_{||} = \mathcal{L}^{at}(H_1)$ to check inclusion, as $\mathcal{L}^{at}(H_2)$ is regular, and closed by commutation(from theorem 11, sequences of atoms of H_2 are recognized by a finite automaton). We can hence compute from H_1 a new HMSC H_1' that replaces each transition labeled by a MSC M from $\mathcal{M}_1$ by any sequences of transitions labeled by an atomic decomposition of M. We can similarly compute a HMSC H_2' labeled by atoms, and its commutative closure H_2''. It is then sufficient to check that $L(H_1') \subseteq L^{at}(H_2'')$. A similar property holds for intersection, where it suffices to check $L(H_1') \cap \mathcal{L}^{at}(H_2'')$. Both procedure yields the EXPSPACE and PSPACE complexities. The hardness part for inclusion comes easily, as one can easily encode the universality problem from loop-connected automata [113] with HMSCs inclusion. For the intersection, [56] encodes computations of polynomially space-bounded Turing Machines with HMSC inclusion.

The more important property of globally cooperative HMSCs is that their linearizations can be represented by a finite structure.

Theorem 12 *[61] Let H be a globally cooperative HMSC, and let b be the maximal existential bound obtained for a bMSC labeling some transition of H. Then, $\mathcal{L}^b(H)$ is a regular set of representatives for $\mathcal{L}(H)$, and can be recognized by an automaton of size $H^{b^2 \cdot |\mathcal{P}|^4 \cdot |H|}$.*

This property gives a way to effectively check whether intersection or inclusion of linearization languages of HMSCs.

Definition 29 *A HMSC is* locally-cooperative *iff, for every pair of transitions (n, M_1, n_1) and (n_1, M_2, n_2), the communication graphs of M_1, M_2, and $M_1 \circ M_2$ are weakly connected.*

Obviously, checking that a HMSC is locally-cooperative is linear in the size of the considered HMSC. Furthermore, locally-cooperative HMSCs admit smaller representations of atomic languages.

Theorem 13 *[56] Let H be a locally cooperative HMSC. Then, $\mathcal{L}^{at}(H)$ is recognized by a non-deterministic automaton of size $k.n.2^p.(p+1)^p$, where k is the maximal size of some MSC in $\mathcal{M}$, n is the number of transitions in H, and p the number of processes.*

The class of globally-cooperative and locally cooperative HMSCs are totally disjoint. Indeed, a HMSC without loops is necessarily globally cooperative, but can be non-locally cooperative. Similarly, one can find a HMSC with two consecutive transitions t_1, t_2 of a HMSC such that the concatenation of both has a disconnected communication graph, and such that t_2 is a connected loop and $t_1.t_2$ is not a cycle.

The subclass of globally cooperative HMSCs is tightly connected to the so-called recognizable HMSCs introduced by [109]. However, as we show below, global cooperation is a decidable syntactic criterion, while recognizability is an undecidable semantic notion.

Definition 30 *Let H be a HMSC over a set $\mathcal{M}$ of MSCs, and $\mathcal{L}$ be an MSC language. H is* recognizable *if and only if there exists a finite automaton over $Atoms(\mathcal{M})$ (i.e. a HMSC) that recognizes $\mathcal{F}_H$.*

Theorem 14 *[109] Let H be a HMSC. Then it is undecidable to know whether H is recognizable.*

The reason for this undecidability comes from the relation between MSC languages and traces. Let Γ be a finite set of atomic bMSCs, ant let $||$ be the independence relation among MSCs. Then, the relation that maps each trace $[a_1.\vdots.a_n] \in \mathbb{M}$ to the MSC $a_1 \circ \ldots a_n$ is an isomorphism. This relation between traces and HMSC sheds a new light on decidability problems for HMSCs, as recognizability, universality, etc. are notoriously undecidable problems for traces.

Theorem 15 *[109] Let $\mathcal{L}$ be a MSC language. Then $\mathcal{L}$ is recognizable if and only if there exists a globally cooperative HMSC H such that $\mathcal{F}_H = \mathcal{L}$.*

Let us comment on this theorem. It means that every globally cooperative has a recognizable language. This is not a surprising result, as one can always transform a HMSC H into another HMSC H^{at} labeled by atoms, by decomposing all MSCs in H into atomic sequences. Of course, nothing forces the atom language of an arbitrary HMSC H^{at} to be recognizable. Now, if H is globally cooperative, the atomic language of H^{at} falls into the class of star-connected expressions [43], which enjoys some nice decidability properties. In particular, one can compute an automaton $\mathcal{A}$ over $Atoms(H)$ that recognizes the language of atomic decomposition of MSCs in $\mathcal{F}_H$ [113]. From this latter work, we also know that the size of $\mathcal{A}$ is in $(n^2.2^a)^{n.a+n+1}$, where n is the size of H^{at} and a the size of $Atoms(H)$. Note also that global cooperation is a purely syntactic property. Consider for instance the two HMSCs of Figure 2.2. HMSC H_1 is not globally cooperative, as actions a and b in MSC $U2$ are independent atoms. However, one can easily show that this HMSC generates MSC with an arbitrary number of actions a on process p and b on process q. This can be easily represented by the HMSC $H2$ of Figure 2.2, which is globally cooperative. As recognizability of a HMSC is undecidable, this also means that there exists no generic algorithm to transform a recognizable non-globally cooperative HMSC into a globally cooperative one.

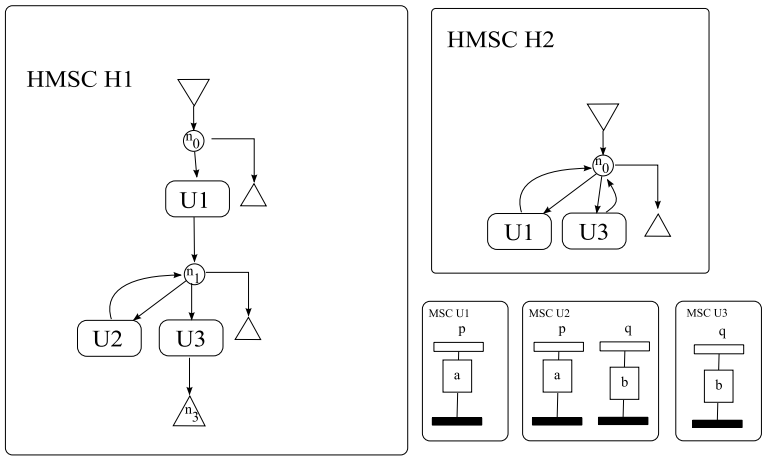


Figure 2.2: A non-globally cooperative HMSC H_1 and an equivalent globally cooperative HMSC H_2

4.3 Local-choice HMSCs

The last subclass of HMSCS addressed in this chapter is the class of Local-choice HMSCs. This class is interesting, as it allows for automated implementation techniques that transform HMSCs into equivalent finite state machines. This implementation problem will be addressed in more details in chapter 7 of this document. As already mentioned in chapter 1, alternatives in HMSCs should be handled with care, as two distinct branches of a choice may generate the same behavior. However, if two branches of a choice are different behaviors, one can interpret alternatives in HMSCs as alternatives in the control flow of a distributed program. At the description level addressed by HMSCs, a choice may involve several processes, that "agree" to perform one scenario rather that another. Consider for instance the HMSC of Figure 2.3. In this example, there is an alternative between a behavior in which process A can decide to send message m, and then this message is received by process B, and another one in which process B sends message n, which is received by process A. In each scenario, either A or B take a decision , and the other process should behave accordingly. In a distributed ans asynchronous context, enforcing these two behaviors **only** can be achieved by some election or consensus mechanism between A and B. Otherwise, careless implementation of this specification can lead to a deadlock, if A send message m and B sends message n simultaneously. This situation is called *non-local choice*, and was first identified in [23]. Locality of choices an HMSC is an interesting property : it helps ensuring implementability. It was also shown in [56] that locality of all choices of an HMSC has an impact on complexity and decidability of intersection and inclusion problems. We recall the definition of local choice HMSCs and complexity results hereafter.

Roughly speaking, a choice node in a HMSC is *local* if and only if one single process can decide which scenario will be followed by all the process participating to the specification after this choice. A MSC can have several minimal instances, called *deciding instances*. The set of deciding instances in a bMSC M is simply $\phi(Min(M))$. They carry the first events that happen in M, and execution one of these events can be interpreted as the decision to execute M. Obviously, these events

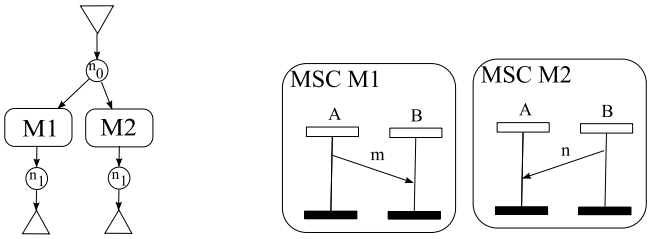


Figure 2.3: A non-local HMSC

cannot be message receptions.

The situation depicted in Figure 2.3 generalizes to choices with an arbitrary number of branches. There are several path starting from a choice node, and each path defines a MSC, with its deciding instances. According to the semantics of HMSCs, the deciding instances must choose to perform exactly the same behavior, and the other non-deciding instances have to conform to this choice. We can formalize local choices as follows:

Definition 31 (Local choice node) *Let $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$ be an HMSC, and let $c \in N$. Choice c is* local *if and only if for every pair of (non necessarily distinct) paths $\rho = c \xrightarrow{M_1} n_1 \xrightarrow{M_2} n_2 \dots n_k$ and $\rho' = c \xrightarrow{M'_1} n_1 \xrightarrow{M'_2} n'_2 \dots n'_k$ there is a single minimal instance in M_ρ and in $M_{\rho'}$ (i.e. $\phi(Min(M_\rho)) = \phi(Min(M'_\rho))$ and $|\phi(Min(M_\rho))| = 1$). H is called a* local-choice *HMSC if all its choices are local.*

Intuitively, the local-choice property [23] guarantees that every choice is controlled by a unique instance. Note however that this definition addresses pairs of path starting from a node, and chosen from an set of path that need not be finite. Fortunately, remarking that $\phi(min(M \circ M)) = \phi(min(M))$, one needs not consider twice the same transition. Hence checking locality of a choice can be brought back to properties from a finite set of paths.

Theorem 16 (Deciding locality) *Let H be an HMSC. H is* not local *iff there exists a node c and a pair of* ***acyclic*** *paths ρ, ρ' originating from c, such that M_ρ and $M_{\rho'}$ have more than one minimal instance.*

Corollary 2 (Complexity of local choice) *Deciding if an HMSC is local-choice is in $co - NP$.*

From theorem 16, an algorithm that checks locality of HMSCs is straightforward. It consists in a width first traversal of acyclic paths starting from each node of the HMSC. If at some time we find two paths with more than one minimal instance, then the choice from which these paths start is not local. Note that minimal instances need not be computed for the whole MSC labeling each path, and can be updated at the same time as paths. Indeed, if $\rho = \rho_1.\rho_2$ is a path of H, then $\phi(Min(M_\rho)) = \phi(Min(M_{\rho_1})) \cup (\phi(Min(M_{\rho_2})) \setminus \phi(M_{\rho_1}))$. It is then sufficient for each path to maintain the set of instances that appear along this path, and the set of minimal instances, without memorizing exactly the scenario played. As we consider only acyclic paths of the HMSC the following algorithm terminates.

The following algorithm was originally proposed in [79]. It builds a set of acyclic paths starting from each node of an HMSC. A non-local choice is detected if there is more than one deciding instance for a node c. The algorithm remembers a set of acyclic paths P, extends all of its members with new transitions when possible, and places a path ρ in MAP as soon as the set of transitions used in ρ contains a cycle.

Algorithm 1 LocalChoice(H)

for c node of H **do**
  P = $\{(t, I, J) \mid t = (c, M, n) \wedge I = \phi(min(M)) \wedge J = \phi(M)\}$
  MAP = $\emptyset$ ***/*Maximal acyclic Paths*/***
  while $P \neq \emptyset$ **do**
    MAP = MAP $\cup \{(w.t, I') \mid w = t_1...t_k \wedge t_k = (n_{k-1}, M_k, n_k), t = (n_k, M, n) \wedge t \in w \wedge (w, I, J) \in P \wedge I' = I \cup (\phi(min(M)) - J)\}$
    P = $\{(w.t, I', J') \mid (w, I, J) \in P, w = n_1...n_k \wedge t_k = (n_{k-1}, M_k, n_k), t = (n_k, M, n) \wedge t \notin w \wedge J' = J \cup \phi(M) \wedge I' = I \cup (\phi(min(M)) - J)\}$
  end while
  DI = $\bigcup_{(w,I) \in MAP}$ I ***/*Deciding Instances*/***
  if $\mid DI \mid > 1$ **then**
    H contains a non-local choice c
  end if
end for

The class of Local-choice HMSCs is particularly interesting, as only a few restrictions to this class allow to defines scenarios that can be implemented (i.e. translated into communicating finite state machines) by simply projecting the specification on each process. We will address the synthesis problem more in detail in chapter 7 of this document. Beyond the synthesis problem, local-choice HMSCs have interesting decidability and complexity issues.

Theorem 17 *[56] Let H_1, H_2 be two local-choice HMSCs, and let s_1, s_2 denote respectively the sum of the sizes of MSCs in $\mathcal{M}_1$ and $\mathcal{M}_2$. Deciding whether $\mathcal{L}(H_1) \subseteq \mathcal{L}(H_2)$ is $NLOGSPACE$-complete. This problem can be solved deterministically in time $O(p^2.(s_1 + s_2)^2)$. Deciding whether $\mathcal{L}(H_1) \cap \mathcal{L}(H_2) = \emptyset$ is $PSPACE$-complete in s_2.*

Note that the class of local choice HMSCs defined in [56] slightly differs from ours: it imposes that the first MSC appearing along a path starting after a choice node has a minimal event located on a process that is active in the MSC preceding the choice. It also imposes that every MSC generated by H starts with a single minimal event. There is also an implicit assumption that the HMSC is designed in such a way that all processes play a role between two choices. This assumption facilitates the implementation of local-choice HMSCs, as we shall see in chapter 7. The definition of [56] also allows for the transformation of local-choice HMSCs into equivalent HMSCs laying in a subclass of locally-cooperative HMSCs, which yields more efficient model checking results. Note however that the complexity results defined in [56] still hold for our definition of local-choice HMSCs, as one can always transform a local choice HMSCs H into a HMSC H' that meet the requirements

of [56] while preserving the results for the intersection and inclusion problems. This can be achieved by concatenating all MSCs between two choice nodes, and adding an initial "fake" transition labeled by an MSC in which an arbitrary chosen process sends a message to all processes appearing in $H_1, H2$. This fake transition does not affect the results nor the complexity.

5 Conclusion

We have seen in this chapter that even if many interesting problems (vacuity of intersection of two HMSC languages, inclusion, etc.) are undecidable for HMSCs in general, there are decidable syntactic subclasses for which these problems are decidable. An interesting fact that should be noticed is that these subclasses do not restrict to HMSCs with regular linearization languages. As indicated previously, classifying a HMSC as globally-cooperative, divergent, regular, ... can be done by analyzing the properties of its cycles. Hence, it is not surprising that all these classes are not disjoint. Indeed, any regular HMSC is also globally cooperative and non-divergent. However, a globally cooperative HMSC can be divergent or not (one can easily build a HMSC with a simple loop that iterates the sending of a message m from a process p to a process q, which is divergent, but globally cooperative). Local-cooperation is decided on pairs of transitions of the HMSC, and local-choice does not depend on cycles, so these properties are clearly orthogonal to cycle-based properties. It has also been shown [56] that a HMSC is regular if and only if it is globally cooperative, and has a universal bound on its channel contents, that is it also not divergent.

The Figure 2.4 below summarizes the relations among syntactic subclasses of HMSCs that were defined in this chapter. Unsurprisingly, regular and locally cooperative HMSCs are also globally cooperative. Note also that a locally-cooperative HMSC that is not regular is necessarily divergent, as there exists a cycle with a connected (but not strongly connected) communication graph in this HMSC, labeled by a MSC with at least two processes.

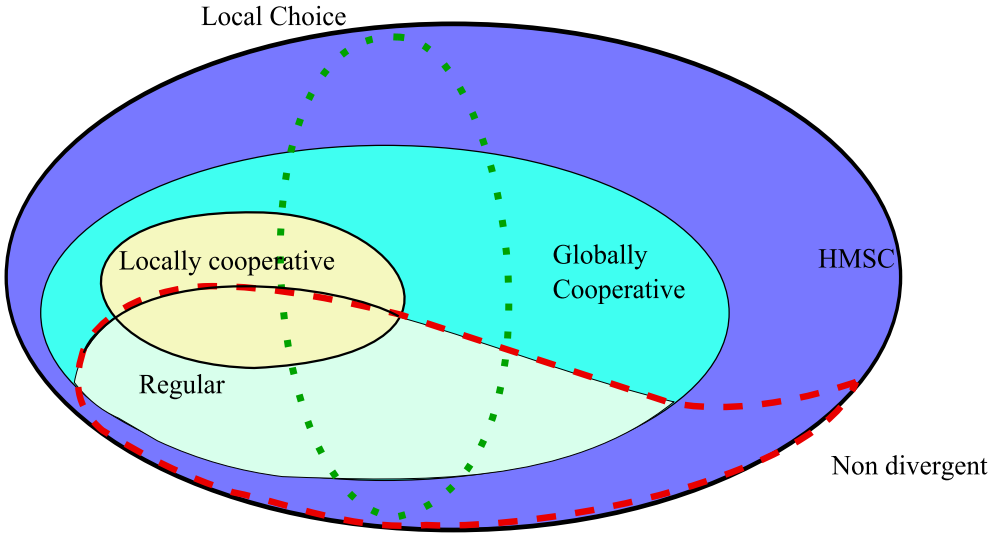


Figure 2.4: A classification of HMSC subclasses

Chapter 3

An extension of partial order automata: splitting messages

Toute tentative en vue de diviser quoi que se soit par deux devrait, a priori, nous inspirer une extrême méfiance.
Attempts to divide anything into two ought to be regarded with much suspicion.
[Charles Percy Snow, The two cultures]

1 Introduction

In the first two chapters of this document, we have shown that partial order automata are a powerful modeling formalism: they can encode traces, rational relations,... Unsurprisingly in this context, many usual problems that are decidable on automata become undecidable for partial order automata. We have seen that reasonable and decidable syntactic subclasses allow for the decision of interesting problems, and that the expressive power of these subclasses go beyond regular languages. One may think that using models outside of these classes resumes to trading expressive power for decidability. However, even in their full generality, HMSCs can not model very simple protocols that are frequently used in distributed systems, the so-called sliding windows protocols.

A sliding window protocol between two processes can be seen as a sequence of questions/answers: a process p asks a question to another process q, which returns his answer. In order to speed up the exchanges between p and q when there is more than one question, process p can send some questions in advance, without waiting for $q's$ answer. To avoid losses, and also to ensure correctness of a sequence of answers, messages exchanged between p and q are labeled by identifiers. Each process sends to his partner his message tagged by such identity, plus the identity of the last message received in a lossless sequence. With this tagging mechanism, a process can detect that one of his messages was lost, and retransmit it, or get a confirmation of receptions, and then decide to send more messages in advance. The number of messages sent in advance is called the "'size of the window". Protocols such as TCP-IP use this kind of mechanism.

One important fact to notice for executions of this kind of protocol is that some executions may have the form of braids. Furthermore, these braids consist of atoms of arbitrary sizes. Such sets of executions of arbitrary size can not be represented

as concatenations from a finite set of atomic MSCs. Consider for instance the two MSCs of Figure 3.1. This figure describes exchanges of questions/answers between a process p and a process q. In MSC M, the answer to a question arrives before the sending of the next question. Clearly, such execution with an arbitrary number of exchanges n can be modeled as n concatenations of a MSC containing a message from p to q followed by a message from q to p. Now, let us consider the MSC N in the same figure. In a systematic way after the second question, the reception of question n occurs just after the sending of answer $n-1$, and the reception of answer n occurs just after the sending of question $n+1$. Hence, the last $2n-1$ messages of this kind of execution form an atom. A MSC language containing such behaviors can not be expressed as a concatenation of finite atoms of bounded sizes, and hence is not the set of MSCs generated by some HMSC.

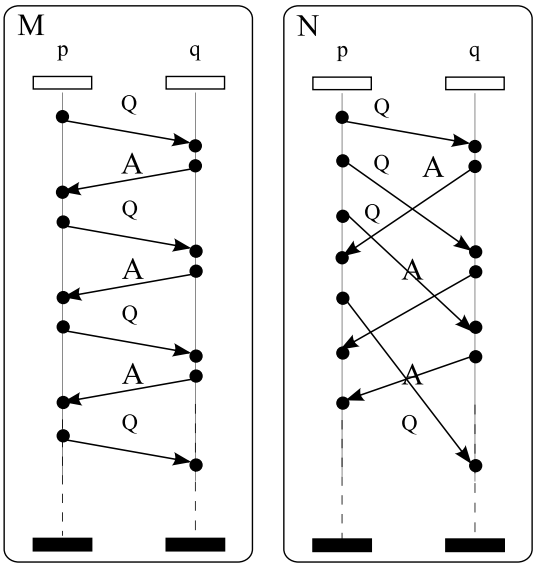


Figure 3.1: Finitely generated MSC language (M) and non-finitely generated MSC language (N)

Definition 32 *Let $\mathcal{F}$ be a (possibly infinite) set of MSCs. We will say that $\mathcal{F}$ is* finitely generated *if and only if there exists a* **finite** *set of MSCs $X = \{M_1, \ldots, M_k\}$ such that every MSC M of $\mathcal{F}$ can be expressed as a concatenation of a finite number of MSCs chosen from X.*

One can immediately notice that the MSC languages designed using HMSCs are finitely generated. One can also notice that an MSC language comporting all MSCs of the form of MSC N in Figure 3.1 (i.e. with an arbitrary number of questions/answers) is not finitely generated, and hence can not be described by a HMSC. This is clearly a limitation of HMSCs, as many protocols use sliding windows. A question that immediately arises is whether some extension to HMSCs can allow for the modeling of sliding windows. Another question is the preservation of decidability/complexity results for such extension. In this chapter, we describe a first proposal, namely compositional MSCs, that consists in allowing incomplete communications in MSCs. In the next chapter, we propose another extension that does

not split communications. The role of this chapter is mainly to define the different subclasses of compositional MSCs that will be needed in the next chapter. For a complete study of compositional MSCs, we refer interested readers to [58].

2 Compositional HMSC

A solution originally proposed by [65] and studied by B. Genest [58] is to authorize incomplete communications in MSCs, and to complete messages during MSC concatenation. The obtained model is called *Compositional MSCs*.

Definition 33 *A* compositional MSC *(or CMSC for short) is a tuple* $C = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \phi)$ *where* $E = E_S \cup E_R \cup E_a$ *is a set of events, each* $<_p$ *is a total ordering on events located on process* p, $\phi : E \longrightarrow \mathcal{P}$ *associates a locality to each event, and* $\mu : E_S \longrightarrow$ *is a* ***partial function*** *from sending events to receiving events, such that if* $f = \mu(e)$, *then* $\alpha(e)$ *is of the form* $p!q(m)$ *and* $\alpha(f)$ *is of the form* $q?p(m)$.

As for MSCs, we require that the message mapping satisfies the FIFO assumption. Similarly, we define the causal order in a CMSC as the closure $\leq = (\bigcup <_p \cup \mu)^*$, and require $\leq$ to be a partial order. We can immediately notice that MSCs are a subclass of CMSCs for which μ is a *total function* from E_S to E_R. When μ is a total function in a CMSC C, we will say that C is *communication-closed*. However, nothing forces CMSCs to have a bijective message mapping, nor to comport an equal number of sending and receiving events in the general case. Sendings that are not associated to a reception $(E_S \setminus \mu^{-1}(E_R))$ and conversely receptions that are not associated with a sending event $(E_R \setminus \mu(E_S))$ are called incomplete messages. As MSCs, CMSCs can be composed to obtain larger specifications. During concatenation, incomplete messages are used to create complete communications.

Definition 34 *Let* $C_1 = (E_1, <_{1,p}, \alpha_1, \mu_1, \phi_1)$ *and* $C_2 = (E_2, <_{2,p}, \alpha_2, \mu_2, \phi_2)$ *be two CMSCs (defined over disjoint sets of events). A sequential composition of* C_1 *and* C_2 *is a CMSC* $C = (E_1 \cup E_2, (<_p), \alpha_1 \cup \alpha_2, \mu, \phi_1 \cup \phi_2)$, *where : The ordering* $<_p$ *on each process* $p \in \mathcal{P}$ *is defined as:*

$$<_p = (<_{1,p} \cup <_{2,p} \cup \{(e, f) \in E_1 \times E_2 \mid \phi(e) = \phi(f) = p\})$$

The message mapping μ *is defined as:*

$$\mu(e,f) = \begin{cases} \mu_1(e,f) \text{ if } e,f \in E_1, \\ \mu_2(e,f) \text{ if } e,f \in E_2 \\ \text{is an element of } \begin{array}{l}(E_1 \cup E_2) \setminus (\mu_1^{-1}(E_1) \cup \mu_2^{-1}(E_2)) \\ \times (E_1 \cup E_2) \setminus (\mu_1(E_1) \cup \mu_2(E_2))\end{array} \\ \text{or is undefined otherwise} \end{cases}$$

Of course, as $C = C_1 \circ C_2$ is a CMSC it must satisfy the conditions on consistency between message mapping and labeling, meet the FIFO assumption, and guarantee that $\leq$ is a partial order (that is newly created message mappings should not create cycles in the causal dependency relation). Note also that a message sent in C_2 can be received in C_1. The sequential concatenation of two CMSCs may result in several

CMSCs, as the message mapping need not be total. However, it was proved in [58] that for a given sequence of CMSCs $C_1.C_2 \ldots C_k$, there exists **at most** one MSC in $C_1 \circ C_2 \circ \cdots \circ C_k$.

The notion of *communication graph* defined in chapter 1 needs to be adapted for CMSCs. For a given CMSC $C = (E, \leq, \alpha, \mu, \phi)$, the communication graph $CG(C) = (\mathcal{P}, V)$ is a graph such that $p \in \mathcal{P}$ if and only if there exists an event $e \in E$ such that $\alpha(e) = p!q(m)$ or $\alpha(e) = q?p(m)$ for some q, m. Similarly, $(p, q) \in V$ if and only if there exists an event $e \in E$ such that $\alpha(e) = p!q(m)$ for some m.

Definition 35 *A* compositional HMSC *(or CHMSC for short) is a HMSC which transitions are labeled by compositional MSCs. The partial order family $\mathcal{F}_H$ generated by a CHMSC is the set of* **MSCs** *obtained by concatenation of CMSCS along accepting paths of H. The linearization language of a CHMSC is the union of linearizations of MSCs in $\mathcal{F}_H$.*

With this definition of CHMSCs, we can define a CHMSC H such that $\mathcal{F}_H = \emptyset$. Consider for instance a HMSC $H_\emptyset = (\{n_0, n_1\}, \longrightarrow, \{C_1\}, n_0, \{n_1\})$ with a single transition (n_0, C_1, n_1), and a single CMSC C_1 depicted in Figure 3.2. Clearly, as C_1 is not communication-closed, there is no path of $H_\emptyset$ generating a complete MSC, and $\mathcal{F}_{H_\emptyset} = \emptyset$. This phenomenon can occur as soon as some message sending can not be properly mapped to a reception (or the converse) in any path of the CHMSC. One can easily see that CHMSCs are strictly more expressive than HMSCs (HMSCs are CHMSCs which labels are CMSCs with total message mappings). However, this gain in expressive power has a cost: some problems that can be easily decided on HMSCs become undecidable for CHMSCs.

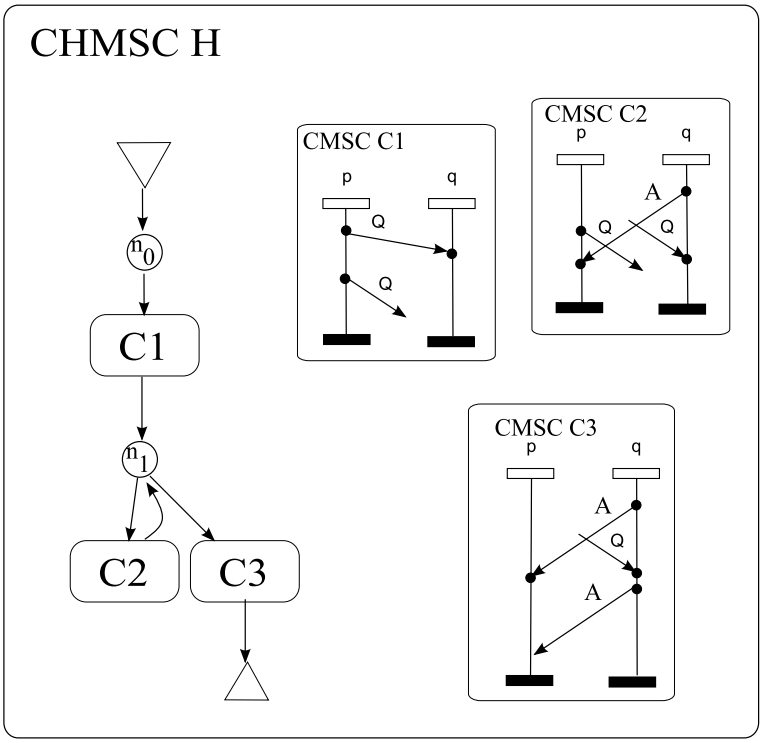


Figure 3.2: A example CHMSC that generates the infinite set of braids of Figure 3.1

Theorem 18 *[58] CHMSCs are strictly more expressive than communicating finite state machines and HMSCs.*

Transforming a set of communicating finite state machines $\mathcal{A}_1, \ldots \mathcal{A}_k$ into a CHMSC is straightforwards. Each $\mathcal{A}_i$ can be transformed into a CHMSC H_i with the same number of transitions as $\mathcal{A}_i$, labeled by CMSCs that contain only one event, located on process i, and labeled by the same action name as in the corresponding automaton transition. Then, serializing all $H_i's$ in any order produces a CHMSC with the same language as $\mathcal{A}_1, \ldots \mathcal{A}_k$.

Theorem 19 *Let H be a compositional HMSC. The following problems are undecidable:*

- *Check if $\mathcal{F}_H = \emptyset$?*
- *Is there a MSC M in $\mathcal{F}_H$ such that a given message m is sent and received in M ? (message problem, see chapter 2)*
- *for a given $b \in \mathbb{N}$, is $\mathcal{F}_H$ existentially $b-$bounded ?*

This theorem can be proved easily, as emptiness, existential boundedness or the message problem are already undecidable for communicating finite state machines [31,32] (one can easily encode a PCP with any of these problems). However, as usual for scenario languages, several syntactic restrictions to the model helps avoiding this undecidability, while preserving an interesting expressive power.

As for HMSCs, subclasses of CHMSCs can be found by considering properties of it cycles. This requires a light adaptation of the definition of communication graph introduced in chapter 1, as in CMSCs, communications can be split. The *communication graph* of a path ρ is a graph in which vertices are processes appearing in ρ and vertices pairs of processes $(p, q^)$ such that there exist an action labeled $p!q(m)$ and and action labeled $q?p(n)$ for some m, n.

Definition 36 *Let H be a compositional HMSC. We will say that H is* safe *if and only if for every accepting path ρ of H, the concatenation of CMSCs along ρ contains a MSC. We will say that H is* well-balanced *if and only if for every cycle ρ of H, and every pair of processes p, q, the number of messages sent from p to q is equal to the number of messages received by q from p in ρ. We will say that H is* globally cooperative *if and only if it is safe, and the communication graph of every loop is* connected. *We will say that H is* regular *if and only if it is safe, and the communication graph of every loop is* strongly connected.

As for HMSCs, detecting global cooperation and regularity are co-NP complete problems. The partial order families generated by safe CHMSCs and well-balanced CHMSCs are equivalent [58]. Safe CHMSCs are existentially bounded, i.e. they generate only existentially b-bounded MSCs for some b. For a given safe CHMSC G we will denote this existential bound by b_G. Furthermore, deciding if a CHMSC is safe or well-balanced is decidable in polynomial time [65]. Note that globally cooperative or regular CHMSCs are again defined by syntactic criteria. Furthermore, the globally cooperative or regular subclasses of HMSCs are strictly included in their CHMSC counterpart.

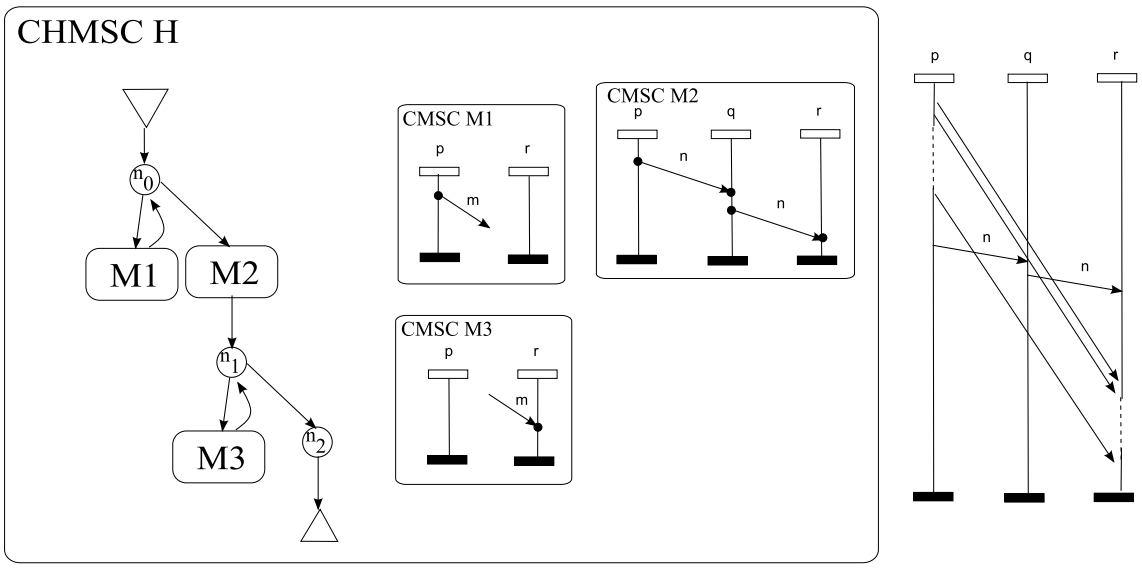


Figure 3.3: An unsafe CHMSC and the partial order family it generates

Of course, being safe is not sufficient to ensure decidability of model-checking problems, as HMSCs are guaranteed to be safe CHMSCs, and one has to rely on the globally cooperative subclass to provide decision procedure for problems on safe CHMSCs.

Theorem 20 *[55, 58] Let G be a safe CHMSC, and let H be a globally cooperative CHMSC. Then $\mathcal{L}(G) \cap \mathcal{L}(H) = \emptyset$ is decidable and is PSPACE-complete in $|H|, b_G$ $\mathcal{L}(G) \subseteq \mathcal{L}(H)$ is decidable, and is EXPSPACE-complete in $|H|, b_G$, where bG is the existential bound for G.*

The decision procedure relies on the fact that $\mathcal{L}^{b_H}(H)$ is regular, and that one can associate a b_G-bounded linearization to every path of G, that is one can compute a regular language L of b_G-bounded representative linearizations for G (even if $\mathcal{L}^{b_G}(G)$ is not regular), and compare it to the regular $\mathcal{L}^{b_H}(H)$. Note that the difference between bounds b_G and b_H is not a problem, as for every $b > b_H$, H is also existentially b-bounded.

3 Conclusion

In this chapter, we have described a way to increase the expressive power of HMSCs by splitting messages, and recomposing them at concatenation time. In particular, this allows for the definition of partial order families that are not finitely generated. We will not give more details on this model, and refer interested reader to [58] for a more complete study of its properties. A drawback of compositional HMSCs is that composing orders that are not communication closed means embedding the expressive power of communicating automata and all their undecidability results. This problem is circumvented by defining a subclass of *safe, globally cooperative and regular* CHMSCs. In the next chapter, we will propose an alternative to CHMSCs, that allows for the definition of non-finitely generated partial order families without splitting messages.

Chapter 4

Extension of partial order automata with commutations

Le désordre est le meilleur serviteur de l'ordre établi.
Disorder is the best servant of established order.
[Jean-Paul Sartre]

1 Introduction

In chapter 3, we have shown that partial order automata were not expressive enough to describe the so-called sliding windows protocols, and we have described an extension called Compositional HMSCs, that allows composing finite pieces of behaviors that are not communication-closed. This extension allows for the modeling of partial order families that are not finitely generated. A drawback of this formalism is that it embeds the expressive power of communicating finite state machines [32] and consequently inherits all their undecidability results. A more decidable class of CHMSCs called safe CHMSCs has been proposed.

Another solution consists in preserving the communication closed characteristics of MSCs, compose them with an order automaton as usual, but modify the concatenation operation in such a way that the respective ordering of events on a process can be rearranged according to a commutation relation. This chapter describes this extension and its properties.

We first extend the notion of an MSC to a *causal* MSC in which the events belonging to each lifeline (process), instead of being linearly ordered, are allowed to be partially ordered. To gain modeling power, we do not impose any serious restriction on the nature of this partial order. However, we assume a suitable Mazurkiewicz trace alphabet [43] for each lifeline and use this to define a composition operation for causal MSCs. This leads to the notion of causal HMSCs.

The *existential boundedness* property in HMSCs or safe CHMSCs leads to proof techniques using regular representative sets of linearizations for establishing decidability results. As we will show later in this chapter, a causal HMSC (i.e. the MSC language associated with a causal HMSC) is *a priori* not existentially bounded. Hence the proof techniques relying on the computation of a finite state automaton recognizing a language of b-bounded representatives can not be used to obtain de-

cidability results. Instead, we need to generalize the methods of [113] and of [56] in a non-trivial way.

The first major result presented in this chapter is to formulate natural -and decidable- structural conditions which ensure that a causal HMSC generates a regular MSC languages. The second major result is that the inclusion problem for causal HMSCs is decidable for causal HMSCs using the same Mazurkiewicz trace alphabets, provided at least one of them is *globally-cooperative* (the definition of global cooperation is adapted to the context of causal HMSCs). Furthermore, we prove that the restriction that the two causal HMSCs have identical Mazurkiewicz trace alphabets associated with them is necessary. These results constitute a non-trivial extension for causal HMSCs of comparable results on HMSCs [14, 76, 113] and [56].

The last result presented is this chapter is the identification of a property called "window-bounded" which appears to be an important ingredient of the "braid"-like MSC languages generated by many protocols. Intuitively, this property bounds the number of messages a process p can send to a process q before having received an acknowledgment to the earliest message. This is a desirable property of communication protocols that is usually enforced by counters in their implementations. We show that we can decide if a causal HMSC generates a window-bounded MSC language. Finally, we compare the expressive power of languages based on causal HMSCs with HMSCs and CHMSCs, and prove that causal HMSCs and compositional HMSCs are rapidly incomparable.

This chapter is organized as follows: in the next section we introduce causal MSCs and causal HMSCs. We also define the means for associating an ordinary MSC language with a causal HMSC. In the subsequent section we develop the basic theory of causal HMSCs. To this end, we identify the subclasses of regular (syntactically regular) and globally-cooperative causal HMSCs and develop our decidability results. In section 3.3, we identify the "window-bounded" property, and show that one can decide if a causal HMSC generates a window-bounded MSC language. In section 4 we compare the expressive power of languages based on causal HMSCs with other known HMSC-based language classes.

This work was performed jointly with P.S. Thiagarajan, B. Genest, T. Gazagnaire, and S. Yang within the context of the CASDS and DST associated teams, and during T. Gazagnaire's PhD [51]. It led to two publications [53, 54].

2 causal MSCs

For this chapter, we fix a finite nonempty set $\mathcal{P}$ of process names with $|\mathcal{P}| > 1$. For convenience, we let p, q range over $\mathcal{P}$ and drop any subscript $p \in \mathcal{P}$ when there is no confusion. We also fix finite nonempty sets Msg, Act of message types and internal action names respectively. We define the alphabets $\Sigma_! = \{p!q(m) \mid p, q \in \mathcal{P}, p \neq q, m \in Msg\}$, $\Sigma_? = \{p?q(m) \mid p, q \in \mathcal{P}, p \neq q, m \in Msg\}$, and $\Sigma_{act} = \{p(a) \mid p \in \mathcal{P}, a \in Act\}$. The causal MSCs introduced hereafter are defined over $\mathcal{P}$ with alphabet $\Sigma = \Sigma_! \cup \Sigma_? \cup \Sigma_{act}$. We define the *location* of a letter a in Σ, denoted $loc(a)$, by $loc(p!q(m)) = p = loc(p?q(m)) = loc(p(a))$. For each process p in $\mathcal{P}$, we set $\Sigma_p = \{a \in \Sigma \mid loc(a) = p\}$.

Definition 37 *A* causal MSC *over* $(\mathcal{P}, \Sigma)$ *is a structure* $B = (E, \{\sqsubseteq_p\}, \alpha, \mu, \phi)$, *where* E *is a finite nonempty set of* events, $\alpha : E \to \Sigma$ *is a labeling function,* $\phi : E \to \mathcal{P}$ *assigns a process to each event, and the following conditions hold:*

- *For each process* p, $\sqsubseteq_p \subseteq E_p \times E_p$ *is a* partial order, *where* $E_p = \{e \in E \mid \alpha(e) \in \Sigma_p\}$. *We let* $\widehat{\sqsubseteq}_p \subseteq E_p \times E_p$ *denote the least relation such that* $\sqsubseteq_p$ *is the reflexive and transitive closure of* $\widehat{\sqsubseteq}_p$ *(*$\widehat{\sqsubseteq}_p$ *is the Hasse diagram of* $\sqsubseteq_p$*).*
- $\mu \subseteq E_! \times E_?$ *is a bijection, where* $E_! = \{e \in E \mid \alpha(e) \in \Sigma_!\}$ *and* $E_? = \{e \in E \mid \alpha(e) \in \Sigma_?\}$. *Furthermore, labeling is consistent with messages, that is for each* $(e, e') \in \mu$, $\alpha(e) = p!q(m)$ *iff* $\alpha(e') = q?p(m)$.
- *The transitive closure* $\leq$ *of the relation* $\big(\bigcup_{p \in \mathcal{P}} \sqsubseteq_p\big) \cup \mu$ *is a partial order.*
- $loc(\alpha(e)) = \phi(e)$ *(the locality of an event is consistent with its labeling).*

For each p, the relation $\sqsubseteq_p$ dictates the "causal" order in which events of E_p may be executed. As for MSCs, the mapping $\mu : E_S \mapsto E_R$ identifies pairs of message-emission and message-reception events. We also denote by $|B|$ the size of B, that is the number of events in B.

We will say that a causal MSC $B = (E, \{\sqsubseteq_p\}, \alpha, \mu, \phi)$ is *weak-FIFO* iff for any $f = \mu(e)$, $f' = \mu(e')$ such that $\alpha(e) = \alpha(e') = p!q(m')$ (and thus $\alpha(f) = \alpha(f') = q?p(m)$), we have either $e \sqsubseteq_p e'$ and $f \sqsubseteq_q f'$; or $e' \sqsubseteq_p e$ and $f' \sqsubseteq_q f$. Note that we do not demand *a priori* that a causal MSC must be weak-FIFO. Testing (weak) fifoness of a causal MSC of size b can be done in at most $\mathcal{O}(\frac{b^2}{8} - \frac{b}{4})$, by considering all pairs of messages in the MSC.

A *linearization* of B is a word $a_1 a_2 \dots a_\ell$ over Σ such that $E = \{e_1, \dots, e_\ell\}$ with $\alpha(e_i) = a_i$ for each i; and $e_i \leq e_j$ implies $i \leq j$ for any i, j. We let $Lin(B)$ denote the set of linearizations of B. Clearly, $Lin(B)$ is nonempty. We set $Alph(B) = \{\alpha(e) \mid e \in E\}$, and $Alph_p(B) = Alph(B) \cap \Sigma_p$ for each p.

The leftmost part of Figure 4.1 depicts a causal MSC M. In this diagram, we enclose events of each process p in a vertical box and show the partial order $\sqsubseteq_p$ in the standard way, that is using its Hasse Diagram. In case $\sqsubseteq_p$ is a total order, we place events of p along a vertical line with the minimum events at the top and omit the box. In particular, in M, the two events on p are not ordered (i.e. $\widehat{\sqsubseteq}_p$ is empty) and $\sqsubseteq_q$ is a total order. Members of μ are indicated by horizontal or downward-sloping arrows labeled with the transmitted message. Both words $p!q(Q).q!p(A).q?p(Q).p?q(A)$ and $q!p(A).p?q(A).p!q(Q).q?p(Q)$ are linearizations of the causal MSC M in Figure 4.1.

Obviously, a causal MSC $B = (E, \{\sqsubseteq_p\}, \alpha, \mu, \phi)$ is an MSC if every $\sqsubseteq_p$ is a *total order*. In an MSC B, the relation $\sqsubseteq_p$ must be interpreted as the visually observed order of events in one sequential execution of p. Let $B' = (E', \{\sqsubseteq'_p\}, \alpha', \mu', \phi')$ be a causal MSC. Then we say the MSC B is a *visual extension* of B' if $E' = E$, $\alpha' = \alpha$, $\sqsubseteq'_p \subseteq \sqsubseteq_p$, $\mu' = \mu$ and $\phi' = \phi$. We let $Vis(B')$ denote the set of visual extensions of B'. Note that as visual extensions are MSCs, for every process p, $\sqsubseteq_p$ is a total order.

In Figure 4.1, $Vis(M)$ consists of MSCs $M1, M2$. The idea of visual ordering is inspired by the difference between visual and causal orders originally highlighted by [13], and discussed in chapter 2.2.4. In causal MSCs, instead of defining a total

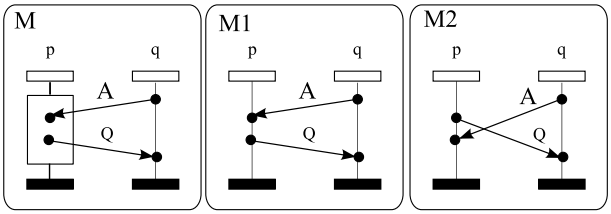


Figure 4.1: A causal MSC M and its visual extensions $M1, M2$.

ordering along an process lifeline, and then interpreting a weaker version of this ordering, we define directly an unambiguous interpretation of events ordering along a process. Lack of ordering among events of the same process can be interpreted as concurrency if a process is seen as a high level description of an entity with some kind of parallelism, or as a lack of information on the relative ordering between events, which can be explicitly set as any of the visual extensions.

Note that the set of visual extensions of a causal MSC B is not necessarily the union of instance per instance linearizations, as an extension of a causal MSC must remain a MSC, i.e., the relation among the events has to remain a partial order. Consider for example, the causal MSC of Figure 4.2, and its visual extensions in Figure 4.3: we cannot have $e_2 \sqsubseteq_p e_1$ and $f_1 \sqsubseteq_q f_2$ at the same time in a visual extension of this causal MSC.

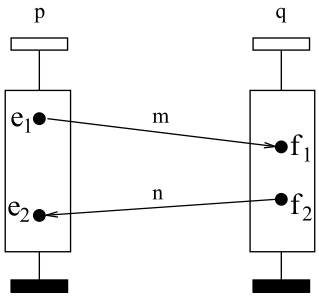


Figure 4.2: An example of causal MSC B: the set of visual extensions of B is not the instance per instance commutative closure of any visual extension of B.

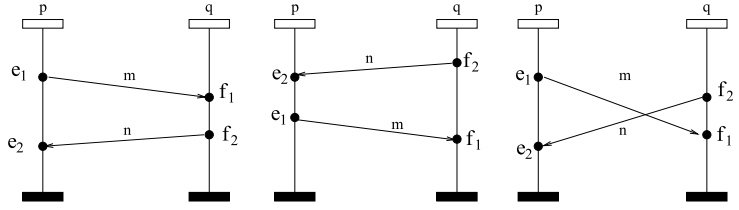


Figure 4.3: Visual extensions of the causal MSC B of Figure 4.2.

2.1 Concatenation of causal MSCs

So far, causal MSCs alone are not more expressive than finite collections of MSCs, as one can express the same set of behaviors using a causal MSC B, or with a finite

set of visual extensions (i.e. MSCs) $Vis(B)$. The whole expressive power of causal MSCs appears when assembling sequences of causal MSCs.

In standard MSCs, concatenation imposes a total ordering on events located on the same process. To keep the spirit of causal MSCs, concatenation of two causal MSCs M_1 and M_2 should not impose such ordering. Indeed, in such a case, a causal HMSC (that we will define later in this chapter) would not be more expressive than a HMSCs in which every causal MSC is replaced by an alternative between all its visual extensions. We hence propose a concatenation that preserves some independence between events of the two assembled diagrams. To express whether there should be a dependency or not, for each process p in $\mathcal{P}$, we fix a concurrent alphabet (Mazurkiewicz trace alphabet [43]) (Σ_p, I_p) for each process $p \in \mathcal{P}$, where $I_p \subseteq \Sigma_p \times \Sigma_p$ is a symmetric and irreflexive relation called the *independence relation* over the alphabet of actions Σ_p. We denote the *dependence relation* $(\Sigma_p \times \Sigma_p) - I_p$ by D_p.

Following the usual definitions of Mazurkiewicz traces [43], for each (Σ_p, I_p), the associated trace equivalence relation $\sim_p$ over $\Sigma_p^\star$ is the least equivalence relation such that, for any u, v in $\Sigma_p^\star$ and a, b in Σ_p, $a\ I_p\ b$ implies $u.a.b.v \sim_p u.b.a.v$. Equivalence classes of $\sim_p$ are called *traces*. For u in $\Sigma_p^\star$, we let $[u]_p$ denote the trace containing u.

Note that the definition of causal MSCs does not take into account this notion of independence, as the ordering among events is not defined according to their labels. However, we can define some consistency between a causal MSC and the independence. Let $B = (E, \{\sqsubseteq_p\}, \alpha, \mu, \phi)$ be a causal MSC. We say $\sqsubseteq_p$ *respects* the trace alphabet (Σ_p, I_p) iff for any $e, e' \in E_p$, the following hold:

(i) $\alpha(e)\ D_p\ \alpha(e')$ implies $e\ \sqsubseteq_p e'$ or $e'\ \sqsubseteq_p e$

(ii) $e\ \widehat{\sqsubseteq}_p\ e'$ implies $\alpha(e)\ D_p\ \alpha(e')$

A causal MSC B is said to respect the trace alphabets $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$ iff $\sqsubseteq_p$ respects (Σ_p, I_p) for every p. In order to gain modeling power, we have allowed each $\sqsubseteq_p$ to be *any* partial order, not necessarily respecting (Σ_p, I_p). However, as we will show later, the fact that causal MSCs respect or not independence relations has an influence on the relations between causal orders, visual orders, and linearizations generated by a high-level graph. We can now define the concatenation operation of causal MSCs using the trace alphabets $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$.

Definition 38 *Let $B = (E, \{\sqsubseteq_p\}, \alpha, \mu, \phi)$ and $B' = (E', \{\sqsubseteq'_p\}, \alpha, \mu', \phi')$ be causal MSCs. We define the* concatenation *of B with B', denoted by $B \circledcirc B'$, as the causal MSC $B'' = (E'', \{\sqsubseteq''_p\}, \alpha'', \mu'', , \phi'')$ where:*

- *E'' is the disjoint union of E and E'. α'' is given by: $\alpha''(e) = \alpha(e)$ if $e \in E$, $\alpha''(e) = \alpha'(e)$ if $e \in E$ and $\mu'' = \mu \cup \mu'$.*
- *For each p, $\sqsubseteq''_p$ is the transitive closure of*

$$\sqsubseteq_p \bigcup \sqsubseteq'_p \bigcup \{(e, e') \in E_p \times E'_p \mid \alpha(e)\ D_p\ \alpha'(e')\}$$

Clearly, $\circledcirc$ is a well-defined and associative operation. Note that in case B and B' are MSCs and $D_p = \Sigma_p \times \Sigma_p$ for every p, then the result of $B \circledcirc B'$ is the sequential composition of B with B' seen in definition 6 in chapter 1. Note that when B and B' are weak FIFO causal MSCs, then their concatenation is also weak FIFO. This property comes from the irreflexive nature of the independence relations. This remark also holds for the concatenation of MSCs. This remark does not hold for strong FIFOness, as two messages of different kind may overtake in a visual extension of $B \circledcirc B'$, even when B and B' are FIFO. We also remark that the concatenation of causal MSCs differs from the concatenation of traces. The concatenation of trace $[u]_p$ with $[v]_p$ is the trace $[uv]_p$, that is uv is closed by commutation, which allows to switch ordering among letters in u or v. However, a causal MSC B needs not respect $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$. Consequently, for a process p, the projection of $Lin(B)$ on $Alph_p(B)$ is *not* necessarily a trace.

Figure 4.4 shows an example of sequential composition of two causal MSCs B_1 and B_2, with the dependency relations D_p and D_q being the commutative and reflexive closure of $\{(p!q(m), p!q(n)), (p!q(n), p?q(u))\}$ and $\{(q?p(n), q!p(v))\}$ respectively. Note that although the dependence relation D_p contains the pair $(p!q(m), p!q(n))$, sendings of messages m and n by process p in B_1 are unordered (B_1 does not respect (Σ_p, I_p)), and remain unordered after composition.

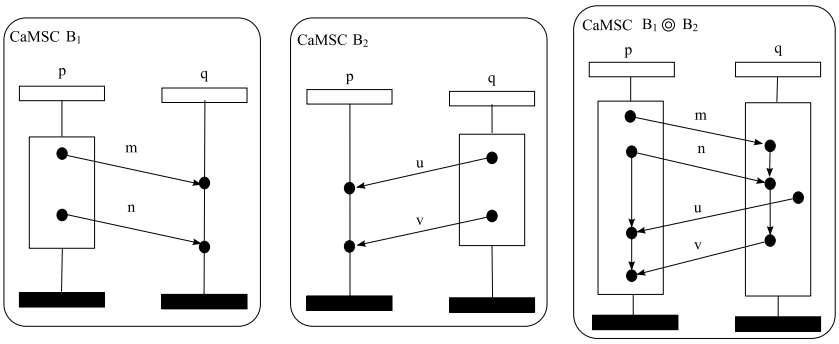


Figure 4.4: Concatenation example.

2.2 Causal HMSCs

As for MSCs and compositional MSCs, we can extend the concatenation mechanism using an automaton labeled by causal MSCs, to produce scenario languages. These automata will be called *causal HMSCs*. Let us fix a set $\mathcal{P}$ of process names and a family $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$ of Mazurkiewicz trace alphabets.

Definition 39 *A* causal HMSC *over* $(\mathcal{P}, \{(\Sigma_p, I_p)\}_{p\in\mathcal{P}})$ *is a structure* $H = (N, \longrightarrow, \mathcal{B}, n_0, F)$ *where* N *is a finite nonempty set of* nodes, $n_0 \in N$ *is the initial node,* $\mathcal{B}$ *a finite nonempty set of causal MSCs,* $\longrightarrow \subseteq N \times \mathcal{B} \times N$ *is the transition relation, and* $F \subseteq N$ *is the set of final nodes.*

Causal HMSCs and HMSCs do not differ in their definition, but rather in their semantic interpretation. The notions of paths, cycles, accepting paths, etc in causal HMSCs are the same as in HMSCs. Let H be a causal HMSC and $\rho = n_0 \xrightarrow{B_1}$

$n_1 \xrightarrow{B_2} \cdots \xrightarrow{B_\ell} n_\ell$ be a path of H. The causal MSC generated by ρ, denoted $\circledcirc(\rho)$, is $B_1 \circledcirc B_2 \circledcirc \cdots \circledcirc B_\ell$. We let $caMSC(H)$ denote the set of causal MSCs generated by accepting paths of H. We also set $Vis(H) = \bigcup\{Vis(M) \mid M \in caMSC(H)\}$ and $Lin(H) = \bigcup\{Lin(M) \mid M \in caMSC(H)\}$. Obviously, $Lin(H)$ is also equal to $\bigcup\{Lin(M) \mid M \in Vis(H)\}$. We shall refer to $caMSC(H)$, $Vis(H)$, $Lin(H)$, respectively, as the causal language, visual language and linearization language of H.

HMSCs can be considered as causal HMSCs labeled with MSCs, and equipped with empty independence relation (for every $p \in \mathcal{P}, I_p = \emptyset$). Hence, a path ρ of a HMSC H generates an MSC by concatenating the MSCs along ρ with operation $\circ$. In such case, $Vis(H)$ is exactly the MSC language denoted by $\mathcal{F}_H$ in chapter 1. Let us recall a basic limitation of HMSCs: their visual languages are *finitely generated*. We can easily show that causal HMSCs generate MSC languages that are not finitely generated, and hence are strictly more expressive than HMSCs. This is an important feature of causal HMSCs. We have already mentioned that many protocols contain sliding windows for efficient messages exchanges. These typical behaviors are not finitely generated. Consider for instance the causal HMSC H over $(\mathcal{P} = \{p, q\}, \{(\Sigma_p, I_p)(\Sigma_q, I_q)\})$ in Figure 4.10, with independence relations given by $I_p = \{((p!q(Q), p?q(A)), (p?q(A), p!q(Q)))\}$ and $I_q = \emptyset$. Clearly, $Vis(H)$ is not finitely generated, as it contains infinitely many MSCs similar to the sliding window behavior shown in chapter 3, Figure 3.1.

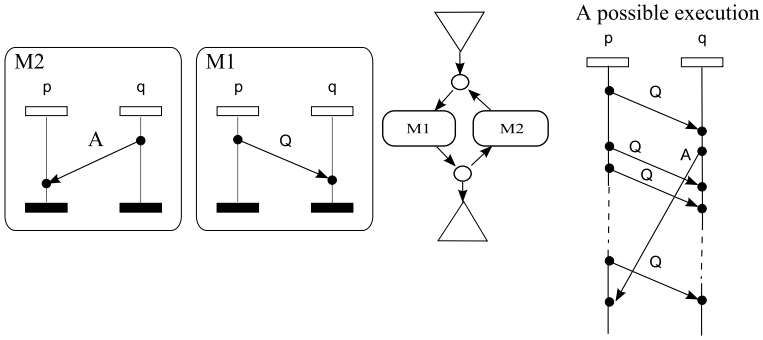


Figure 4.5: An HMSC over two MSCs

2.3 Semantics for causal HMSCs

As things stand, a causal HMSC H defines a set of path $\mathbb{P}_H$ and a set of sequences of causal MSCs $L(H)$, with the same definition as for HMSCs, plus three syntactically different languages, namely its linearization language $Lin(H)$, its visual (MSC) language $Vis(H)$ and its causal MSC language $caMSC(H)$. The next proposition shows that they are also semantically different in general. It also identifies the restrictions under which they match semantically.

Proposition 3 *Let H, H' be two causal HMSCs over the* same *family of trace alphabets $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$. Consider the following six hypotheses:*

(i)	$caMSC(H) = caMSC(H')$	*(i)'*	$caMSC(H) \cap caMSC(H') \neq \emptyset$
(ii)	$Vis(H) = Vis(H')$	*(ii)'*	$Vis(H) \cap Vis(H') \neq \emptyset$
(iii)	$Lin(H) = Lin(H')$	*(iii)'*	$Lin(H) \cap Lin(H') \neq \emptyset$

Then we have:

- $(i) \Rightarrow (ii)$, $(i)' \Rightarrow (ii)'$, $(ii) \Rightarrow (iii)$ *and* $(ii)' \Rightarrow (iii)'$ *but the converses do not hold in general.*
- *If every causal MSC labeling transitions of* H *and* H' *respects* $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$, *then* $(i) \Leftrightarrow (ii)$ *and* $(i)' \Leftrightarrow (ii)'$.
- *If every causal MSC labeling transitions of* H *and* H' *is weak FIFO, then* $(ii) \Leftrightarrow (iii)$ *and* $(ii)' \Leftrightarrow (iii)'$.

A proof of this proposition can be found in appendix. Let us illustrate this proposition with instructive examples, shown in Figure 4.6. We have $Vis(G_1) = Vis(H_1)$ but $caMSC(G_1) \neq caMSC(H_1)$. We also have $Lin(G_2) = Lin(H_2)$ but $Vis(G_2) \neq Vis(H_2)$. One can immediately notice that these inequalities hold for all independence relations.

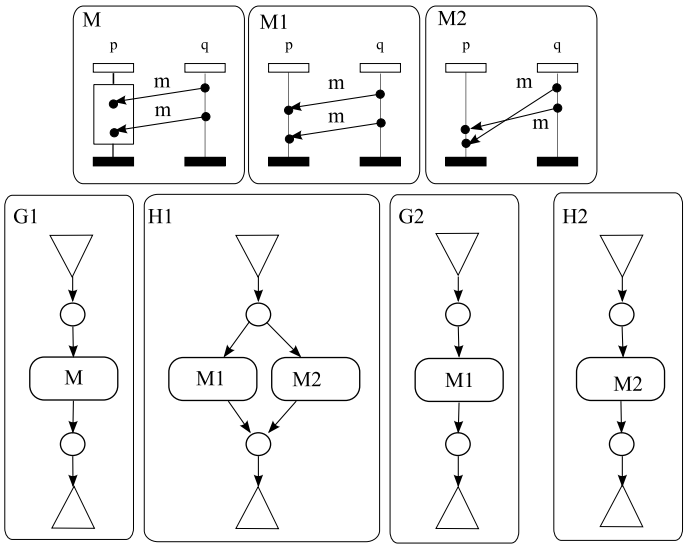


Figure 4.6: Relations between linearizations, visual extensions and causal orders

For most purposes, the relevant semantics for a causal HMSC seems to be its visual language. Indeed, for every causal HMSC H, $Vis(H)$ is an MSC language, that avoids any ambiguity on the ordering of events that may appear in causal MSC languages. With respect to linearizations, is has the advantage of conciseness of partial order models. Properties of visual languages always imply similar properties of linearization languages, and in case of FIFO ordering (which is a reasonable assumption), visual extensions and linearizations of H have similar properties. Note that proposition 3 only relates the semantics of causal HMSCs, and does not mean that inclusion or intersection is decidable in general. In the rest of the chapter, we focus on decidability and regularity issues. We first consider regularity of the linearization languages of causal HMSCs in section 3.1. Then, we focus in section 3.2 on the causal language properties. Using the above proposition 3, it is then straightforward to translate these properties to the visual language of causal HMSCs, when the right hypothesis apply.

3 Decidability for causal HMSCs

3.1 Regular sets of linearizations

It is undecidable in general whether an HMSC has a regular linearization language [113], but a syntactic subclass of regular HMSCs [14,113]) has been identified. Similar syntactic regular class has been identified for compositional HMSCs, and we will show that a decidable syntactic regular subclass also exists for causal HMSCs. The key notion characterizing regular HMSCs is the *communication graph* of an MSC. Nice syntactic subclasses of HMSCs and compositional HMSCs rely of the decidable fact that communication graphs of MSCs labeling cycles have connected or strongly connected communication graphs. We can define a similar notion for causal MSCs. As processes do not necessarily impose an ordering on events, it is natural to focus on the associated Mazurkiewicz alphabet. Thus the communication graph is defined w.r.t the dependency relations $\{D_p\}_{p\in\mathcal{P}}$ used for the concatenation while the dependencies among letters of the same process are disregarded.

Definition 40 *Let $B = (E, \alpha, \{\sqsubseteq_p\}, \mu, \phi)$ be a causal MSC. The* communication graph *of B is denoted by CG_B, and is the directed graph $(Q, \rightsquigarrow)$, where $Q = \alpha(E)$ and $\rightsquigarrow \subseteq Q \times Q$ is given by: $(x, y) \in \rightsquigarrow$ iff*

- *$x = p!q(m)$ and $y = q?p(m)$ for some $p, q \in \mathcal{P}$ and $m \in Msg$, or*
- *xD_py.*

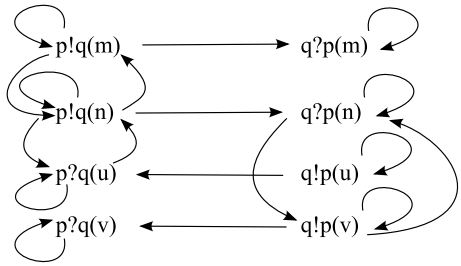


Figure 4.7: Communication graph for causal MSC $B_1 \circledcirc B_2$ of Figure 4.4.

The example of figure 4.7 shows the communication graph for the causal MSC $B_1 \circledcirc B_2$ in Figure 4.4. For instance, there are arrows between $q?p(n)$ and $q!p(v)$ since $q?p(n)\ D_q\ q!p(v)$. However, there is no arrow between $q?p(m)$ and $q?p(n)$, even though some events of $B_1 \circledcirc B_2$ labeled by $q?p(m)$ and $q?p(n)$ are dependent. Note that for a pair of causal MSCs B, B', the communication graph $CG_{B\circledcirc B'} = (Q, \rightsquigarrow)$ can be computed from the communication graphs $CG_B = (Q_B, \rightsquigarrow_B)$ and $CG_{B'} = (Q_{B'}, \rightsquigarrow_{B'})$ as follows: $Q = Q_B \cup Q_{B'}$ and $\rightsquigarrow = \rightsquigarrow_B \cup \rightsquigarrow_{B'} \cup \big(Q^2 \cap \bigcup_{p\in\mathcal{P}} D_p\big)$. Hence, for a fixed set of independence relations, if a causal MSC B is obtained by sequential composition, that is $B = B_1 \circledcirc B_2 \circledcirc \cdots \circledcirc B_k$, then the communication graph of B does not depend on the respective ordering of $B_1, \cdots B_k$, nor on the number of occurrences of each B_i. Hence, for any permutation f on $1..k$ and any $B' = B_{f(1)} \circledcirc B_{f(2)} \circledcirc \cdots \circledcirc B_{f(k)}$, we have that $CG_B = CG_{B'}$.

In the sequel, we will say that the causal MSC B is *tight* iff its communication graph CG_B is weakly connected. We say that B is *rigid* iff its communication graph is strongly connected. We will focus here on rigidity and study the notion of tightness in section 3.2.

Definition 41 *Let $H = (N, \longrightarrow, \mathcal{B}, n_0, F)$ be a causal HMSC. We say that H is* regular *(resp.* globally-cooperative*) iff for every cycle ρ in H, the causal MSC $\circledcirc(\rho)$ is* rigid *(resp.* tight*).*

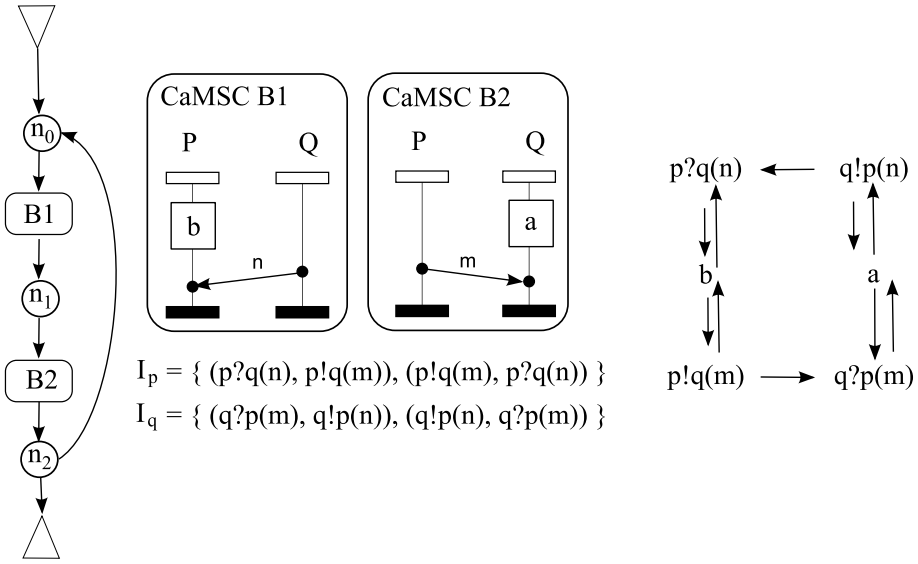


Figure 4.8: A non finitely generated regular causal HMSC and its communication graph.

As for HMSCs or compositional HMSCs, the definitions of regular/globally cooperative causal HMSCs rely on properties of cycles. And as for HMSCs, we can use properties of communication graphs w.r.t. composition to bring the definition back to properties of elementary cycles. Indeed, we have $CG_{B \circledcirc B} = CG_B$, and as already discussed, the rigidity of $B_1 \circledcirc \ldots \circledcirc B_\ell$ does not depend on the order in which $B_1, \ldots, B_\ell$ are listed. Hence, we can use the following equivalent definition to obtain an algorithm: H is regular (resp. globally-cooperative) iff for every strongly connected subgraph G of H with $\{B_1, \ldots, B_\ell\}$ being the set of causal MSCs appearing in G, we have $B_1 \circledcirc \ldots \circledcirc B_\ell$ is rigid (resp. tight). It is easy to see that the simple protocol modeled by the causal HMSC of Figure 4.8 is regular, since the only elementary cycle is labeled by two local events a, b, one message from p to q and one message from q to p. The communication graph associated to this elementary cycle is strongly connected. Note that the visual language of this causal HMSC is not finitely generated, as messages m and n can cross between two occurrences of a and b. The alternative definition leads to a co-NP-complete algorithm to test whether a causal HMSC is regular.

Theorem 21 *Let $H = (N, \longrightarrow, \mathcal{B}, n_0, F)$ be a causal HMSC. Testing whether H is regular (respectively globally-cooperative) can be done in time $\mathcal{O}((|N|^2 + |\Sigma|^2) \cdot 2^{|\mathcal{B}|})$. Furthermore these problems are co-NP complete.*

The proof of this theorem can be found in appendix. It follows the same lines as the one in [56, 113] (one can find a subgraph of H and check its communication graph in polynomial time, hardness comes from an encoding of a SAT problem).

Theorem 22 *Let $H = (N, \longrightarrow, \mathcal{B}, n_0, F)$ be a regular causal HMSC. Then $Lin(H)$ is a regular subset of $\Sigma^\star$, i.e. we can build an automaton $\mathcal{A}_H$ over Σ that recognizes $Lin(H)$. Furthermore, the number of states of $\mathcal{A}_H$ is at most in*

$$\left(|N|^2 \cdot 2^{|\Sigma|} \cdot (\Sigma + 1)^{K \cdot M} \cdot 2^{f(K \cdot M)}\right)^K ,$$

where $K = |N| \cdot |\Sigma| \cdot 2^{|\mathcal{B}|}$, $M = \max\{|B| \mid B \in \mathcal{B}\}$ (recall that $|B|$ denotes the size of the causal MSC B) and the function f is given by $f(n) = \frac{1}{4}n^2 + \frac{3}{2}n + \mathcal{O}(\log_2 n)$.

A complete proof of this theorem can be found in appendix. The main idea of the proof is to show that for every path $\rho = B_1 \dots B_2 \dots B_K$ of a regular causal HMSC H that is long enough, there is only a finite number pairs of causal MSCS $B_i, B_j, i < j$ and events $e \in B_i, e' \in B_j$ such that $e \not\leq e'$. Dependencies among events arise from the fact that cycles necessarily occur in a long enough path, and hence, as H is $s - regular$, the communication graph of each cycle is strongly connected, and creates a causal dependency from events on $\phi(e)$ to events on $\phi(e')$.

The last step of the proof is to show that each linearization of a path of H can be represented by a sequence of transitions between configurations (these transitions are labeled by letters from Σ). For a given path ρ followed in H configurations only need to memorize subsequences of transitions in ρ in which all events have not yet been executed together with the set of its not-yet executed events. When H is s-regular, this information is bounded and contains at most $K_{residue} = |N| \cdot |\Sigma| \cdot 2^{|\mathcal{B}|} \cdot \max\{|B| \mid B \in \mathcal{B}\}$ events.

3.2 Inclusion and Intersection of causal HMSC Languages

As shown in chapter 2, problems of inclusion, equality and non-emptiness of the intersection of the MSC languages associated with HMSCs are all undecidable. Obviously, these undecidability results also apply to the causal languages, visual languages, and linearization languages of causal HMSCs. As in the case of HMSCs, linearizations inclusion and intersection problems are decidable for regular causal HMSCs since their linearization languages are regular. These decidability results can be brought back to visual languages if the considered behaviors are all weak FIFO, and to causal languages if causal MSCs of $\mathcal{B}$ respect $\{(\Sigma_p, I_p)\}_{p \in \mathcal{P}}$.

It is natural to ask whether we can still obtain positive results for these problems beyond the subclass of regular causal HMSCs. In the setting of HMSCs and compositional HMSCs, there is a bound b such that b-bounded linearizations of any globally-cooperative HMSC form a regular set of representative linearizations. Unfortunately, the results of [55] uses Kuske's encoding [91] into traces that is based on the existence of an (existential) bound on communication channels. Consequently, this technique does not apply to globally-cooperative causal HMSCs, as the visual language of a causal HMSC needs not be existentially bounded. For instance, consider the causal HMSC H of Figure 4.9. It is globally-cooperative and its visual language contains MSCs shown in the right part of the figure: in order to receive

the first message from p to r, the message from p to q and the message from q to r have to be sent and received. Hence every message from p to r has to be sent before receiving the first message from p to r, which means that H is not existentially bounded.

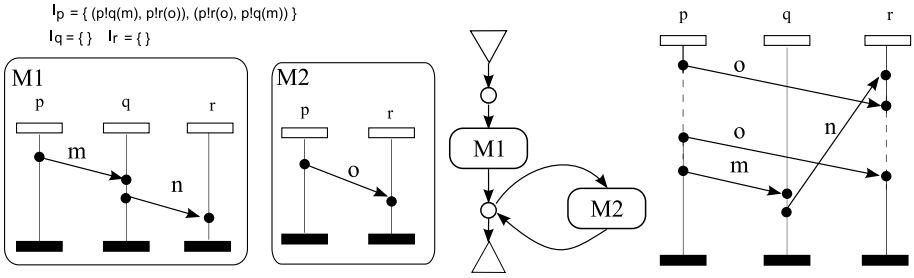


Figure 4.9: A globally-cooperative causal HMSC that is not existentially bounded

This does not mean however that inclusion and intersection are undecidable outside the class of regular causal HMSCs, but rather that we need to rely on different proof techniques. We shall instead use atoms [3, 71], and adapt the notion of atomic language from [56], that we have recalled in chapter 1. As for HMSCs, we want to compare automata labeled by atoms rather than a set of linearizations. First of all, we need to adapt the notion of MSC atom defined in chapter 1, definition 7 to causal MSCs.

Definition 42 *A causal MSC B is a* basic part *(w.r.t. the trace alphabets $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$) if there do not exist causal MSCs B_1, B_2 such that $B = B_1 \circledcirc B_2$.*

Note that we have required in definition 37 that the set of events of a causal MSC is not empty, which simplifies the definition of a basic part. Basic parts are like atoms in MSCs, but in addition to the requirement of avoiding message splitting, they have to preserve concurrency and dependencies among events that can not be obtained by concatenation. Now for a causal MSC B, we define a *decomposition* of B to be a sequence $B_1 \cdots B_\ell$ of basic parts such that $B = B_1 \circledcirc \cdots \circledcirc B_\ell$. For a set $\mathcal{B}$ of basic parts, we associate a trace alphabet $(\mathcal{B}, I_\mathcal{B})$ (w.r.t. the trace alphabets $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$) where $I_\mathcal{B}$ is given by: $B\ I_\mathcal{B}\ B'$ iff for every p, for every $a \in Alph_p(B)$, for every $a' \in Alph_p(B')$, it is the case that $a\ I_p\ a'$. As for MSCs, when $B\ I_\mathcal{B}\ B'$, we have $B \circledcirc B' = B' \circledcirc B$.

We let $\sim_\mathcal{B}$ be the corresponding trace equivalence relation and denote the trace containing a sequence $u = B_1 \ldots . B_\ell$ in $\mathcal{B}^\star$ by $[u]_\mathcal{B}$ (or simply $[u]$). For a language $L \subseteq \mathcal{B}^\star$, we define its trace closure $[L]_\mathcal{B} = \bigcup_{u\in L} [u]_\mathcal{B}$. We can prove (see [54] for details) that the decomposition of any causal MSC B is unique up to commutation. More precisely, when $B_1 \ldots B_k$ is a decomposition of a causal MSC B, then the set of decompositions of B is $[B_1 \ldots B_k]$. It is thus easy to compute the (finite) set of basic parts of a causal MSC B, denoted $Basic(B)$, since it suffices to find one of its decompositions. Decomposition can be performed as for MSCs atoms using a connected components algorithms, but taking into account the dependencies. Unsurprisingly, we then have:

Proposition 4 *For a given causal MSC B, we can effectively decompose B in time $O(|B|^2)$.*

Using the result of Proposition 4, we can consider, without loss of generality, that causal HMSCs are automata labeled by basic parts. Indeed, we can simply decompose each causal MSC in H into basic parts and decompose any transition of H into a sequence of transitions labeled by these basic parts. Given a causal HMSC H over a set of causal MSCs $\mathcal{B}$, we let $Basic(H)$ be the set of basic parts labeling transitions of H, that is
$Basic(H) = \{B \mid \exists X \in \mathcal{B}, B_1, \ldots, B_k$ causal MSCs , $B_1 ... B ... B_k$ is a decomposition of $X\}$.
Trivially, a causal MSC is uniquely defined by its basic part decomposition. Then instead of the visual language we can use the *basic part language* of H, denoted by $BP(H) = \{B_1 \ldots B_\ell \in Basic(H)^\star \mid B_1 \circledcirc \ldots \circledcirc B_\ell \in caMSC(H)\}$. Notice that $BP(H)$ is closed by commutation, that is $BP(H) = [BP(H)]$. We can also view H as a finite state automaton over the alphabet $Basic(H)$, and denote by $\mathcal{L}_{Basic}(H) = \{B_1 \cdots B_\ell \in Basic(H)^\star \mid n_0 \xrightarrow{B_1} n_1 \cdots \xrightarrow{B_\ell} n_\ell$ is an accepting path of H$\}$ its associated (regular) language.

We show in appendix that $BP(H) = [\mathcal{L}_{Basic}(H)]$. As for HMSCs, when a causal HMSC is globally cooperative, the closure $[\mathcal{L}_{Basic}(H)]$ is a star-connected language of basic parts, that is recognized by a finite automaton, and we can use the results of [113] to obtain the following decidability and complexity results:

Theorem 23 *Let H, H' be causal HMSCs labeled by basic parts and defined over the* same *family of trace alphabets $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$. Suppose H' is globally-cooperative. Then we can build a finite state automaton $\mathcal{A}'$ over $Basic(H')$ such that $\mathcal{L}_{Basic}(A') = [\mathcal{L}_{Basic}(H')]$. Moreover, $\mathcal{A}'$ has at most $2^{O(n\cdot b)}$ states, where n is the number of nodes in H and b is the number of basic parts in $Basic(H)$. Consequently, the following problems are decidable:*

(i) Is $caMSC(H) \cap caMSC(H') = \emptyset$?

(ii) Is $caMSC(H) \subseteq caMSC(H')$?

Furthermore, the complexity of (i) is PSPACE-complete and that of (ii) is EXPSPACE-complete.

The above theorem shows that we can model-check a causal HMSC against a globally-cooperative causal HMSC specification. If H and H' are not labeled by basic parts, one can easily transform them in equivalent causal HMSCs labeled by basic parts (this can be done in $O((t.gb^2 + t'.gb'^2)$, where t (resp t') is the size of the transition relation of H (resp. H'), and gb is the size of the largest causal MSC labeling H (resp H'). If the causal HMSCs H, H' in theorem 23 satisfy the additional condition that every causal MSC labeling the transitions of H and H' respects $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$, then we can compare the visual languages $Vis(H)$ and $Vis(H')$, thanks to Proposition 3 of this chapter.

Note that we can only apply Theorem 23 to causal HMSCs defined with the *same* independence relation. If the independence relations are different, the atoms of H and H' are unrelated, and the results on traces of [113] do not apply. Even

worse, the following Theorem 24 states that comparing the MSC languages of two globally-cooperative causal HMSCs H, H' using different independence relations is actually undecidable. The only way to compare causal HMSCs with distinct independence relations is then to compare their linearization languages. Currently for this problem, a solution exists only for regular causal HMSCs.

Theorem 24 *Let G, H be globally-cooperative causal HMSCs with respectively families of trace alphabets $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$ and $\{(\Sigma_p, J_p)\}_{p\in\mathcal{P}}$, where for each p, I_p and J_p are allowed to differ. Then determining if $Vis(G) \cap Vis(H) = \emptyset$ is undecidable.*

3.3 Window-bounded causal HMSCs

The main interest of causal HMSCs is to allow the specification of behaviors containing braids of arbitrary size such as those generated by sliding windows protocols. Very often, sliding windows protocols appear in a situation where two processes p and q exchange bidirectional data. Messages from p to q are of course used to transfer information, but also to acknowledge messages from q to p. If we abstract the type of messages exchanged, these protocols can be seen as a series of query messages from p to q and answer messages from q to p. Implementing a sliding window means that a process may send several queries in advance without needing to wait for an answer to each query before sending the next query. Very often, these mechanisms tolerate losses, i.e. the information sent is stored locally, and can be retransmitted if needed (as in the alternating bit protocol). To avoid memory leaks, the number of messages that can be sent in advance is often bounded by some integer k, that is called the size of the sliding window. Note however that for scenario languages defined using causal HMSCs, such bound on window sizes does not always exist. This is the case for example for the causal HMSC depicted in Figure 4.10 below with independence relations $I_p = \{((p!q(Q), p?q(A)), (p?q(A), p!q(Q)))\}$ and $I_q = \{((q?p(Q), q!p(A)), (q!p(A), q?p(Q))\}$. The language generated by this causal HMSC contains scenarios where an arbitrary number of messages from p to q can cross an arbitrary number of messages from q to p, and conversely, as shown in the figure.

Usually, communication protocols impose a bound on the number of messages that can be sent to an distant process without receiving an acknowledgment. When this number is reached, the protocol delays the next sending until it receives the expected acknowledgment, and may also start resending messages that have not yet been acknowledged. A question that naturally arises is to know if the number of messages crossings is bounded by some constant in all the executions of a protocol specified by a causal HMSC. This information is useful, for several reasons; First of all, when a protocol is modeled with causal HMSCs, detecting that the number of messages crossed by a single message is bounded is a good indication that the model is close to a realistic situation. The second reason is that an implementation of a sliding windows protocol will necessarily impose such bound. When a reasonably small bound is guaranteed by the specification, there is no need to count messages and impose an arbitrary limit on the number of unacknowledged messages sent, which could result in unnecessary delays. In the following, we formally define message crossings, give several results on windows bounds that appeared in [54].

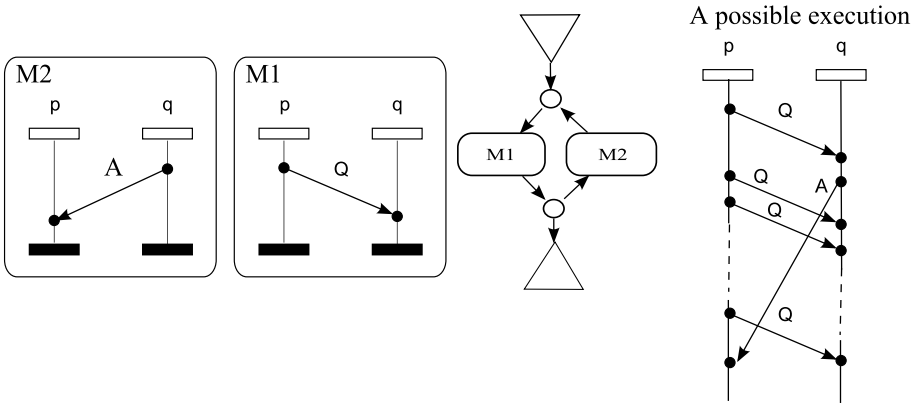


Figure 4.10: A causal HMSC with unbounded message crossings

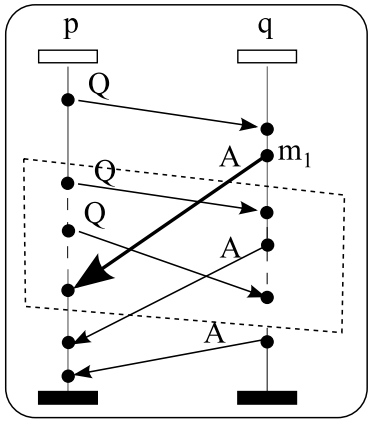


Figure 4.11: Window of message m_1

Definition 43 *Let $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \phi)$ be an MSC For a message (e, f) in M, that is, $(e, f) \in \mu$, we define the* window *of (e, f) in M, denoted $W_M(e, f)$, as the set of messages $\{(e', f') \in \mu \mid \phi(e') = \phi(f)$ and $\phi(f') = \phi(e)$ and $e \leq f'$ and $e' \leq f\}$.*

We say that a causal HMSC H is *K-window-bounded* iff for every $M \in Vis(H)$ and for every message (e, f) of M, it is the case that $|W_M(e, f)| \leq K$. H is said to be *window-bounded* iff H is K-window-bounded for some K.

Figure 4.11 illustrates notion of window, where the window of the message m_1 (the first answer from q to p) is symbolized by the area delimited by dotted lines. It consists of all but the first message Q from p to q. Clearly, the causal HMSC H of Figure 4.10 is not window-bounded, as for a chosen K, one can always exhibit an execution of this HMSC in which one message A from q to p is crossed by more than K messages from p to q. However, we can show that window-boundedness of a causal HMSC can be effectively checked.

Theorem 25 *Let $H = (N, \longrightarrow, \mathcal{B}, n_0, F)$ be a causal HMSC. Then we have:*

(i) If H is window-bounded, then H is K-window-bounded, where K is at most $b \cdot |N| \cdot |\Sigma|$, where $b = max\{|B| \mid B \in \mathcal{B}\}$.

(ii) Further, we can effectively determine whether H is window-bounded in time $O(s \cdot |N|^2 \cdot 2^{|\Sigma|})$, where s is the sum of the sizes of causal MSCs in $\mathcal{B}$.

The proof of this theorem if provided in appendix. The principle is to build an automaton that follows paths of H (that is which transitions are labeled by causal MSCs). It chooses randomly a message m, and then its states memorize the kind of messages that can still overtake m. The number of overtaking message types is decreasing along paths of this automaton, as dependency relations may forbid future crossings. If the automaton contains a cycle in which at least one kind of message can overtake m infinitely often, then the causal HMSC is not window bounded. A similar automaton construction is performed, considering transitions backwards, to handle overtaking of messages appearing before m along a path.

A similar technique can be used to detect if the visual language of a causal HMSC contains only FIFO behaviors.

4 Relationship with Other Scenario Models

Causal HMSCs are a powerful model, that allows for the modeling of sliding windows-like behaviors. It was shown in chapter 3 that this kind of behavior can not be modeled by High-level Message Sequence Charts, but can be represented by Compositional HMSCs. An immediate question is whether causal HMSCs and compositional HMSCs have the same expressive power. We will show later in this section that causal HMSCs can not model any Communicating Finite State Machine, even for the restricted class of bounded machines. On one hand, this could be considered as bad news, as causal HMSCs can not be seen as an implementation model. On the other hand, as causal HMSCs do not embed the whole expressive power of CFMs, more problems might be tractable for causal HMSCs than for cHMSCs. Indeed we can easily show the following theorem.

Theorem 26 (simple message problem for causal HMSCs) *Let H be a causal HMSC. Then one can decide in linear time if there exists a MSC M in $Vis(H)$ containing a message m.*

Though this theorem is trivial, it has some importance, as one has to remember that this problem was in general undecidable for compositional HMSCs. This problem becomes decidable for safe compositional HMSCs, as it can be brought back to a pattern matching problem proved decidable in [58](proposition 23).

In the rest of this section, we compare the expressive power of HMSCs and cHMSCs subclasses with causal HMSCs subclasses. For comparison, we will only consider weak-FIFO scenario languages, that is HMSCs that are labeled by weak FIFO MSCs and causal HMSCs that are labeled by causal MSCs which visual extensions are weak FIFO. Compositional HMSCs generate only weak-FIFO MSCs, by definition. A first reason to consider only weak-FIFO MSC languages is that non weak-FIFO scenarios can be seen as a little degenerate descriptions, as they can be differentiated by their visual languages, but not by their linearization languages. A second reason is that one can easily impose weak FIFOness in definitions of HMSCs causal HMSCs. The last argument is that checking weak-FIFOness of the MSC language of a HMSC or causal HMSC is decidable, and hence one can easily partition known subclasses according to the character of the communications. So, we do not consider FIFOness as an element of comparison. This facilitates comparisons with compositional HMSC subclasses. Furthermore, within this weak-FIFO setting, the comparisons established in this section holds both for visual languages and linearization languages.

An important question is the class of Communicating Finite State Machine corresponding to scenario languages implementations. It has been shown in [76] that regular (compositional) HMSCs corresponds to universally bounded CFSMs. The natural model to compare causal HMSCs and CFSMs could be asynchronous cellular automata with type [25], also called mixed machine in [51], which are networks of asynchronous automata. It has been shown in [51] that using the same regular definition as in this chapter, universally bounded mixed model and regular causal (compositional) HMSCs coincide.

Let us first compare subclasses of causal HMSCs. Obviously, a regular causal HMSC is also globally cooperative. Figure 4.9 shows a globally-cooperative causal HMSC which is not in the subclass of regular causal HMSCs. Thus, regular causal HMSCs form a strict subclass of globally-cooperative causal HMSCs.

Let us now compare causal HMSCs and HMSCs. By definition, causal HMSCs, regular causal HMSCs and globally-cooperative causal HMSCs extend respectively HMSCs, regular HMSCs and globally-cooperative HMSCs, as a HMSC is simply a causal HMSC with total causal orderings on processes and dependency relation of the form $D_p = \Sigma_p \times \Sigma_p$. Recall that when a HMSC H is *globally-cooperative* [56], there exists a K, such that the set of K-bounded linearizations form a regular representative set. In other words, one can build an automaton $\mathcal{A}_H$ such that for every $M \in \mathcal{F}_H$, there exists a linearization of M that is K-bounded and recognized by $\mathcal{A}_H$. The class of globally cooperative causal HMSCs does not enjoy such a good property: the visual language of the (globally cooperative) causal HMSC in Figure 4.9 is not existentially bounded, and cannot be recognized by a finite state machine. Hence, there exist globally-cooperative causal HMSCs which visual languages are

not languages of HMSCs.

Last, Figure 4.8 displays a regular causal HMSC whose visual language is not finitely generated, and therefore can not be defined as the language of a HMSC. It follows that (regular/globally-cooperative) causal HMSCs are strictly more powerful than (regular/globally-cooperative) HMSCs.

Let us now compare causal HMSCs and compositional HMSCs. Recall that as for HMSCs, a CHMSC H generates a set of MSCs, denoted $\mathcal{F}_H$ obtained by concatenation of compositional MSCs along a path of the graph, but that some accepting paths of a CHMSC may not generate a correct MSC. The class of CHMSC for which each path generates exactly one MSC is the class of *safe* CHMSC, and is a strict extension of HMSCs. Regular and globally-cooperative HMSCs have also their strict extensions in terms of safe CHMSCs, namely as regular CHMSC and globally-cooperative CHMSCs. It was also shown (see for instance [58], proposition 21) that safe CHMSCs generate only existentially bounded MSC languages.

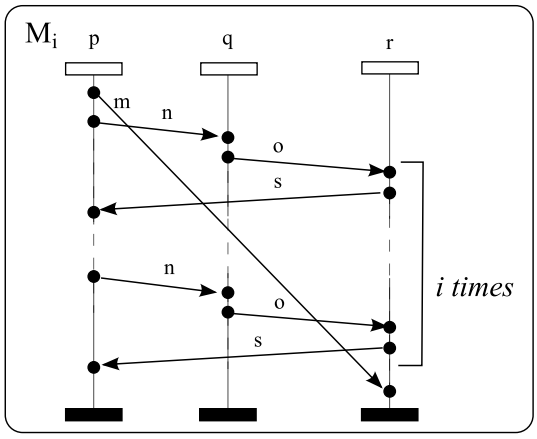


Figure 4.12: A regular (but not finitely generated) set of MSCs

We can now show that the class of regular causal HMSCs is strictly contained in the class of regular compositional HMSCs. It is not hard to build a regular compositional HMSC which MSC language can not be defined with a causal HMSC. An example is a CHMSC H that generates the visual language $Vis(H) = \{M_i \mid i = 0, 1, \ldots\}$, where each M_i consists of an emission of a message m from p to r, then a sequence of i blocks of three messages: a message n from p to q followed by a message o from q to r then a message s from r to p. And at last the reception of message m on r. This MSC language is represented in Figure 4.12. This visual language can be easily defined with a CHMSC, by separating emission and reception of m and iterating a MSC containing messages n, o, s an arbitrary number of times. Clearly, this $\mathcal{F}_H$ is not finitely generated, and it is not either the visual language of a causal HMSC. Assume for contradiction, that there exists a causal HMSC G with $Vis(G) = \mathcal{F}_H$. Let k be the number of messages of the biggest causal MSC which labels a transition of G. We know that M_{k+1} is in $Vis(G)$, hence $M_{k+1} \in Vis(\circledcirc(\rho))$ for some accepting path ρ of G. Let $N_1, \ldots, N_\ell$ be causal MSCs along ρ, where $\ell \geq 2$ because of the size k. It also means that there exist $N'_1 \in Vis(N_1)$, ..., $N'_\ell \in Vis(N_\ell)$ such that $N'_1 \circ \cdots \circ N'_m \in Vis(G)$. Thus, $N_1 \circ \cdots \circ N_\ell = M_j$ for some j, a contradiction since M_j is a basic part (i.e. cannot be the concatenation of two MSCs). That is

(regular) compositional HMSCs are not included into causal HMSCs. On the other hand, regular causal HMSCs have a regular set of linearizations (Theorem 22). Also, it was shown in [76], theorem 3.15 that a MSC language is regular if and only if there exists a bounded CFM with the same linearization language (or equivalently a CHMSC that defines this language). It is then immediate that the class of visual languages of regular compositional HMSCs captures all the MSC languages that have a regular set of linearizations. Hence the class of regular causal HMSCs is included into the class of regular compositional HMSCs. Last, we already know with Figure 4.9 that globally-cooperative causal HMSCs are not necessarily existentially bounded, hence they are not included into safe compositional HMSC.

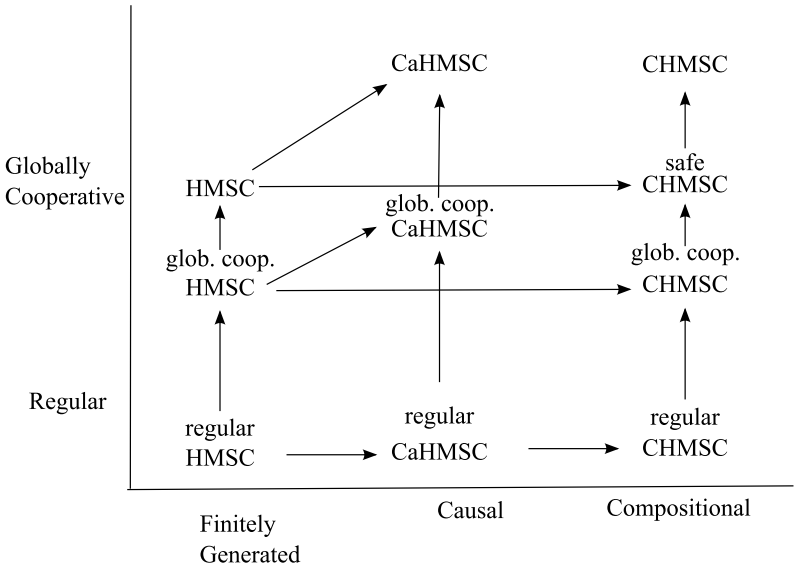


Figure 4.13: Comparison of Scenario languages

The relationships among these scenario models are summarized by Figure 4.13, where arrows denote *strict* inclusion of visual languages. Two classes are incomparable if they are not connected by a transitive sequence of arrows. From this diagram and the preceding results, one can immediately notice that causal HMSCs and compositional HMSCs differ as soon as globally cooperative classes of specifications are considered. This is an important property of the model, as globally cooperative specifications can be seen as a reasonable subclass of scenario languages with respect to decidability and modeling power. However, within these classes, causal HMSCs and compositional HMSCs define disjoint sets of MSC languages.

5 Conclusion

This chapter has considered an extension of HMSCs with commutations called causal HMSCs, that allows for the modeling of braids, such as those appearing in sliding window protocols. Unlike compositional HMSCs, this formalism does not split messages, but extends usual HMSCs using commutations on events located on the same process. As for HMSCs and compositional HMSCs, we have identified decidable syntactic subclasses, namely regular and globally cooperative causal HMSCs with

the same decidable problems as for their HMSC or compositional counterpart. An interesting class that emerges is globally-cooperative causal HMSCs. This class is incomparable with safe compositional HMSCs because the former can generate scenario collections that are not existentially bounded. Yet, decidability results for verification (inclusion, intersection,...) can be obtained for this class.

Let us consider the merits of causal HMSCs beyond the nice theoretical expressiveness and decidability results. Specifying partially ordered events on a process in MSCs was already allowed by the Z.120 standard, using a mechanism called *generalized coregion*. However, concurrency in coregions remains limited to a finite set of events enclosed within a MSC. Causal HMSCs extend HMSCs with commutation relations, hence generalizing the concept of coregion to sets of events of arbitrary sizes. Though the concept of commutation might be rather complex, there is no need to understand the whole theory of traces to use causal HMSCs. Indeed, commutations can be presented as the fact that the sequential ordering among events of some type is not strict. We believe that in many protocols, that are reactive, distributed behaviors can be seen as interleavings of protocol phases, that can be perceived as some closure by commutation of a behavior obtained by ordering strictly these phases. In addition to the material presented in this chapter, a simplified model of the TCP-IP protocol has been designed in [54], to demonstrate the usefulness of the language.

Two interesting problems remain: one is whether the visual language of a causal HMSC is finitely generated. To solve this problem, we shall consider the technique used in [59], where linearizations of atoms of compositional HMSCs are recognized by finite automata (this technique is also presented in chapter 5 of the document). However, it is not yet clear whether such technique can be adapted to CaHMSCs.

A second interesting problem is wheter a causal HMSC can be represented by an equivalent HMSC. Of course, this problem is undecidable in general: the projection of a HMSC on a single process is a regular language, and the projection of a caHMSC on a process is a rational trace language. As regularity is undecidable in general for rational trace languages, one can not decide if a causal HMSC has an HMSC equivalent. However, the problem may find solutions for subclasses of the language.

Another interesting issue is to consider the class of causal HMSCs whose visual languages are window-bounded. The set of behaviors generated by these causal HMSCs seems to exhibit a kind of regularity that could be exploited. Here, regularity shall not be understood as "recognizable by a finite automaton", but rather as the fact that, considering all MSCs in $\mathcal{F}_H$ as graphs, these graphs seem to be definable as the productions of a graph rewriting system using only a finite number of patterns. One can also note that window boundedness is required for causal HMSCs to be finitely generated. Hence, syntactic subclasses of window-bounded causal HMSCs may provide solutions to the finite generation problem mentionned above.

Finally, designing suitable machine models (along the lines of Communicating Finite Automata [32]) and synthesis algorithms for this model is also an important future line of research.

Chapter 5

Towards a partial order algebra ?

On dit qu' on apprend avec ses erreurs, mais à mon avis c'est une erreur... et si je me trompe au moins j' aurai appris quelque chose !
One often says that we learn from our errors, but I think this statement is wrong. And if I am wrong, at least I will learn something.
[Philippe Geluck]

1 Introduction

In previous chapters, we have introduced partial order automata, and shown several ways to assemble finite orders to obtain very expressive formalisms defining MSC languages. The scenario formalisms addressed so far (HMSCs, compositional MSCs and causal MSCs) take as initial paradigm an assembling of finite partial orders by means of automata. The main variants use MSCs that are not communication closed (cHMSCs), or change the way MSCs met along a path of the support automaton are assembled by a sequential composition operator (causal HMSCs). These improvements radically increase the expressive power of partial order automata without changing the nature of their decidable subclasses, nor the complexity classes of decidable problems.

The next step is to define high-level operators for partial order automata. Being able to compose HMSC specifications as in process algebra would allow to use this formalism as a modular specification language, with a global view of interactions and a strong emphasis on concurrency. The usual operators encountered in process algebra are sequence, parallel composition, and projection, and obviously one would like to obtain similar operators for HMSCs. The sequence of two HMSCs is a trivial operation, and preserves most of good properties of a specification. However, it does not bring new modeling power to the model. Projection can be a useful operator to compare partial order automata on their common events, or to extract information on causal dependencies is a large specification.

The most needed operator is a composition of views, that is a parallel composition operator that shuffles MSC languages. Indeed, even an intuitive specification such as Message Sequence Charts has to face the limits of users understanding. Scenarios avoid state space explosion problem, thanks to the explicit representation of concurrency as partial orders. However, one can not expect a HMSC with hundreds of nodes to be understood by human readers. A possible approach is to decompose a

specification into views of a system, that abstract away processes or internal details of a distributed system. The main idea is that the overall system is an object which projection on chosen set of processes and events is one of the specified views.

In this chapter we will study projections and parallel compositions of HMSCs. We will first show that parallel composition and projection of HMSCs are not easy to define, as a projection of a HMSC of the parallel composition of two HMSCs may not be expressible as a new HMSC. Furthermore, in general, these operations do not preserve properties of the composed models.

For this reason, we consider a more pragmatic way to assemble scenarios, called fibered product. This product is some kind of synchronous product of HMSCs, in which transitions of two HMSCS are assembled pairwise. We will however show that this kind of composition may produce ill-formed HMSCs, and that defining it synchronized pairs of transitions and how to assemble them can be cumbersome.

The work presented in this section was realized in cooperation with A. Muscholl, B. Genest, P. Darondeau, B. Caillaud. J. Klein, and led to three publications [39, 59, 88].

2 Trivial operators : sequential composition and iteration of HMSCs

The sequencing of two HMSCs is a purely syntactic operation, and is rather straightforward. It is defined as the standard composition of automata.

Definition 44 *Let* $H_1 = (N_1, \longrightarrow_1, \mathcal{M}_1, n_0^1, F_1)$ *and* $H_2 = (N_2, \longrightarrow_2, \mathcal{M}_2, n_0^2, F_2)$ *be two HMSCs defined over disjoint sets of nodes* N_1 *and* N_2*. The sequencing of* H_1 *and* H_2 *is denoted* $H_1.H_2$ *and is the HMSC* $H_1.H_2 = (N_1 \cup N_2, \longrightarrow_1 \cup \longrightarrow_{1.2}, \mathcal{M}_1 \cup \mathcal{M}_2, n_0^1, F_2)$*, where:* $\longrightarrow_{1.2} = \longrightarrow_1 \cup \longrightarrow_2 \cup \{(n, M_\epsilon, n_0^2) \mid n \in F_1\}$ *and* M_ϵ *is the empty MSC.*

Obviously, concatenation of HMSCs does not introduce new cycles. As a consequence, concatenation preserves global cooperation, regularity, and divergence. Furthermore, if H_1 is existentially bounded for some bound b_1, and H_2 is existentially bounded for some bound b_2, then obviously $H_1.H_2$ is $\max(b_1, b_2)$ existentially bounded. However, even if H_1 and H_2 are local HMSCs, the concatenation of H_1 and H_2 needs not be local.

As for concatenation, we can easily define the iteration of a HMSC H.

Definition 45 *Let* $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$ *be a HMSC. The iterated version of* H *is a HMSC* $H^* = (N, \longrightarrow', \mathcal{M}, n_0, F)$*, where* $\longrightarrow' = \longrightarrow \cup \{(n, M_\epsilon, n_0) \mid n \in F\}$.

As for concatenation, iteration may change some properties of a HMSC, as it creates new cycles. The iteration of a regular, globally cooperative, or non-divergent HMSC is not necessarily regular, globally cooperative, or non-divergent. A non-regular, non-globally cooperative, or divergent HMSC remains non-regular, non-globally cooperative, or divergent. Even when H is a local-choice HMSC, H^* can be non-local. Indeed, a final node for instance can be a choice node too, and hence iterating H connects new branches to this node, possibly with distinct deciding instances.

3 Projection

Even if HMSCs are frequently seen as abstract descriptions of systems, the definition of a projection operator remains very useful. First of all, abstraction can help decreasing the complexity for automatic verification of large systems [35, 64, 122], by reducing the state space, interleavings, and so on. In general, the role of a projection is to reduce the size of a system while preserving some property to check. Then, verifying this property on the projected model can be brought back to a verification of the complete model. Projections are also frequently used to define security properties. Non-interference, for instance,is expressed as the capacity for an attacker to distinguish between a fully described system, and a projected version in which secrets (particular actions that should remain hidden to non-authorized users) are abstracted. We will come back to security issues later in chapter 10.

Scenarios are also supposed to remain rather concise. However, in some cases they may have been designed with too many details, which are not relevant for the property to check. Moreover, the details may hide important information concerning the causalities. Finally, a motivation for projecting an HMSC is to be able to verify properties on a model that is hopefully smaller.

Abstraction for HMSCs can be performed by collapsing nodes of the High-level graph, as for standard word automata. Usually, state aggregation is done in order to preserve behaviors and build an equivalent (bisimilar) specification (this can be done by using standard partitioning algorithms [120]). For HMSCs, bisimilarity of two HMSCs is undecidable, so one can not expect to use bisimulation to quotient the node space of HMSC and then build comparable canonical models. Yet, one can always build minimal (w.r.t bisimulation) HMSCs labeled by atoms. This does not bring in general new comparison means, but may help reducing the size of the automata that shall be built to model-check regular of globally cooperative HMSCs. Note however that very often in HMSCs, there is a bijection between the transitions $\longrightarrow$ and the alphabet $\mathcal{M}$ of HMSCs and that quotienting a HMSC does not necessarily result in huge space gain. When nodes aggregation is performed regardless of any equivalence, the abstraction can produce more behaviors than the initial model. Hence, this results in an over approximation of the set of behaviors.

Another solution is to simply forget transitions, that is replace a pair of transitions $(n, M1, n').(n', M2, n'')$ by a single transition $(n, M1, n'')$. With this projection on the set of transition, some causal dependencies may be lost. Thus the result when model-checking properties on the set of remaining transitions and events is only up to rough approximation. Consider for instance the example of Figure 5.1, and let us define as Π_1 a mechanism that abstracts away from HMSC H the transition of the HMSC labeled by MSC $M2$. The result of Π_1 is a new HMSC H_1 with MSC language $\mathcal{F}_{H_1} = M_1 \circ M_3$. In the original model, event a occurred before events b and c, which is not the case anymore in the projected HMSC.

Abstraction can also be defined by projecting away some events or instances in each MSC of $\mathcal{M}$, without changing the graph. This abstraction of an HMSC is still an HMSC, but has as a negative side effect the loss of certain causalities between events. Consider for instance the example of Figure 5.1, and the projection Π_2 that hides from MSCs the sending and reception of message m_1. We then have $\Pi_2(M_1) = M_1'$, $\Pi_2(M_2) = M_2$ and $\Pi_2(M_3) = M_3$. The result of projecting the original HMSC H is a HMSC H_2 such that $\mathcal{F}_{H_2} = \{M_1' \circ M_2 \circ M_3\}$. According to

$\mathcal{F}_{H_2}$, event a is not a causal predecessor of events b and c anymore. In general, one can remark for such projection that $\Pi_2(M_1 \circ M_2) \neq \Pi_2(M_1) \circ \Pi_2(M_2)$.

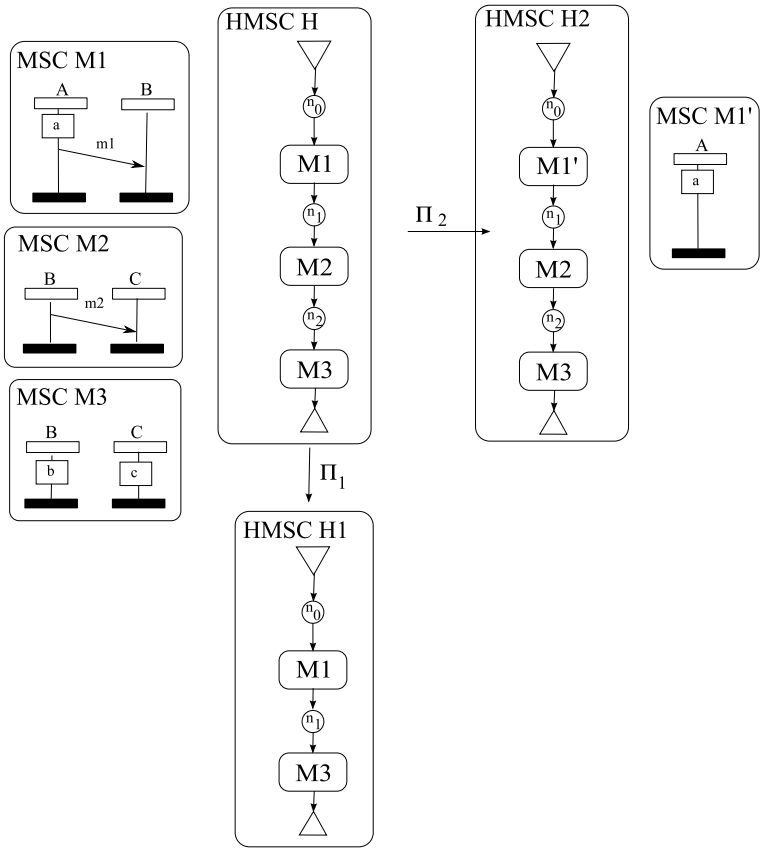


Figure 5.1: Two "bad" projection mechanisms (w.r.t. causal dependencies)

The projection mechanisms Π_1 and Π_2 defined above are simple syntactic projections and, the semantics of the projected model is again an over approximation of projection of real behaviors of the original model. A property referring to causal ordering of events (for instance "it is always the case in $\mathcal{F}_H$ that a is a causal predecessor of b") may not be preserved.

In this section, we present a projection mechanism that gives an exact abstraction of HMSCs. This abstraction hides (projects away) specified events while preserving the causalities among remaining events. This can have several benefits. First, it can provide a better comprehension of the interactions between particular instances (see if actions performed by a process can have an influence of the actions performed by another one). Second, when comparing two MSCs, a designer might be interested in comparing the behaviors involving common features of both scenarios. Hiding information while preserving causal dependencies becomes then a central point for this kind of comparison.

In this section, we propose an abstraction that preserve the causal order, hence the property "a precedes b and c"' of the example above.

The main problem raised by projections of HMSCs that preserve the causalities is the representation of the projected HMSC. First, the projection of an MSC is not always an MSC, as hiding may produce events that represent at the same time sends

and receives. A more severe problem is that, projected HMSCs (even bounded ones, [14,113]) cannot be represented by means of a *finite HMSC*, in general. However, in this section, we first show that we can always represent the projection of an HMSC by a safe *compositional* HMSC (cHMSCs, [65]). Moreover, we give an algorithm that tests whether the projection of an HMSC can be represented by an HMSC : we can decide in polynomial space whether a safe cHMSCs (in particular, the projection of an HMSC) can be represented by an HMSC. This result is then used to give an effective construction of an HMSC representing the projection of the original HMSC, whenever this is possible. We then show that model-checking projections of HMSCs is decidable under some reasonable assumptions.

3.1 Projections of MSCS and HMSCs

Let us illustrate on an example how a "semantic" projection works. Figure 5.2-a shows an example of MSC projection. This MSC can be seen as a partial order, which Hasse diagram is represented in Figure 5.2-b. The projection of this example on events e_1, e_2, e_3, e_5, e_8 is shown in Figure 5.2-c, and is simply the projection of the partial order on remaining events. Note that the dependency from e_1 to e_5 is not a message anymore, but this difficulty can be dealt with by extending event types, that is allow an event in a MSC to be at the same time a set of receptions from a set of processes and a set of sendings to another set of processes.

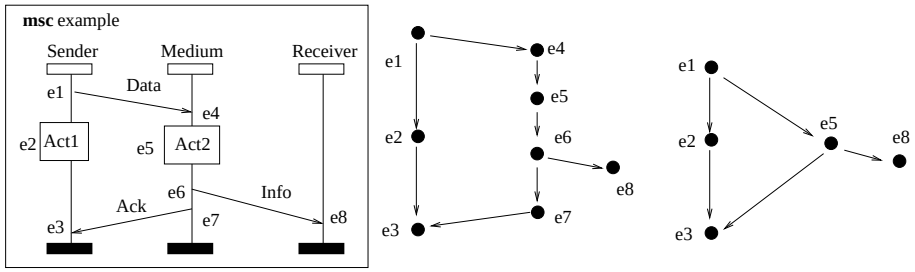


Figure 5.2: a) MSC example b) partial order associated c) projection

Let us now formally define projections of MSCs and HMSCs.

Definition 46 *Consider an MSC $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \phi)$ and a subset of events $E' \subseteq E$. The* projection *of M on E' is denoted $\pi_{E'}(M)$, and is the restriction of the poset M to E', defined as $\pi_{E'}(M) = (E', (<'_p)_{p\in\mathcal{P}}, \alpha', \mu', \phi')$, where $<'_p, \alpha', \phi'$ are the restrictions of $(<_p)_{p\in\mathcal{P}}, \alpha, \phi$ to E'. We also set $\mu' = \{(e, f) \in E' \times E' \mid e \lessdot' f$ and $P(e) \neq P(f)\}$. Events from the set E' will be called* non-erased *events.*

Intuitively, the *projection* of M on E' is obtained by erasing the events in $E \setminus E'$, and inheriting the causal dependencies from M. The set E' can represent for example all the events located on a subset of processes (instances). Note that projecting on a subset of events $E' \subseteq E$ can break communications, as E' need not be communication-closed. The causal dependency among events in a projected MSC is not simply a restriction of the original ordering relation. Slightly abusing our notation, we denote causality among events located on different processes in a projection by a causality (aka message) relation μ'. However, μ' is usually a larger

relation that simply $\mu \cap E' \times E'$. Indeed, $(e, f) \in \mu'$ if e and f are events located on different processes, $e < f$ in M and there is no intermediate non-erased event $g \in E'$ with $e < g < f$. The projection of an MSC will be called a *pMSC* for short.

Note that a pMSC is not necessarily an MSC, since an event in the projection may gather several actions of the initial MSC (these events will be called *multi-type events*). The example of Figure 5.2-c shows the projection of the MSC in Figure 5.2-a on $E' = \{e_1, e_2, e_3, e_5, e_8\}$. Event e_5, which is associated with a sending action in the original MSC can be considered as a "merge" of e_4 ,e_5, e_6 and e_7 in the projection. Since $e_1 \lessdot e_5$, we have to create a message between e_1 and e_5 to keep ordering. Similarly, as $e_5 \lessdot e_8$, we have to create a message between e_5 and e_8. In the projection, event e_5 has *several types*: it represents a receive from the *Sender* process and two sends to *Receiver* and *Sender*. However, multi-type events are not a real problem for modeling, as pMSCs are still partially ordered event sets, with information on locality of events. The sequential composition of pMSCs is defined alike that of MSCs.

We can define *atomic* pMSCs similarly to MSCs: A pMSC M is atomic if the connection graph $Conn(M)$ is strongly connected. For example, the pMSC $M' = \pi_{\{e_1,e_2,e_3,e_5,e_8\}}(M)$ in Figure 5.2-c is atomic. Thus, in addition to the edges represented in the Hasse diagram of Figure 5.2-c the connection graph $CG(M')$ contains the back edges $(e_5, e_1), (e_3, e_5)$ and (e_8, e_5), and is strongly connected. Note that projections do not preserve atomicity nor atoms sizes and number. In general, atoms of $\pi_E(M)$ can be larger than those of M. This can be easily seen of the example of Figure 5.2: each message and atomic action is an atom in the original MSC, that is there are 5 atoms of size at most 2. The pMSC obtained by projection is an atom of size 5.

Projection of a MSC M simply consists in defining a subset of $E' \subseteq E$ to preserve, project ordering, and labeling on E', and build a new message relation. We can define a similar notion of projection a HMSC H, but as already mentioned, we need to define a "semantic" projection that is we consider projection of MSCs generated by H, i.e. projection will occur after concatenation. Within this setting, a MSC of $\mathcal{F}_H$ to project may contain several copies of the same MSC form $\mathcal{M}$.

Let $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$ be a HMSC, and let $E_{\mathcal{M}} = \bigcup_{M \in \mathcal{M}} E_M$. To define a projection of H, we choose a subset $E' \subseteq E_{\mathcal{M}}$. For a given path $\rho = (n_0, M_1, n_1).(n_1, M_2, n_2) \dots (n_{k-1}, M_k, n_k)$ of H, one can associate a concatenation of MSCs $M_1 \circ M_2 \circ \dots \circ M_k$. Each M_i is an isomorphic copy of some MSC M from $\mathcal{M}$, and two MSCs $M_i \neq M_j$ are defined over distinct set of events. Hence, for path ρ, one can define k isomorphisms $\phi_1, \dots \phi_k$, so that each ϕ_i is a partial bijections from $E_{\mathcal{M}}$ to E_{M_i}. Projecting $M_\rho = M_1 \circ M_2 \circ \dots \circ M_k$ on $E' \subseteq E_{\mathcal{M}}$ means projecting M_ρ on all copies of E'. We denote $E'(\rho) = \bigcup_{i \in 1..k} \phi_i(E')$ as the set of isomorphic copies of events from E' appearing in a MSC of path ρ. We can then define $\pi_{E'}(M_\rho)$ as the MSC projection $\pi_{E'(\rho)}(M_\rho)$.

Definition 47 *The projection of a HMSC $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$ on a subset of events $E \subseteq E_{\mathcal{M}}$ is called a pHMSC, and is denoted $\pi_E(H)$. An event of pHMSC will be called a* non-erased *event. The set of pMSCs defined by the pHMSC $H' = \pi_E(H)$ is the set*

$$\mathcal{F}_{\pi_E(H)} = \{\pi_{E(\rho)}(M_\rho) \mid \rho \textit{ is an accepting path of } H\ \}$$

With this definition of projection for HMSCs, we ensure that for every $M_\rho \in \mathcal{F}_H$, and for every pair of events e, f in the projection $N = \pi_{E(\rho)}(M_\rho)$, $e \leq_N f$ if and only if $e \leq_{M_\rho} f$.

3.2 Comparing pHMSCs with HMSCs

The description of a pHMSC by an HMSC, together with a projection function, has several drawbacks. First, since causal dependencies are only implicitly given by the HMSC, a projected scenario is difficult to understand. Second, an implicit representation is not convenient for algorithmic manipulations. Third, by projecting an HMSC we usually want to obtain a smaller object, with a more compact representation. An immediate question appears when projecting an HMSC H to some pHMSC $H' = \pi_E(H)$, namely whether there exists some equivalent HMSC G, i.e., such that $\mathcal{L}(G) = \mathcal{L}(H')$? In particular, if H' is equivalent to some HMSC, then there exists a finite set X of generators for $\mathcal{L}(H')$. That is, there exists a finite set X of MSCs such that every $M \in \mathcal{L}(\pi_E(H))$ is a sequential composition of elements from X. We show below two situations that can prevent the existence of such a set X.

The first case is called an *unbounded crossing*. Intuitively, a pHMSC contains an unbounded crossing if there is a communication pattern that can be iterated an arbitrary number of times between two events situated on different processes that are causally related. For example, the HMSC of Figure 5.3-a generates an unbounded crossing for a projection on the instances A and B. Figure 5.3-b shows the partial orders generated by the HMSC of Figure 5.3-a, and Figure 5.3-c shows the partial orders after the projection on A and B. The MSCs in the projection are all atomic, hence there is no finite set generating them.

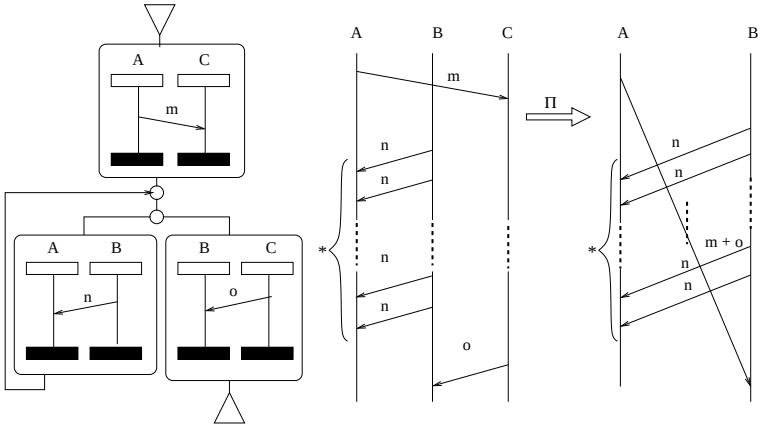


Figure 5.3: a) HMSC generating an unbounded crossing b) MSC c) pMSC

A second situation ruling out a finite representation is called a *crown*, and is a pattern similar to the braids described in chapter 3. Let us illustrate the presence of a crown on an example. Consider the HMSC of Figure 5.4. This HMSC describes scenarios for data transmission and acknowledgment for a multicast protocol called RMTP2 [108], and is an excerpt from a more complete example that can be found in [68]. The RMTP2 network is organized as a tree, propagating data packets

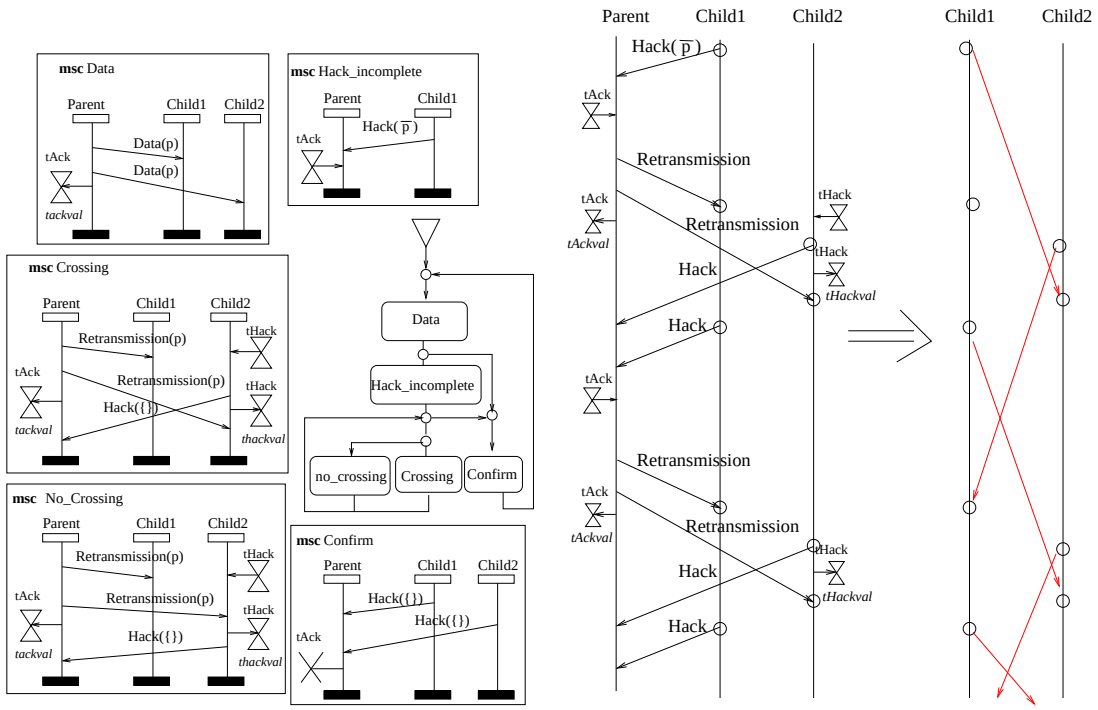


Figure 5.4: A part of RMTP2 protocol and a crown generated by hiding instance Parent

from a data source in the network, and aggregating acknowledgments in order to retransmit missed packets. Some nodes are designated to store a copy of the data sent, and retransmit each missed packet to the child subnetwork, if necessary. When a child receives a data packet, it may send an acknowledgment message *Hack* to its parent. A receiver may also decide to send an acknowledgment after a certain delay *tHackval*. The situation depicted by HMSC of Figure 5.4 shows communications between a node and two children. Child_1 always sends an acknowledgment to its parent upon data reception, while Child_2 always acknowledges data packets after the delay expiration. Furthermore, packets are never missed, but Child_1 can receive corrupted data, and retransmission and *Hack* packets may cross. The left part of Figure 5.4 shows the partial order associated with $(\mathit{Hack\_incomplete} \circ \mathit{Crossing})^*$. The right part of Figure 5.4 shows the same order after hiding the *Parent*, and all timer events. It is clear that without breaking messages (or more precisely causal dependencies between distinct instances) the order obtained after projection can not be defined as a composition of finite communication patterns. This example also shows that crowns appear in HMSCs projections that correspond to real-life protocols, and are not only an artifact of ill-formed HMSCs.

Obviously, if a projection does not abstract away complete messages in a HMSC, one is more likely to obtain a compositional HMSC than HMSCs by projection. Actually, both situations described above can be easily described by cHMSCs, and we will show that pHMSCs and safe cHMSCs have equal expressive power (up to redefinition of event types to allow multi type events). Since pHMSCs involve in general multi-type events we actually need enriched versions of MSCs and cMSCs, by allowing multi-type events. For simplicity, in the rest of this section, we will assume that the projections do not generate multi-type events when comparing pHMSCs

and cHMSCs. However, our constructions can be easily adapted to situations where sending events and receptions are collapsed to form multitype events.

Of course, pHMSCs are more expressive than HMSCs (the pHMSC $\pi_{E_{\mathcal{M}}}(H)$ is exactly the HMSC H. We will show later that cHMSCs are more expressive than pHMSCs, and that one can decide whether a given pHMSC is equivalent to an HMSC. Note that this question is undecidable in general for cHMSCs. Indeed, CHMSCs embed the expressive power of communicating finite state machines, and the question of whether a CFSM is equivalent to some HMSC was shown undecidable by [114].

Figure 5.5 below is an example of cHMSCs that cannot be expressed as the projection of an HMSC. In the cHMSC in the left part of Figure 5.5 two or more isolated send events α are matched after an arbitrary number of β messages. If we have to describe this behavior using a pHMSC, we would need for each event α a new process, that disappears through projection (processes C_1, C_2 in the middle part of Figure 5.5). More formally, for matching a sequence $e_1 < \cdots < e_n$ of sends on process A with a sequence $f_1 < \cdots < f_n$ of receives on process B, we need for each pair e_i, f_i a new process C_i. If for instance $C_1 = C_2$, then $e_1 < e_2 < f_1 < f_2$, hence we do not have $e_2 \lessdot f_2$, that is we do not create a message from e_2 to f_2. Since the number of processes is fixed we therefore cannot describe the behavior of the cHMSC in the right part of Figure 5.5 by a pHMSC.

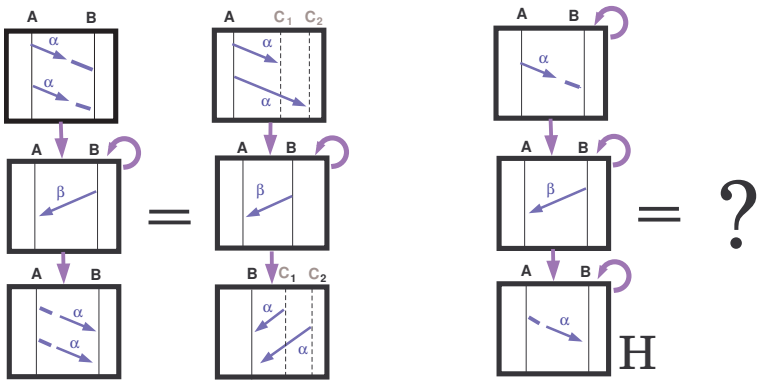


Figure 5.5: The cHMSC in the right part is not a pHMSC.

From the previous examples, we can see that cHMSCs seem good candidates for describing pHMSCs without multi-type events. We can indeed show that pHMSCs correspond precisely to the subclass of *safe* cHMSCs. Given a pHMSC, we can build a cHMSC which transitions are labeled by a single event. The cHMSC is built online while following a linearization of the projected HMSC, and the type of each non-erased is guessed (one can not ensure online whether an event will have a causal successor on a process). Incorrect guesses restricting the transition relation of the cHMSC.

Theorem 27 *[59] Let H be an HMSC with n nodes over a set of processes $\mathcal{P}$, and consider a projection $H' = \pi_E(H)$. Then we can construct a safe cHMSC G that is equivalent to H', of size $n2^{O(\wp^3)}$. Moreover, G is $|\mathcal{P}|^2$-bounded.*

A sketch for the proof of this theorem is given in appendix, and a complete proof can be found in [59].

We now know that any HMSC projection can be translated to a safe and $\wp^2$-bounded CHMSC. We can also show that any safe CHMSC can be seen as a projection of some HMSC. Indeed, we can construct a pHMSC from a cHMSC by introducing new processes remembering that a message was sent but not yet received. As for any path of a safe CHMSC, only a finite number of such events exists, we only need to use a finite number of processes processes memorizing unreceived messages in the projected HMSC.

Consider for instance the pHMSC of Figure 5.6-a). It is safe, and can be translated as a projection of the HMSC in Figure 5.6-b on events located on processes A, B, C. Note that as two messages of type m can be transiting from A to B, one needs to use two additional processes $AB1$ and $AB2$ to simulate message passing.

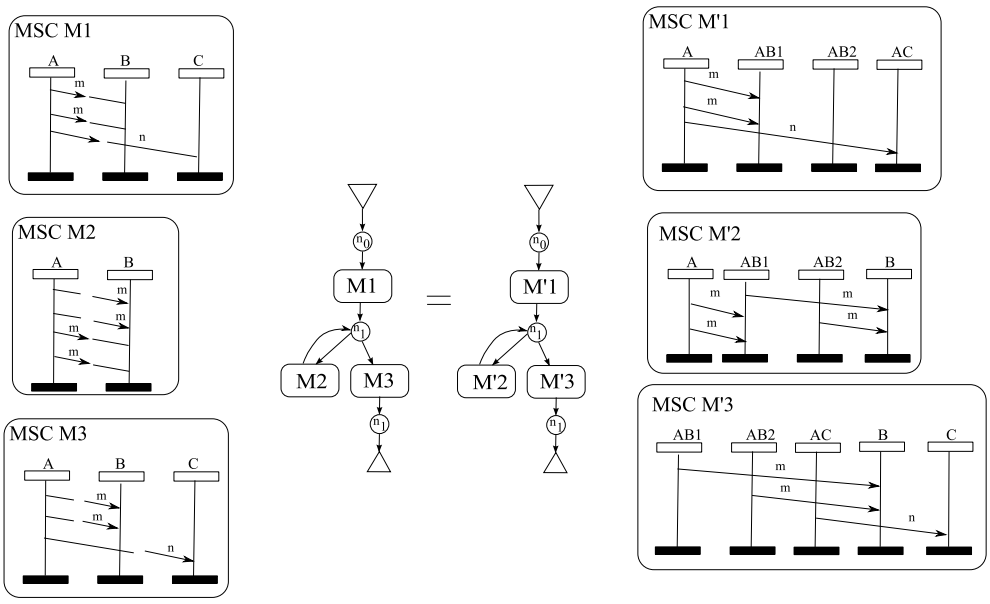


Figure 5.6: A safe CHMSC -a) and an equivalent pHMSC (projected on A, B, C) -b)

Theorem 28 *safe cHMSCs and pHMSCs (without multi-type events) have the same expressive power.*

This result is not really surprising, as projecting a HMSC can not generate behaviors of the form of figure 4.9 in chapter 4. Indeed, in pHMSCs, a sending of a message which is a non-erased event from a MSC M_i in a path of H is necessarily received in a MSC $M_j, j \geq i$ of the same path. This theorem gives additional information on pHMSCs. First as safe CHMSCs are existentially bounded, any pHMSC is also existentially bounded. Concerning cHMSCs, we know from theorem 28 that a safe CHMSC can be seen as the projection of some HMSC.

3.3 Finitely generated pHMSCs and cHMSCS

The next question that immediately arises when projecting a HMSC is whether the projected language can be expressed as a HMSC. This may not be the case, as

illustrated by the examples of figures 5.3 and 5.4, which are not the MSC languages of HMSCs. However, in some cases, the languages obtained by projection can be represented by a HMSC. To be representable by a HMSC, a MSC language L must be finitely generated, that is there must exist a finite set of MSCs X such that every MSC $M \in L$ must be expressible as a concatenation of MSCs chosen from X. In addition, L must be recognizable by a finite HMSC over X (some finitely generated MSC languages are not recognizable, as for instance the language $L = \{M_1^n \circ M_2^n \mid n \in \mathbb{N}\}$ where M_1 and M_2 are finite MSCs).

Theorem 29 *Let G be a safe cHMSC, and let $Atoms(G)$ be the set of atomics factors in $\mathcal{F}_G$. Then we can construct effectively a finite automaton $\mathcal{A}^{gen}(G)$ that accepts only linearizations of $Atoms(G)$, and such that for every $M \in Atoms(G)$ at least one linearization of M is accepted by $\mathcal{A}^{gen}(G)$.*

Using the automaton constructed in Theorem 29 we can test whether a given safe cHMSC (or a pHMSC) is equivalent to an HMSC in polynomial space:

Theorem 30 *Checking whether $Atoms(G)$ is finite for a given safe cHMSC G (pHMSC G, resp.) can be done in PSPACE. Moreover, the problem is co-NP-hard.*

Note that the construction of the automaton $\mathcal{A}^{gen}(G)$ of theorem 29 does not apply for non-safe CHMSCs, as the construction relies of a bound on the number of send and not yet received messages.

3.4 From safe cHMSCs to HMSCs

Theorem 30 gives an algorithm for checking whether the MSC language of a safe cHMSC G (or equivalently of a pHMSC) is finitely generated. The following theorem shows that cHMSCs with finitely generated languages always generate HMSC languages. The proof of the theorem and an effective construction of an equivalent HMSC H for safe CHMSCs with finitely generated languages are provided in appendix.

Theorem 31 *Let G be a cHMSC where $Atoms(G)$ is finite. Let n be the number of nodes of G, e the number of events, maxsize the maximal size of MSCs in $Atoms(G)$ and b the maximal number of unmatched sends on initial paths of G. Then we can construct an equivalent HMSC H from G of size $O((n^{2b} \cdot 2^{\wp b} \cdot e)^{maxsize})$.*

Since there are at most $2^{O(\text{maxsize})}$ atoms, H is of exponential size. A priori, maxsize can be exponential in the size of the pHMSC, yielding an HMSC of doubly exponential size, but we believe this to be very unlikely.

3.5 Model-Checking HMSC projections against HMSCs

Validating HMSCs specifications against regular properties or against other HMSCs is undecidable in general, as shown is chapter 2 (see also [14, 113]), except for the subclasses of *bounded* HMSCs [14,113], and for the larger family of globally cooperative HMSCs [56]. This section considers model-checking pHMSCs against HMSC properties and we show two settings for which the problem is decidable, with

the same complexity as for HMSCs. The next theorem shows that model-checking becomes decidable if a pHMSC is arbitrary, but a HMSC property is globally cooperative:

Theorem 32 *Let G be a globally cooperative HMSC, H an HMSC and $\pi_E(H)$ a projection of H. Then we can check in PSPACE whether $\mathcal{L}(\pi_E(H)) \cap \mathcal{L}(G) = \emptyset$, and in EXPSPACE whether $\mathcal{L}(\pi_E(H)) \subseteq \mathcal{L}(G)$.*

This theorem is a straightforward consequence of decidability of the same properties for globally-cooperative cHMSCs (see theorem 20 chapter 3), as pHMSCs are a subset of safe cHMSCs. Even if the complexities we stated are rather high, note that in practice both the HMSC property and the reduced specification (the HMSC projected on a small part) can be reasonably small.

Theorem 33 *Let G be a regular HMSC, H an HMSC and $\pi_E(G), \pi_{E'}(H)$ their respective projections. Then we can check in PSPACE whether $\mathcal{L}(\pi_E(G)) \cap \mathcal{L}(\pi_{E'}(H)) = \emptyset$, and in EXPSPACE whether $\mathcal{L}(\pi_{E'}(H)) \subseteq \mathcal{L}(\pi_E(G))$.*

This second decidability result is based on the fact that the projection of a regular HMSC preserves boundedness. Hence, one can compare a regular set of representants for $\mathcal{L}(\pi_{E'}(H))$ and the regular language $\mathcal{L}(\pi_E(G))$. This result shows that we can compare two projections of HMSCs (e.g. in order to find common parts), as long as one of them is regular, with the same complexity as for regular HMSCs.

3.6 Conclusion on HMSC projections

Let us summarize the results obtained on HMSC projections. Existentially bounded CFSMs, safe CHMSCs and pHMSCs have the same expressive power (up to some adequate typing of events). One can also decide if a pHMSC or a safe CHMSC has a finitely generated language, which ensures that it can then be represented as an HMSC. As already pointed out, projections may produce larger HMSCs than the original specification. However, this is just a worst-case estimation that is unlikely in practice. The size of the projection depends on the chosen set of non-erased events. For example, hiding a complete instance would probably have a greater impact on the shape of the projection than a random choice of the hidden events. Model checking of pHMSCs is not always possible, and a projection of globally cooperative HMSC is not always globally cooperative. Hence, deciding properties such as $\mathcal{L}(\pi_{E'}(H)) \bigcap \mathcal{L}(\pi_{E'}(H')) \neq \emptyset$ is not decidable in general, even for a pair of globally cooperative HMSCs. However, projection can be compared with globally cooperative HMSCs, and two projections can be compared is one specification is regular.

Even when model-checking is not possible, HMSC projections may still be a useful design tool to extract causality information from scenario descriptions. Moreover, the presence of unbounded crossings or crowns may reveal some properties of the system under study. For example, when a projection hides a communication medium, the presence of a crown can indicate the impossibility to save a consistent global snapshot of the system in some executions, which may call for a redesign of a protocol.

Let us now consider how projection preserves syntactic properties of HMSCs.

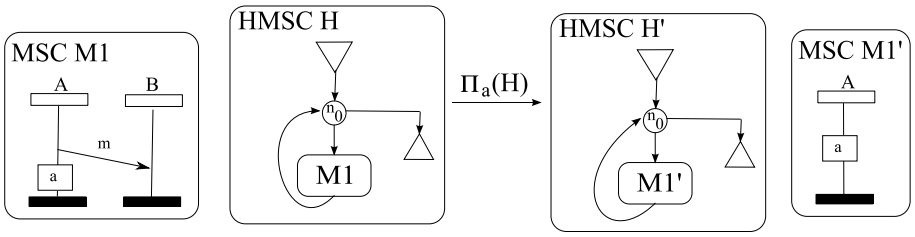


Figure 5.7: Projection of a divergent HMSC

- **Locality** Unfortunately, local-choice HMSCs do not necessarily remain local choice after projection. Consider a HMSC with a single local choice, and project it on events that are not located on the deciding instance. Obviously, this HMSC is not local-choice anymore. Similarly, for a non-local choice HMSC, projection can help erasing instances when a choice is non-local, yielding a local-choice projection.

- **Existential bounds** HMSCs are always existentially bounded, and remain existentially bounded after projection. Though this statement might look surprising, recall that pHMSCs are a subset of safe cHMSCs. Furthermore, erasing a receptions does not increase an existential bound, as it forces to redefine the type of the corresponding sending.

- **Universal bounds** Similarly, HMSCs with universal bounds have regular linearization language. The projection of a regular language is also a regular language, so the linearization language of a pHMSC is also regular. pHMSCs also remains *regular*, as communication graphs of cycles remain connected after projection. This is explained by the fact that projection performs a transitive closure of the ordering, and hence of connected communication graphs. Note that events that were seen as sendings in the original HMSC, but that have no corresponding receive in the projection are seen as atomic actions.

- **global cooperation** Projection does not necessarily preserve connectivity among instances appearing in a cycle. However, if no instance of a cycle is completely erased a globally cooperative HMSC (safe cHMSC) remains globally cooperative after projection.

- **divergence** Divergence is the third property that is not preserved by projection. Consider a simple divergent HMSC, composed of an simple loop in which a message m is sent from a process p to a process q. Clearly, this HMSC is divergent. Now, removing m by projection transforms this divergent HMSC into a non-divergent one. Note that unlike for locality, a non-divergent HMSC can not become divergent by projection. However, non-divergent HMSCs remain non-divergent after projection. Consider the example of Figure 5.7 which is a divergent HMSC. Its projection on event a produces a non-divergent model. This example also shows that non-regular specification can become regular by projection.

4 Products

HMSCs have been designed to be graphical, intuitive, and rather concise. The state space explosion problem which is rapidly met with interleaved models such as finite state machines can be avoided up to a certain limit. However, the size of HMSCs that a designer can draw, understand, and display remains limited. A solution to deal with complexity is to build a set of partial views of a system, and then assemble them automatically. The actual specification should then be designed as a set of tractable size descriptions (for instance one per functionality of a system) or *views*. However, designing a system this ways supposes tools to give a semantics to a set of views, or to assemble them to provide a larger description, preferably using the same language as the one used to define the view, to provide incremental composition.

In this section, we tackle the problem of assembling views defined as HMSCs. We first propose a product of MSC-languages, assuming that views do not share messages but can share internal events. The product we choose is the mixed product of MSCs, that amounts to shuffling their respective events on each process, simultaneously and independently, except for the shared events that are not interleaved but coalesced. The objective of the work started here is to be able to compute the aggregation of MSC-graph descriptions of subsystems, thus yielding support to the modular design of distributed systems.

As shown in previous chapters an important feature of MSC-graphs is existential boundedness [96], that is the existence of an upper bound on the contents of the message channels, within which all MSCs in the language can be run. CHMSCs that are not existentially bounded are hard to implement with usual implementation models such as CFMS. So, it is a desirable property that HMSC views are defined by existentially bounded MSC formalisms, and furthermore that the product of two views remains existentially bounded. Note also that in order to have an HMSC representation, a product of HMSCs must be existentially bounded. However, results on view composition show that knowing whether the product of two MSC-graphs is existentially bounded is undecidable. However, if the shared events belong to only one process, then this question becomes decidable. Once a product is known to be existentially bounded, results on representative linearizations of [55] can be used. Given two globally cooperative CMSC-graphs such that their product is existentially bounded, this product can be represented with a globally cooperative CMSC-graph, and hence all decidability results seen in chapter 2 apply to the product.

4.1 Background

In this section, we will consider weakly FIFO cHMSCs defined over a set of processes $\mathcal{P}$, messages $\mathcal{M}$, and actions $\mathcal{A}$. This choice of weak FIFO specification is not mandatory, but allows for easier definition of CHMSC products. Note however, that results presented in this section apply in a FIFO setting, albeit with more technicalities. Note that the weak FIFO assumption suffices to ensure that a linearization corresponds to a unique (up to isomorphism) CMSC. Processes may perform *send* events $\mathcal{S}$, *receive* events $\mathcal{R}$ and *internal* events $\mathcal{I}$. That is, the set of types of events of an MSC is $\mathcal{E} = \mathcal{S} \cup \mathcal{R} \cup \mathcal{I}$ where $\mathcal{S} = \{p\,!\,q\,(m) \mid p, q \in \mathcal{P}, p \neq q, m \in \mathcal{M}\}$,$\mathcal{R} = \{p\,?\,q\,(m) \mid p, q \in \mathcal{P}, p \neq q, m \in \mathcal{M}\}$, and $\mathcal{I} = \{p\,(a) \mid p \in \mathcal{P}, a \in \mathcal{A}\}$. For each $p \in \mathcal{P}$, we let $\mathcal{E}_p = \mathcal{S}_p \cup \mathcal{R}_p \cup \mathcal{I}_p$ where $\mathcal{S}_p$, $\mathcal{R}_p$, and $\mathcal{I}_p$ are the restrictions of $\mathcal{S}$, $\mathcal{R}$,

and $\mathcal{I}$, respectively, to the considered process p (*e.g.,* $p\,?\,q\,(m) \in \mathcal{S}_p$). Hence, for fixed $\mathcal{P}$, $\mathcal{M}$, and $\mathcal{A}$, CMSCs will be defined as partial orders labeled by $\mathcal{E}$

Given a MSC X we recall that $\mathcal{L}(X)$ is the set of its linearizations (hence a set of words of $\mathcal{E}^*$). For a set $\mathcal{X}$ of MSCs, let $\mathcal{L}(\mathcal{X})$ denote the union of $\mathcal{L}(X)$ for all $X \in \mathcal{X}$. Linearizations can be defined irrespective of MSCs as follows:

Definition 48 *Let $\mathcal{L}in \subseteq \mathcal{E}^*$ be the set of all words w such that for all p, q and m, the number of occurrences $q\,?\,p\,(m)$ is at most equal to the number of occurrences $p\,!\,q\,(m)$ in every prefix v of w, and both numbers are equal for $v = w$.*

Any linear extension w of an MSC belongs to $\mathcal{L}in$. Conversely, a word $w = \epsilon_1 \dots \epsilon_n \in \mathcal{L}in$ is a linear extension of $Msc(w) = (\{1, \dots, n\}, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \phi)$ with:

- $i <_p j$ if and only if $i < j$ and $\phi(i) = \phi(j)$
- $\alpha(i) = \epsilon_i$ and $i <_p j$ if $i < j$ & $\epsilon_i, \epsilon_j \in \mathcal{E}_p$,
- $\phi(i) = p$ if and only if $\alpha(i) \in \epsilon_p$
- $\mu(i) = j$ if the letter $\epsilon_i = p\,!\,q\,(m)$ occurs n times in $\epsilon_1 \dots \epsilon_i$ and the letter $\epsilon_j = q\,?\,p\,(m)$ occurs n times in $\epsilon_1 \dots \epsilon_j$ for some p, q, m, n.

Note that thanks to the weak FIFO assumption, the mapping μ is unique, and hence so is $Msc(w)$.

Definition 49 *Two words $w, w' \in \mathcal{L}in$ are equivalent (notation $w \equiv w'$) if $Msc(w)$ and $Msc(w')$ are isomorphic. For any language $\mathcal{L} \subseteq \mathcal{L}in$, we write $[\mathcal{L}] = \{w \mid w \equiv w', w' \in \mathcal{L}\}$. A language $\mathcal{L} \subseteq \mathcal{L}in(\mathcal{X})$ is a* representative set *for $\mathcal{X}$ if $\mathcal{L} \cap \mathcal{L}in(X) \neq \emptyset$ for all $X \in \mathcal{X}$, or equivalently, if $[\mathcal{L}] = \mathcal{L}in(\mathcal{X})$.*

We deduce the following properties. For any MSC X, $\mathcal{L}in(X)$ is an equivalence class in $\mathcal{L}in$. For any MSC X and for any $w \in \mathcal{L}in$, $w \in \mathcal{L}in(X)$ if and only if X is isomorphic to $Msc(w)$. A similar property does not hold for arbitrary CMSCs. For instance, $(p\,!\,q\,(m))\,(q\,?\,p\,(m))\,(q\,?\,p\,(m))$ belongs to $\mathcal{L}in(X)$ for two different CMSCs X, where the send event is matched by μ either with the first or with the second receive event.

Figures 5.8 and 5.9 show two HMSCs. Concatenating *OK* and the local event *sync* gives an MSC with 3 events. The reception of *OK* and the event *sync* are unordered (in G_1). On the contrary, the event *sync* and the reception of *Void* are ordered (in G_2).

A safe CHMSC G may always be expanded into a safe *single-event* CHMSC-graph G', that is a graph in which each transition is labeled with a single event CMSC, such that $\mathcal{L}(G) = \mathcal{L}(G')$. In the following, every safe CHMSC is assumed to be single-event. The expansion yields because one can always build a regular representative set for $\mathcal{L}(G)$.

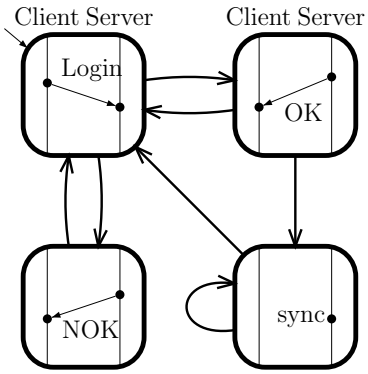


Figure 5.8: Identification Scenario G_1.

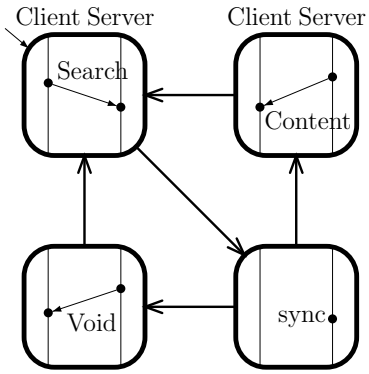


Figure 5.9: Searching Scenario G_2.

4.2 Mixed Product of MSC-languages

In order to master the complexity of distributed system descriptions, it is desirable to have at one's disposal a composition operation that allows to weave different aspects of a system. When system aspects are CMSC-graphs with disjoint sets of processes, the concatenation of their MSC-languages can be used to this effect. Else, some parallel composition is needed: we propose here to synchronize shared events and shuffle non-shared events per process. All shared events are internal events (messages are never shared).

Before entering into formal details, let us illustrate how a mixed product of MSCs should work. Consider the two MSCs M_1 and M_2 in Figure 5.10. They contain one common instance, and two common events, represented as internal actions a, b. The result of a product of M_1 and M_2 is the MSC language $L = \{M_3, M_4\}$.

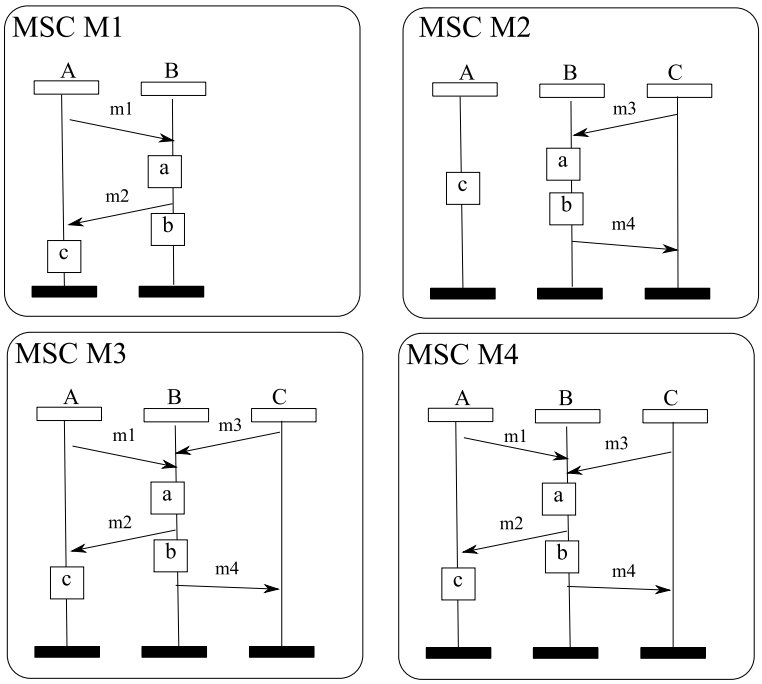


Figure 5.10: Mixed product of two MSCs M_1, M_2

First, we recall the definition of the *mixed product* $L_1 \parallel L_2$ of two languages L_1, L_2 of words (see [44]), defined on two alphabets Σ_1, Σ_2 not necessarily disjoint. Let $\Sigma = \Sigma_1 \cup \Sigma_2$. For $i = 1, 2$ let $\pi_i : \Sigma^* \to \Sigma_i^*$ be the unique monoid morphism such that $\pi_i(\sigma) = \sigma$ for $\sigma \in \Sigma_i$ and $\pi_i(\sigma) = \varepsilon$ otherwise. Then $L_1 \parallel L_2 = \{w \mid \pi_i(w) \in L_i, i = \{1, 2\}\}$ is the set of all words $w \in \Sigma^*$ with respective projections $\pi_i(w)$ in L_i. E.g., $\{ab\} \parallel \{cad\} = \{cabd, cadb\}$.

Definition 50 *For $i = \{1, 2\}$, let $\mathcal{X}_i$ be a MSC-language over $\mathcal{E}_i$, such that $x \in \mathcal{E}_1 \cap \mathcal{E}_2$ implies $x = p(a)$ for some p, a. The* mixed product *$\mathcal{X}_1 \parallel \mathcal{X}_2$ is $Msc\,(\mathcal{L}in(\mathcal{X}_1) \parallel \mathcal{L}in(\mathcal{X}_2))$ and it is a MSC-language over $\mathcal{E}_1 \cup \mathcal{E}_2$.*

The mixed product operation may serve to compose the languages of two CHMSCs that share only internal events, as is the case for the CHMSCs G_1, G_2 of Figures 5.8,5.9. The synchronization on the shared events *sync* ensures that in any MSC in $\mathcal{F}_{G_1} \parallel \mathcal{F}_{G_2}$, the server never answers a search request from the client unless the client is logged in. Note that even though X_1 and X_2 are MSCs, $\{X_1\} \parallel \{X_2\}$ may contain more than one MSC, as illustrated by Figure 5.10, where $M_1 \parallel M_2 = \{M_3, M_4\}$. Also note that $w_1 \parallel w_2 \subseteq \mathcal{L}in$ for $w_1 \in \mathcal{L}in$ and $w_2 \in \mathcal{L}in$. Mixing all linearizations pairwise yields all linearizations of a product:

Proposition 5 $\mathcal{L}in(\mathcal{X}_1 \parallel \mathcal{X}_2) = \mathcal{L}in(\mathcal{X}_1) \parallel \mathcal{L}in(\mathcal{X}_2) = [\mathcal{L}in(\mathcal{X}_1) \parallel \mathcal{L}in(\mathcal{X}_2)]$

Note however that, by considering two particular linerizations of two MSCs X_1, X_2 and computing their mixed product, one obtains a set of linearizations that may not be representative for the mixed product of X_1 and X_2. Indeed, for fixed representations $w_1 \in \mathcal{L}in(X_1)$ and $w_2 \in \mathcal{L}in(X_2)$, $\{X_1\} \parallel \{X_2\}$ may be larger than $Msc\,(w_1 \parallel w_2)$. consider the following example :

$w_1 = (p\,!\,q\,(m_1))\,(q\,?\,p\,(m_1))\,(p\,!\,q\,(m_1))\,(q\,?\,p\,(m_1))$,
$w_1' = (p\,!\,q\,(m_1))^2\,(q\,?\,p\,(m_1))^2$,
$w_2 = (q\,!\,p\,(m_2))\,(p\,?\,q\,(m_2))\,(q\,!\,p\,(m_2))\,(p\,?\,q\,(m_2))$,
$w_2' = (q\,!\,p\,(m_2))^2\,(p\,?\,q\,(m_2))^2$,
$w_3 = (p\,!\,q\,(m_1))^2\,(q\,!\,p\,(m_2))^2\,(p\,?\,q\,(m_2))^2\,(q\,?\,p\,(m_1))^2$.

and $X_1 = Msc(w_1) = Msc(w_1')$, $X_2 = Msc(w_2) = Msc(w_2')$, $X_3 = Msc(w_3)$. Now $X_3 \in Msc(w_1' \parallel w_2')$, but $X_3 \notin Msc(w_1 \parallel w_2)$. This observation shows that products must be handled with care, and can not be dealt with by only considering one representative word per MSC. Indeed, w_1 is a representative for X_1, w_2 is for X_2, but $w_1 \parallel w_2$ is not a set of representatives for $X_1 \parallel X_2$.

4.3 Mixed products and existential bounds

The next question to address is of course how mixed products change bounds on shuffled languages, on MSC languages, and on MSC languages defined using partial order automata. Being B-bounded for an MSC language is important, as it means that this language has a chance to be implementable by standard CFMS. For CHMSCs, we have shown that safe CHMSCs have existentially bounded MSC languages. Safe CHMSCs can be represented by a regular set of representative words, which may allow for verification (when the considered CHMSC is also globally cooperative). Hence, preservation of a bound during product is essential to allow for verification of products of CHMSCs.

Proposition 6 $\mathcal{L}in^b(\mathcal{X}_1 \parallel \mathcal{X}_2) = \mathcal{L}in^b(\mathcal{X}_1) \parallel \mathcal{L}in^b(\mathcal{X}_2)$.

The above result shows that mixed product behaves nicely with respect to bounded linearizations. If $\mathcal{X}_1$ and $\mathcal{X}_2$ are $\forall$-B-bounded, then $\mathcal{L}in(\mathcal{X}_i) = \mathcal{L}in^B(\mathcal{X}_i)$, and their product is also $\forall$-B-bounded. However, it may occur that both $\mathcal{X}_1$ and $\mathcal{X}_2$ are $\exists$-B-bounded but their mixed product is not existentially bounded. For instance let for all j X_1^j be the MSC with j messages m_1 from p to q and X_2^j be the MSC with j messages m_2 from q to p. Both MSCs are $\exists$-1-bounded since $(p!q(m_1)q?p(m_1))^j$ is 1-bounded. Define $\mathcal{X}_1 = \{X_1^j \mid j > 0\}$ and $\mathcal{X}_2 = \{X_2^j \mid j > 0\}$, thus $\mathcal{X}_1$, $\mathcal{X}_2$ are $\exists$-1-bounded, but $\mathcal{X}_1 \parallel \mathcal{X}_2$ is not $\exists$-B-bounded for any B since $Msc(p!q(m_1)^B(q!p(m_2)p?q(m_2))^Bq?p(m_1)^B)$
$\in \mathcal{X}_1 \parallel \mathcal{X}_2$, but is not $\exists$-$(B-1)$-bounded.

An important question regarding MSC-languages and their products is verification. Most often, in decidable cases [56, 113], verifications performed on a MSC-language $\mathcal{X}$ amount to check the membership of a given MSC X, or to check that $\mathcal{L}in(\mathcal{X})$ has an empty intersection with a regular language L (representing the complement of a desired property). In the case of a product language $\mathcal{X}_1 \parallel \mathcal{X}_2$, membership can be checked using the projections, since $X \in \mathcal{X}_1 \parallel \mathcal{X}_2$ if and only if $\pi_i(X) \in \mathcal{X}_i$ for $i = 1, 2$. In order to check regular properties of $\mathcal{L}(G_1) \parallel \mathcal{L}(G_2)$, one often needs computing a safe CMSC-graph G such that $\mathcal{L}(G) = \mathcal{L}(G_1) \parallel \mathcal{L}(G_2)$, and in particular an existential bound B for the product. The example above shows that in general, one can not expect products to preserve bounds. A key question is then whether the MSC language obtained as a product of two safe CHMSCs has an existential bound. Unfortunately, the theorem below shows that one cannot decide whether such G exists when G_1 and G_2 share events on two processes or more.

Theorem 34 *Let G_1, G_2 be two HMSCs. It is undecidable whether $\mathcal{L}(\mathcal{G}_1) \parallel \mathcal{L}(G_2)$ is existentially bounded.*

The proof of this theorem again relies on a PCP encoding. One can easily show that mapping sequences of events from distinct HMSC specifications can be used to encode equality of words. This proof (provided in appendix) is similar to the proof that $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) = \emptyset$ is undecidable for generic MSC-graphs G_1, G_2 [115]. As it builds on emptiness of intersection of languages, the following corollary is immediate.

Corollary 3 *Let H_1, H_2 be two HMSCs. It is undecidable whether $\mathcal{L}(\mathcal{G}_1) \parallel \mathcal{L}(G_2) = \emptyset$.*

Of course, these results are bad news for view composition, as one can not even decide if a specification given in terms of a set of HMSCs contains at least one behavior. This seems an essential property to ensure, as a pair of views may define contradictory ordering for similar events, and such inconsistency should be detected. Even worse, these negative results already hold for globally cooperative HMSCs (the PCP encoding of theorem 34 need globally cooperative HMSCs only). This means that the powerful view composition mechanism we were looking for does not exist, and that we have to rely on weaker composition mechanisms.

4.4 Monitored product of MSC-languages

The negative results of Theorem 34 and 3 motivate the introduction of a a *monitor* process *mp* and a *monitored product* in which all shared events are internal events located on the monitor process.

Definition 51 *Let $\mathcal{X}_1, \mathcal{X}_2$ be two sets of MSCs defined over alphabets $\mathcal{E}_1$ and $\mathcal{E}_2$. The* monitored product *$\mathcal{X}_1 \,|||\, \mathcal{X}_2$ with monitor process mp is defined as the mixed product $\alpha_1(\mathcal{X}_1) \parallel \alpha_2(\mathcal{X}_2)$ where α_1, α_2 are two bijective renamings such that the shared alphabet $\mathcal{SE}$ after renaming by α_1, α_2 is included in $\{mp(a) \mid a \in \mathcal{A}_1 \cap \mathcal{A}_2\}$.*

Intuitively, in a monitored product, $\mathcal{SE}$ describe which events should be considered as common events in both views, with the requirement that all these common events are atomic actions of the monitored process *mp*. For instance, with the cHMSCs of Fig. 5.8 and Fig. 5.9, we can chose $mp = server$ and $\mathcal{SE} = \{mp(sync)\}$ in $\mathcal{F}_{G_1} |||\mathcal{F}_{G_2}$. The adequacy of the monitored product to weave aspects of a distributed system is confirmed by the following theorem.

Theorem 35 *[39] Given two safe CHMSCs G_1, G_2, one can decide in co-NP whether the monitored product of $\mathcal{F}_{G_1} \,|||\, \mathcal{F}_{G_2}$ is $\exists$-bounded.*

The proof of the theorem follows from the following propositions:

Proposition 7 *Given two safe CHMSCs G_1 and G_2, the MSC-language $\mathcal{F}_{G_1} |||\, \mathcal{F}_{G_2}$ is existentially bounded if and only if it is existentially B-bounded for $B = (2\,|\mathcal{P}| + 2)^2 \times (|G_1| + 1) \times (|G_2| + 1) \times K$, where $|G_i|$ is the number of events in G_i and K is the square of the sum $2\,|\mathcal{P}| + (|\mathcal{P}|)^2/2 \times (\,|\mathcal{M}_1| \times |G_1| + |\mathcal{M}_2| \times |G_2|)$.*

Proposition 8 *Given two safe CHMSCs G_1, G_2 and an integer B, one can decide in co-NP whether $\mathcal{F}_{G_1} |||\, \mathcal{F}_{G_2}$ is $\exists$-B-bounded.*

We do not give the complete proof of the propositions, which are rather long and can be found in [39], and in an extended version of this work [40]. However, a proof sketch is provided in appendix.

When $\mathcal{F}_{G_1} |||\mathcal{F}_{G_2}$ is $\exists$-bounded, one may wish to compute a safe CHMSC representation of this MSC-language. This representation can then be used to help designers, but also serve as input to existing tools for model-checking and realization. In [55] a correspondence was established between *globally cooperative* CMSC-graphs [56], and MSC-languages $\mathcal{X}$ with regular representative sets $\mathcal{L}in^B(\mathcal{X})$ for some $B > 0$.

As already mentioned, the undecidability result of theorem 34 holds even when composed views are globally cooperative. Quite remarkably, $\mathcal{F}_{G_1} \cap \mathcal{F}_{G_2} = \emptyset$ is decidable as soon as G_1 or G_2 is globally cooperative [56]. This decidability comes from the fact that globally cooperative models can be equivalently represented by equivalent regular sets of representatives. The results in [55] hold for FIFO cHMSCs with a single message type, but we can extend them to weak-FIFO CHMSCs with message contents. We do not detail the proof, which can be found in [39], but give an intuition for it. One can encode a message sending of the form $p!q(m)$ by a message sending $p!pq_m$, in which pq_m is a new process, which role is to simulate labeling of messages. Similar translation can be performed for receptions. As a result, we

obtain a cHMSC with FIFO buffers and one single message type. One can then built a CHMSC with one message and FIFO communications from an arbitrary cHMSC. Furthermore, we can show that there exists a bijective correspondence between the two kinds of model. Then, as an extension of [55] we obtain:

Theorem 36 *[39] Let $\mathcal{X}$ be a set of (weak-FIFO) MSCs. The following are equivalent:*
– $\mathcal{X} = \mathcal{F}_G$ for some globally cooperative CMSC-graph G,
– $\mathcal{L}in^B(\mathcal{X})$ is a regular representative set for $\mathcal{X}$ for sufficiently large $B > 0$. Moreover, B and a finite automaton recognizing $\mathcal{L}in^B(\mathcal{X})$ can be computed effectively from G. Conversely, G can be computed effectively from $\mathcal{L}in^B(\mathcal{X})$.

Now let G_1, G_2 be two globally cooperative CMSC-graphs. If $\mathcal{L}(G_1) \,|||\, \mathcal{L}(G_2)$ is $\exists$-bounded, then this MSC-language is $\exists$-B-bounded for the bound B defined in Prop. 7. Therefore, $\mathcal{L}in^B(\mathcal{L}(G_1) \,|||\, \mathcal{L}(G_2))$ is a representative set for $\mathcal{L}(G_1) \,|||\, \mathcal{L}(G_2)$. By proposition 6, $\mathcal{L}in^B(\mathcal{L}(G_1) \,|||\, \mathcal{L}(G_2)) = \mathcal{L}in^B(\mathcal{L}(G_1)) \,|||\, \mathcal{L}in^B(\mathcal{L}(G_2))$. Since both G_1, G_2 are globally cooperative, both $\mathcal{L}in^B(\mathcal{L}(G_1))$ and $\mathcal{L}in^B(\mathcal{L}(G_2))$ are regular and effectively computable. Since the shuffle of regular language is regular, we get the following.

Theorem 37 *Let G_1, G_2 be two globally cooperative CMSC-graphs such that $\mathcal{L}(G_1) ||| \mathcal{L}(G_2)$ is $\exists$-bounded. Then one can effectively compute a globally cooperative CMSC-graph G with $\mathcal{L}(G) = \mathcal{L}(G_1) \,|||\, \mathcal{L}(G_2)$. Moreover, G is of size at most exponential in the size of $|G_1|, |G_2|$.*

Proposition 9 *Let H_1, H_2 be two HMSCs, let $X_1 \in \mathcal{F}_{H_1}$ and $X_2 \in \mathcal{F}_{H_2}$ be two MSCs such that $\Pi_{\mathcal{SE}}(X_1) = \Pi_{\mathcal{SE}}(X_2)$. Then $\mathcal{L}(X_1 \,|||\, X_2) \neq \emptyset$.*

An immediate consequence of this proposition is that one can effectively test if the monitored product of two HMSC languages is empty or not. Indeed, the projection of a HMSC over a subset of events located on a single process forms a regular language. Hence, ensuring that $\mathcal{L}(G_1) \,|||\, \mathcal{L}(G_2) \neq \emptyset$ amounts to checking whether $\mathcal{L}(\Pi_{\mathcal{SE}}(G_1)) \cap \mathcal{L}(\Pi_{\mathcal{SE}}(G_2)) = \emptyset$. We then have the following result:

Theorem 38 *Let H_1, H_2 be two HMSCs. Deciding if $\mathcal{F}_{H_1} \,|||\, \mathcal{F}_{H_2} \neq \emptyset$ is PSPACE-complete.*

The monitored product, although less powerful than the product of (c)HMSCs is more decidable. One can check that existential bounds are preserved during composition, and when this is the case, if the composed views are globally cooperative, then one can compute a product, which is also globally cooperative. When the composed views are HMSCs, a simple test allows to decide if the two views describe at least a common consistent run.

4.5 Conclusion on CHMSC and HMSC products

We have shown in this section that assembling of HMSC views by a shuffle operation does not work in practice, as deciding if a product of two globally cooperative HMSCs has an empty language is already undecidable. This may seem surprising, as

globally cooperative HMSCs already allow for model checking. The main reason for undecidability is that in products, HMSCs can be seen as rational relation among observable events located on distinct processes. This immediately gives the undecidability, regardless of the fact that some messages allow for global cooperation in loops.

To overcome this problem, one solution is to enforce common events in views to appear on a single designated process. In this case, we obtain what we called a monitored product of HMSCs or safe CHMSCs, for which it is decidable whether the product language is bounded. Note that monitored product does not necessarily preserve boundedness of the composed views. When the composed graphs are globally cooperative, if an existential bound for the MSCs generated by the monitored product exists, then the monitored product can be represented by a globally cooperative CHMSC. However, the size of the composed model can be exponential in the size of the original models. Beyond the theoretical results brought by this study of HMSC products, these results are rather disappointing in terms of feasibility or practical use of view composition for partial order automata.

5 Fibered Product

The previous section has addressed two notions of products for (c)HMSCs, namely mixed product, and monitored product. Mixed product can not be used to produce a new CHMSC specification out of two CHMSC components, so it is a rather intentional operator. Monitored product can compose safe and globally cooperative CHMSCs, and in some cases outputs a safe and globally cooperative product CHMSCs. This seems more usable in practice, as the class of globally cooperative partial order automata allows for model-checking, implementation, and remains quite expressive. Furthermore, when assembling pieces of a specification, it seems reasonable that each functionality is safe and globally cooperative, and that the result remains globally cooperative too. However, even when a safe and globally cooperative specification can be maintained, section 4.4, theorem 37 shows that a product can be of exponential size (w.r.t the sizes of the composed models). This rapidly hinders applicability of monitored product if monitored product is used several times to design a complete specification.

In this section, we define a less powerful operator, namely a fibered product of HMSCs, but which properties help mastering the size of a product. This product is a kind of synchronous product of HMSCs (with the usual meaning of [16]), where synchronized transitions are labeled by a MSC that merges orderings defined in the MSCs labeling the synchronized transitions. Though this composition seems more limited than a mixed product or monitored product, we will show that is has nice properties with respect to existential and universal bounds and to finite generation.

Another motivation for this work was to be able to design in a compact way specifications with a lot of redundancy, and then build and study complete model by assembling specializations of theses parts. Indeed, a drawback of standard scenario formalisms is that the number of participants in a given interaction cannot be parameterized. With a fixed number of objects in an interaction, it is hard to describe behaviors of systems with dynamic architectures. Message Sequence Charts propose instance creation, but the mechanism proposed in the Z.120 standard needs to know

a priori the instances that will be created. This problem will be addressed in the next chapter, where we propose a notion of dynamic MSCs handling dynamically an arbitrary number of instances. However, we will show that fibered product can be efficiently used to define and assemble parametric models that depict redundant functionalities of a system.

The fibered product of HMSCs proposed in this section builds a new HMSC from two HMSC operands. Hence, our product is endogenous. The product described needs to clearly specify common and identical parts in composed scenarios. It first identifies pairs of transitions that must be "synchronized" in two HMSCs, and realizes their union with an amalgamated sum of MSCs labeling the synchronized transitions. Common parts and synchronizations are defined though the notions of MSC and HMSC morphisms, that guarantee that a product of two HMSCs is unique, and is the smallest model to contain both views.

The benefits of scenario merging do not only concern view composition. In fact, an endogenous merge operator can be used to propose formal and well founded model transformations and design patterns for scenarios. In this section, we first defined the notion of fibered product of HMSCs, and we then show the usefulness of this construction on a concrete application, ie, the introduction of a consensus algorithm to localize choices in a HMSC.

5.1 Amalgamated Sum of MSCs

In the sequel, our formal definition of MSC sums will be based on the notions of preorder : *preorder* on a set of elements E is a relation $R \subseteq E^2$ that is reflexive (ie. $\forall e \in E, eRe$) and transitive (ie. $\forall e, f, g \in E, eRf \wedge fRg \Longrightarrow eRg$). A *partial order* is an antisymmetric preorder (ie. $\forall a, b \in E, aRb \wedge bRa \Longrightarrow a = b$).

The standard composition operations for MSCs are usually parallel or sequential composition, iteration or choice. Other operations on MSCs have been proposed, such as instance refinement [103], message refinement [46], virtuality [128], or projections [59] (also addressed in section 3 of this chapter). As shown in former section, when two MSCs depict different viewpoints of the same behavior, the intended designed behavior is some kind of merged MSC that contains both scenarios without creating copies of similar elements. In previous section, such product was defined as a mixed product of MSCs, and also as a monitored product (a mixed product in which similar events of the composed MSC are necessarily on a designated process). These operators assume that all events with common labels should have compatible orderings in the composed scenarios. We propose another merge operator for MSCs called *amalgamated sum*. Amalgamated sum is a MSC product in which common events are explicitly defined. This allows to define a merge of two scenarios containing two occurrences of the same event, each one coming from one operand. This amalgamated sum uses basic concepts of category theory. First, we need to define the notion of MSC morphisms, that will be essential to specify common parts in scenarios.

Definition 52 (MSC Morphism) *An* instance set morphism *is an injective mapping $l : I \to I'$ from an instance set I to another instance set I'. Let I and I' be two finite sets of instances and $l : I \to I'$ an instance set morphism. A* MSC morphism *along l, from $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \phi)$, a MSC over I, to $M' = (E', (<'_p$*

$)_{p\in\mathcal{P}}, \alpha', \mu', \phi')$, *MSC over* I', *is a pair of mappings* $\gamma = (\gamma_1, \gamma_2)$ *where* $\gamma_1 : E \to E'$ *is injective,* $\gamma_2 : A \to A'$ *is a renaming mapping, and:*

i) $\forall (e, f) \in E^2, e \leq f \Rightarrow \gamma_1(e) \leq' \gamma_1(f)$

ii) $\forall (e, f) \in E^2, f = \mu(e) \Rightarrow \gamma_1(f) = \mu'(\gamma_1(e))$

iii) $l \circ \phi = \phi' \circ \gamma_1$

iv) $\gamma_2 \circ \alpha = \alpha' \circ \gamma_1$

Intuitively, *i*) means that morphisms preserve ordering, *ii*) means that morphisms preserve messages. Property *iii*) means that the event morphisms is consistent with the instance morphism. It also means that all events located on a single instance of M are sent by γ_1 on a single instance of M'. Last, property *iv*) means that the event morphisms γ_1 is consistent with the labellings of M and M' and the label morphism γ_2. Note that is many sums that will be defined hereafter, the labeling morphism will be the identity morphism *id*. When no instance set morphism is specified, MSC morphisms are defined by triples $\gamma = (l, \gamma_1, \gamma_2)$ such that (γ_1, γ_2) is a MSC morphism along l.

Figure 5.11-b shows an example of MSC morphism $f =< f_0, f_1, f_2 >$ from a MSC M to a MSC M'. For the sake of clarity, we have represented f_0 as a plain line from an instance to another, f_1 as dotted lines from events of M to events of M', but f_2 is omitted. In this example, f_0 send instance A onto instance A, but instance B is sent onto B'. However, event localization is respected. Similarly, message $m2$ in M is sent onto a message n in M'.

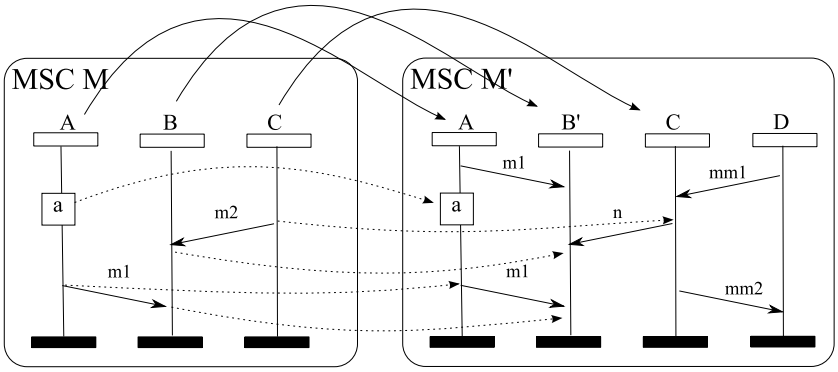


Figure 5.11: MSC morphism example

Definition 53 (Amalgamated Sum of Two Sets) *Let* I, J *and* K *be three finite sets. Let* $f : I \to J$ *and* $g : I \to K$ *be two injective maps. The amalgamated sum* $J\,{}_f +_g K$ *is defined as* $J\,{}_f +_g K = \big(J \backslash f(I)\big) \biguplus \big(K \backslash g(I)\big) \biguplus I$. *The amalgamated sum yields two injections* $\tilde{f} : J \to J\,{}_f +_g K$ *and* $\tilde{g} : K \to J\,{}_f +_g K$ *defined as follows:*

$$\begin{cases} \forall i \in f(I), & \tilde{f}(i) = f^{-1}(i) \\ \forall i \in J \setminus f(I), & \tilde{f}(i) = i \end{cases} \qquad \begin{cases} \forall i \in g(I), & \tilde{g}(i) = g^{-1}(i) \\ \forall i \in K \setminus g(I), & \tilde{g}(i) = i \end{cases}$$

Note that as we use $\uplus$ (disjoint union) in our definition, the result of an amalgamated sum can contain several copies of similar elements. Amalgamated sums of sets will be used to amalgamate sets of instances, events or actions of two MSCs to be composed. This mechanism allows for the definition of a product where events with similar labels in two operands can be nevertheless duplicated.

Definition 54 (Amalgamated Sum of two MSCs) *Let* $M_0 = (E_0, (<_{0,p})_{p\in\mathcal{I}_0}, \alpha_0, \mu_0, \phi_0)$ $M_1 = (E_1, (<_{1,p})_{p\in\mathcal{I}_1}, \alpha_1, \mu_1, \phi_1)$ $M_2 = (E_2, (<_{2,p})_{p\in\mathcal{I}_2}, \alpha_2, \mu_2, \phi_2)$ *be three MSCs respectively defined over three sets of instances* I_0, I_1, I_2*, and* $f = (f_0, f_1, f_2)$ *be a MSC morphism from* M_0 *to* M_1*,* $g = (g_0, g_1, g_2)$ *be a MSC morphism from* M_0 *to* M_2*. The* amalgamated sum *of* M_1 *and* M_2 *wrt.* f *and* g *is denoted* $M = M_1 \,{}_f{+}_g\, M_2$*, and is a tuple* $M = (E, \leq, \lambda, \mu, \phi)$ *where:*

- $I = I_1 \,{}_{f_0}{+}_{g_0}\, I_2$; $\quad E = E_1 \,{}_{f_1}{+}_{g_1}\, E_2$; $\quad A = A_1 \,{}_{f_2}{+}_{g_2}\, A_2$;
- *Preorder relation* $\leq$ *is the transitive closure of* $\tilde{f_1}(\leq_1) \cup \tilde{g_1}(\leq_2)$;
- $\forall e \in E, \lambda(e) = \begin{cases} \lambda_1(e) \text{ if } e \in E_1 \backslash f_1(E_0) \\ \lambda_2(e) \text{ if } e \in E_2 \backslash f_2(E_0) \\ \lambda_0(e) \text{ otherwise} \end{cases}$, $\quad \phi(e) = \begin{cases} \phi_1(e) \text{ if } e \in E_1 \backslash f_1(E_0) \\ \phi_2(e) \text{ if } e \in E_2 \backslash f_2(E_0) \\ \phi_0(e) \text{ otherwise} \end{cases}$
- $\mu = \tilde{f_1}(\mu_1) \cup \tilde{g_1}(\mu_2)$.

The MSC M_0 *is called the* interface *of the amalgamated sum* $M_1 \,{}_f{+}_g\, M_2$.

Note that $M = M_1 \,{}_f{+}_g\, M_2$ is not always an MSC, as $\leq$ need not be a partial order. Indeed, $\tilde{f_1}(\leq_1) \cup \tilde{g_1}(\leq_2)$ can be a preorder (i.e it can contain cycles) if M_1 and M_2 disagree on the respective order of two events e, f that belong to the interface. When $\leq$ is a partial order we will say that the amalgamated sum $M_1 \,{}_f{+}_g\, M_2$ is *well-formed*. Obviously, checking well-formedness of an amalgamated sum of MSC can be done in $O(E^2)$. If we relax the assumption that events along a process are totally ordered (that is if we allow coregions in MSCs), well-formed sums can be considered as MSCs, as one can compute the relations $(<_p)_\wp \in I$ by setting $<_p = \{(e, f) \in \leq |\ \phi(e) = \phi(f) = p\}$.

Let us illustrate the use of amalgamated sum on the example of Figure 5.12. Considering MSCs $M_1 = (E_1, (<_{1,p})_{p\in\mathcal{I}_1}, \alpha_1, \mu_1, \phi_1)$ and $M_2 = (E_2, (<_{2,p})_{p\in\mathcal{I}_2}, \alpha_2, \mu_2, \phi_2)$ as two partial observations of the same system, we want to produce a behavior that contains M_1 and M_2. Let us also suppose that even if M_1 and M_2 have different instance sets, instance X in M_2 and instance *sender* in M_1 (resp. Y and *medium*) represent the same object in the system. Intuitively, merging M_1 and M_2 then amounts to inserting an atomic action between *send* reception and *return* sending in M_1, and renaming the instances. Formally, the merge consists in the definition of an interface that identifies the common elements (events, action name, and instances) in M_1 and M_2 and renames them. For our example, this is done using a new MSC $M_0 = (E_0, (<_{0,p})_{p\in\mathcal{I}_0}, \alpha_0, \mu_0, \phi_0)$, and two MSC morphisms $f : M_0 \to M_1$ and $g : M_0 \to M_2$ defined below.

- Morphism $f = (f_0, f_1, f_2)$ from M_0 to M_1 is a triple where:
  - $f_0 : I_0 \to I_1$ is the identity,

  - $f_1 : E_0 \to E_1$ sends respectively $e1, e2, e3, e4$ onto $d1, d2, d3, d4$,
  - $f_2 : A_0 \to A_1$ is the identity.

- Morphism $g = (g_0, g_1, g_2)$ from M_0 to M_2 is a triple where:
  - $g_0 : I_0 \to I_2$ sends *sender* onto X and *medium* onto Y,
  - $g_1 : E_0 \to E_2$ sends respectively $e1, e2, e3, e4$ onto $r1, r2, r3, r4$,
  - $g_2 : A_0 to A_2$ sends respectively $Sender!Medium(send)$,$Medium?Sender(send)$, $Medium!Sender(return$,$Sender?Medium(return)$ onto $X!Y(m1)$, $Y?X(m1)$, $Y!X(m2)$, $X?Y(m2)$.

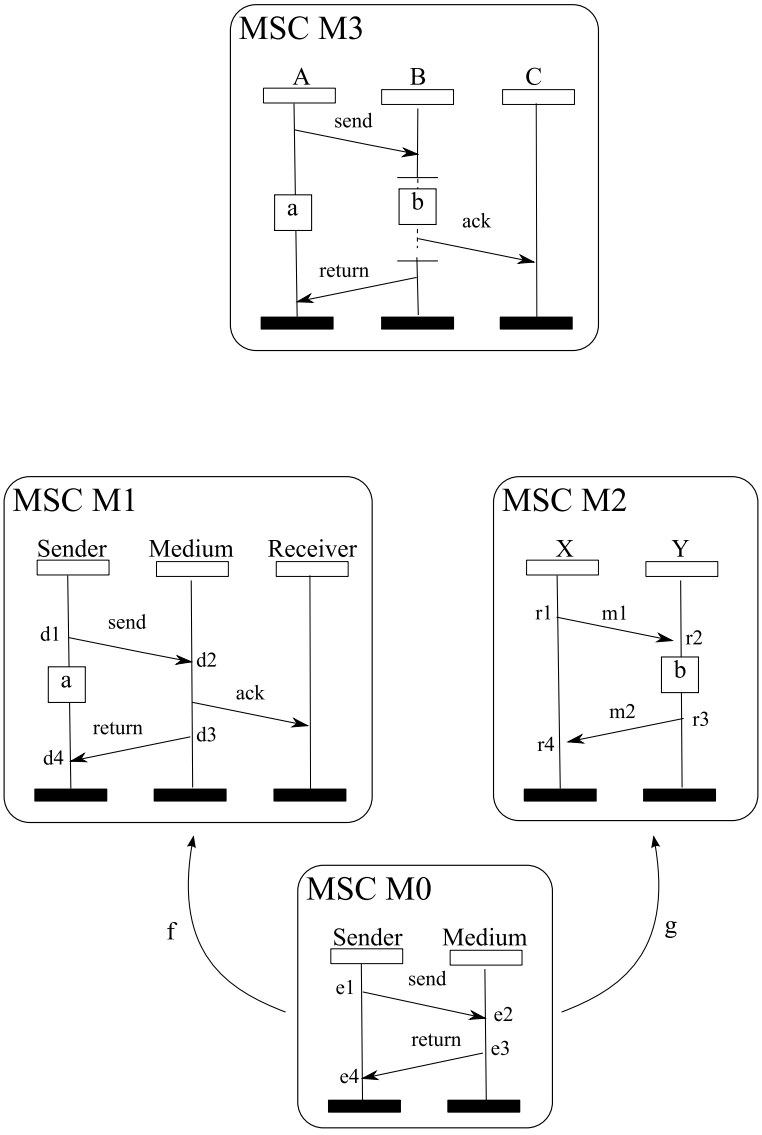


Figure 5.12: An example of amalgamated sum

The result of the amalgamated sum $M_1 \,{}_f{+}_g\, M_2$ is the MSC M_3 of Figure 5.12. Note that the names of the resulting instances on common parts are defined by the instance names of the interface. Other conventions are possible, such as defining

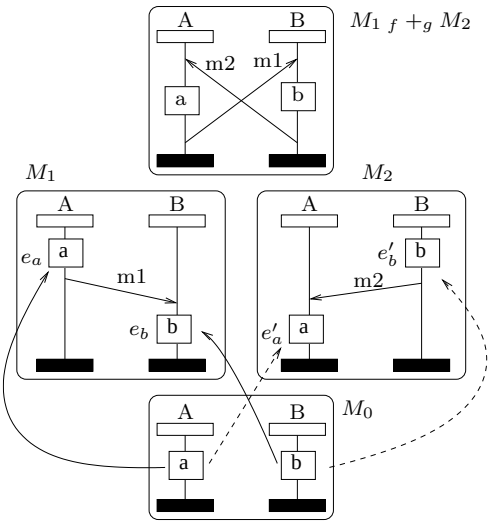


Figure 5.13: An amalgamated sum that is not well-formed

new instance names of the form "$X + Sender$" instead of keeping $Sender$ in the amalgamated sum.

As already mentioned, the $\leq$ relation in an amalgamated sum of two MSCs is not always well-formed, as the least preorder containing $\leq_1$ and $\leq_2$ may not be antisymmetric. Let us consider an example of amalgamated sum that is not well-formed, as in Figure 5.13. MSC M_1 imposes that $e_a \leq e_b$ while MSC M_2 states that $e'_b \leq e'_a$. Clearly, if e_a and e'_a are images of a single event a in the interface M_0 via f and g, and if if e_b and e'_b are also images of a single event b in M_0 via f and g, then the orders defined in M_1 and M_2 are contradictory, and summing them creates a symmetry. In such case, the two scenarios are considered incompatible, at least with the proposed morphisms f and g. In this example, we need to remove only one event (either a or b) from the interface to obtain a well-formed sum.

The result of the amalgamated sum highly depends on the interface and on how it is mapped onto composed MSCs by morphisms. Hence, there can be several way to assemble a pair of MSCs. Event if this number is finite, it can grow very fast. If we want to build interfaces automatically, we need to focus on elements with identical labels in both operands. An algorithm to identify the largest interfaces allowing an amalgamated sum was proposed in [69]. However, assembling two views with distinct alphabets describing similar concepts will need human interaction to specify a non-trivial interface.

5.2 Fibered product of HMSCs

As for standard products, it is not sufficient to assemble finite MSCs to design interesting models, and we want to extend the amalgamated sum approach to HMSCs, or to languages of MSCs generated by HMSCs. An obvious and immediate idea is to work on the set of MSCs generated by a HMSC, that is define an amalgamated sum of MSC languages. The composition of two sets of MSCs would be the set of well-formed amalgamated sums obtained by merging pairs of MSCs from each set. However, to precisely define this question, one needs to be able to define au-

tomatically an interface for two operands. If we use as interfaces any isomorphic sub-order in both operands, the obtained language will not be meaningful, as any pair of MSCs can be assembled with the empty interface. One can also rely on construction of maximal interfaces, but as an interface to assemble two MSCs need not be unique, this composition is also likely to bring an exponential explosion in the number of generated orders. Another bothering aspect of such composition is that the resulting language is not necessarily existentially bounded, nor the language of a safe cHMSC. Consider, for example, the two HMSCs of Figure 5.14. One of them generates the MSC language $\mathcal{L}_1 = \{M_1 \circ M_2^k \circ M3 \mid k \in \mathbb{N}\}$, while the other only generates $\mathcal{L}_2 = \{M\}$. As any MSC in $\mathcal{L}_1$ contains only one occurrence of a and b, an intuitive interpretation for the sum of both languages is the language $\mathcal{L}_3 = \{N\ {}_{f_N}+_{g_M}\ M\}$, where f_N (reps g_M) is a mapping from a MSC that contains only two atomic actions a and b onto N (resp M), with the obvious mappings on events, instances, and labels.

The set of MSCs in $\mathcal{L}_3$ is represented in Figure 5.15. Note that there is no ordering between the receipt and sending of the message m, and the events corresponding to the messages n. However, $\mathcal{L}_3$ is not generated by a HMSC, as the messages of type m can cross an arbitrary number of messages of type n. This kind of specification can be expressed by means of Compositional Message Sequence Charts [65], or Extended compositional Message Sequence charts [92], but not with HMSCs. From this example, one can also easily design an example generating an MSC language that is not existentially bounded, i.e. can not be represented as a safe CHMSC.

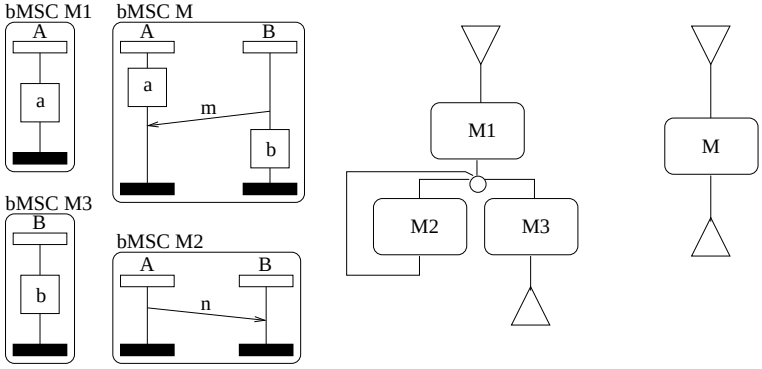


Figure 5.14: Event by event matching

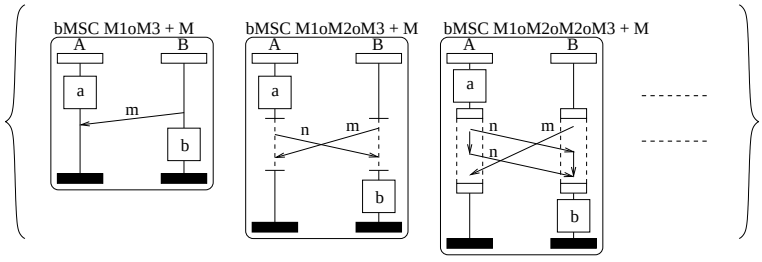


Figure 5.15: Result

Hence, languages of HMSCs and even languages of safe CHMSCs are not closed under amalgamated sum. This is not so surprising, as amalgamated sum of MSC

languages is another way to define mixed products of HMSCs. Keeping in mind the objective of defining a product such that HMSCs are closed under this operation, we propose hereafter a weaker operator, namely the fibered product of HMSCs. This syntactic operator is based on the fibered product of asynchronous transition systems proposed in [20].

The fibered product of HMSCs is defined through two composition mechanisms. First, transitions of the two support automata are partially synchronized. Then, the amalgamated sums of the MSCs attached to the transitions being synchronized are computed to create new MSCs. As for the amalgamated sum, the formal definition of the fibered product of HMSCs relies on a notion of morphism.

As for MSCs, we can define a notion of morphism for HMSCs. HMSC morphisms can be defined, as triples of morphisms or mappings: *i*) a morphism of instance sets, *ii*) a morphism of labeled transition systems and *iii*) a mapping that associates MSC morphisms to transitions.

Definition 55 (HMSC morphism) *Let $H_1 = (N_1, \longrightarrow_1, \mathcal{M}_1, n_0^1, F_1)$, $H_2 = (N_2, \longrightarrow_2, \mathcal{M}_2, n_0^2, F_2)$, be two HMSCs, respectively defined over sets of instances I_1,and I_2. A HMSC morphism f from H_1 to H_2 is a quadruple $f = (f_0, f_1, f_2, f_3)$, where*

- *$f_0 : I_2 \to I_1$ is an instance set morphism,*
- *$f_1 : N_1 \to N_2$ is a total function from nodes of H_1 to nodes of H_2, and $f_2 : T_1 \rightharpoonup T_2$ is a partial function from transitions of H_1 to transitions of H_2 which satisfy:*

  i) $f_1(n_0^1) = n_0^2$;

  ii) $t_1 = (n, a, n')$ in $\longrightarrow_1$ and $f_2(t_1)$ defined imply $\exists b \in \Sigma_2, f_2(t_1) = (f_1(n), b, f_1(n'))$ in $\longrightarrow_2$;

  iii) $t_1 = (n, a, n')$ in T_1 and $f_2(t_1)$ undefined imply $f_1(n) = f_1(n')$ in $\mathcal{S}_2$.

  Condition i) ensures that morphisms preserve initial states. According to conditions ii) and iii), any transition $t \in T_1$ of $\mathcal{S}_1$ is mapped to a transition of $\mathcal{S}_2$ via f_2 if f_2 is defined in t. In other words, f_2 defines which transitions of H_1 have observable effects in H_2. For convenience, we can add an artificial empty transition $q \xrightarrow{M_\epsilon} q$ to each state $q \in S$. With this convention, transitions of H_1 for which f_2 is not defined are mapped to an empty transition of H_2, and f_2 becomes a total function. Hence, we can succinctly rewrite the second and third conditions as follows: if $t = (p, a, q)$ is a transition of $\longrightarrow_1$, then there exists $a' \in \Sigma_2 \cup \epsilon$ such that $f_2(t) = (f_1(p), a', f_1(q))$ is a transition of $\longrightarrow_2$.

  We will use this convention in the rest of the chapter, and hence assume that there exists a transition of the form $q \xrightarrow{M_\epsilon} q$ for every state q of a transition system.

- *f_3 maps each transition $(n_1, M, n_1') \in \longrightarrow_1$ to MSC morphisms from $M' \in \mathcal{M}_2$ to M, where M' is the MSC labeling transition $(n_2, M', n_2') = f_2((n_1, M, n_1'))$.*

We can now use this notion of HMSC morphism to define a fibered product of HMSCs. Despite the apparent complexity of HMSC morphisms and product, the intuition is rather simple. To compose two HMSCs H_1, H_2, an interface HMSC H_0 is designed. Transition of H_1 and H_2 that are sent onto the same transition of H_0 are descriptions of similar interactions: they are hence synchronized, and the resulting transition is labeled by a sum of the MSCs labeling the synchronizes MSC labels. In the rest of the section, we will consider that the obtained sums are always well-formed. Note that the synchronization of HMSC transitions can be seen as a partially synchronized product of labeled transition [16] (in which synchronizations are defined using synchronization vectors).

Definition 56 (Fibered product of HMSCs) *Let H_0, H_1 and H_2 be three HMSCs, and let $f = (f_0, f_1, f_2, f_3) : H_1 \to H_0$ and $g = (g_0, g_1, g_2, g_3) : H_2 \to H_0$ be two HMSC morphisms. The fibered product of H_1 and H_2 over f and g is noted $H_1{}_f\times_g H_2$, and is the HMSC $H_1{}_f\times_g H_2 = (N, \longrightarrow, \mathcal{M}, n_0, F)$, defined over instances $I = I_1{}_{f_0} +_{g_0} I_2$, and where:*

- $N = N_1 \times N_2$, $n_0 = (n_0^1, n_0^2), F = F_1 \times F_2$
- $\longrightarrow = \{\big((n_1, n_2), M, (n_1', n_2')\big) \mid \exists t_1 = (n_1, M_1, n_1') \in \longrightarrow_2, t_2 = (n_2, M_2, n_2') \in \longrightarrow_2 , f_2(t_1) = g_2(t_2) \wedge M = M_1{}_{f_3(t_1)} +_{g_3(t_2)} M_2\}$
- $\mathcal{M} = \{M_1{}_{f_2(t_1)} +_{g_2(t_2)} M_2 \mid \exists t_1 = (n_1, M_1, n_1') \in \longrightarrow_2, t_2 = (n_2, M_2, n_2') \in \longrightarrow_2 , f_1(t_1) = g_2(t_2)$.

Roughly speaking, a fibered product of H_1 and H_2 contains one transition for each pair of transitions (t_1, t_2) of $\longrightarrow_1 \times \longrightarrow_2$ that are sent onto the same transitions $t_0 = (n_0, M_0, n_0')$ of $\longrightarrow_0$. This transition is labeled by the amalgamated sum of MSCs labeling t_1 and t_2, which uses as interface MSC M_0. The morphisms from M_0 to labels of t_1 and t_2 are provided by $f_3(t_1)$ and $g_3(t_2)$.

Let us illustrate our fibered product of HMSCs on a simple example. Figure 5.16 shows three HMSCs H_0,H_1 and H_2, with silent transitions, labeled by an empty MSC M_ϵ. Dotted arrows from H_1 to H_0 depicts a morphism f_2 from transitions of H_1 to transitions of H_0, and a morphism g_2 from transitions of H_1 to transitions of H_0. Amalgamated sums that use empty interfaces are simply unions of operands (events sets, and ordering relations), and for every M, and every pair of morphisms f, g, we have $M = M_{\ f} +_g M_\epsilon$. Letting M_1, M_2, M_0, f and g be the MSCs of example 5.12, the fibered product obtained is the HMSC H_3. Note that in H_3, transitions labeled by the empty MSC are not represented. Note also that node (n_0, n_1') is not final and that no final node can be reached from it. Hence, the transition $\big((n_0, n_1'), M6, (n_0, n_1')\big)$ can be removed from the model without changing its semantics (no MSC in $\mathcal{F}_{H_3}$ starts with $M6$).

HMSCs are closed under fibered product. Their size grow polynomially, but their existential bounds grow linearly.

Proposition 10 *Let H_1, H_2 be two HMSCs, with n_1 nodes (resp n_2) m_1 transitions (resp m_2 and existential bound b_1 (resp b_2), and with MSCs of size at most s_1 (resp s_2). Then, form any HMSC morphisms and interface, the fibered product of H_1 and H_2 is an HMSC with at most $n_1.n_2$ nodes, $m_1.m_2$ transitions, labeled by MSCs of size at most $s_1 + s_2$ and an existential bound of $\frac{s_1+s_2}{2}$.*

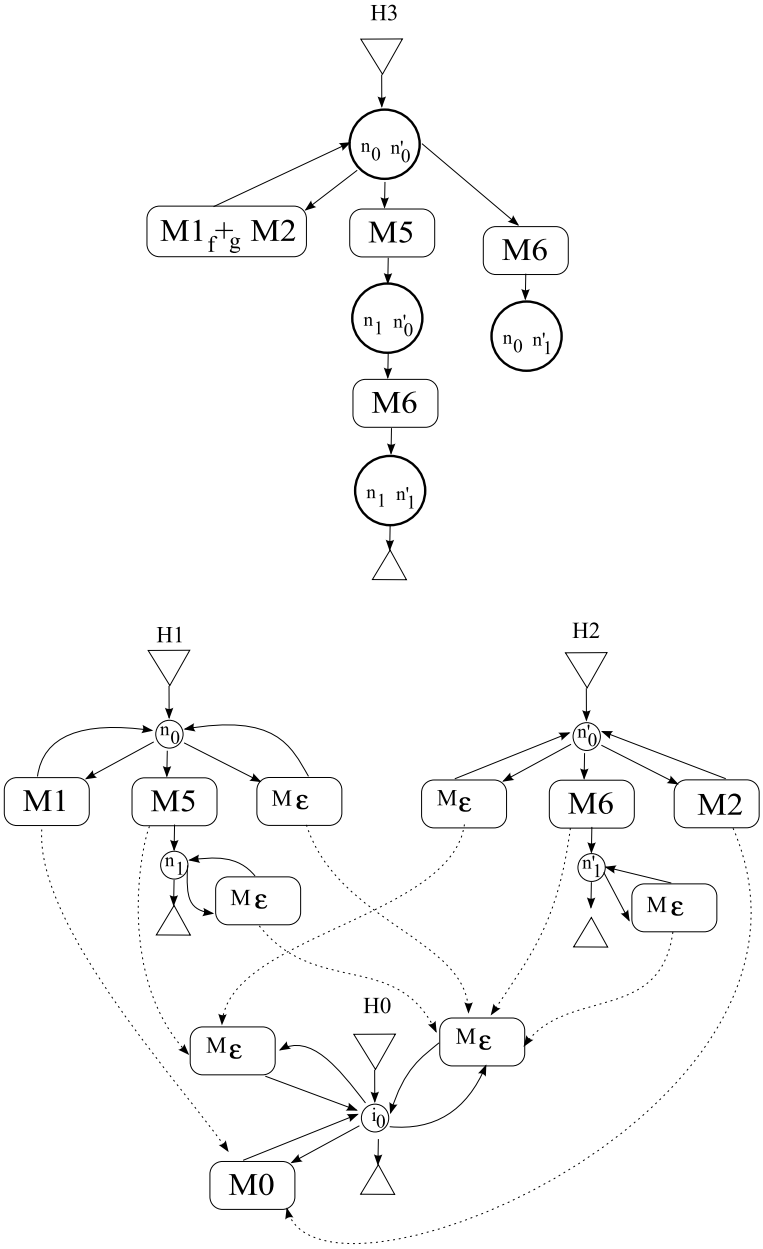


Figure 5.16: an example of fibered product of HMSC

5.3 Applications of fibered product

The fibered product should be considered as a syntactic operator, as it merges transitions of two HMSCs H_1, H_2 to create a new model. Hence, it considered the way MSC in $\mathcal{F}_{H_1}$ and in $\mathcal{F}_{H_2}$ are produced. Another inconvenient of the operator is that an interface for the composed HMSCs have to be defined, which can be a tedious task: one have to specify how MSCs in H_1 and H_2 synchronize, and also to define the MSC morphisms from the MSCs in the interface to the MSCs in the operand. These drawbacks are clearly limiting the use of mixed product, but proposition 10 shows that product allows to master the size of specifications, and preserves boundedness. So even if mixed product is not a very powerful operator and can be tedious to use, it has some nice properties. We show below that this product can also have nice practical applications for model design.

Non-local choices suppression

The first application we consider for mixed product is localization of HMSCs. As already mentioned in chapter 2, local choice can be a problem to implement a specification given as an HMSC. Considering an HMSC as an abstraction of a more complex systems in which processes unambiguously decide which scenario should be played, it seems natural to build a new HMSC in which this choice is made explicit. Making a distributed choice explicit consists in adding messages in such a way that the obtained HMSC becomes local. As we will show in chapter 7, all local choice HMSCs can be implemented. To implement a system described by a non-local HMSC, an experimented programmer would certainly add some well known distributed consensus protocol to the original HMSC. We can use a fibered product of HMSCs to integrate automatically a consensus protocol in a non-local HMSC. An interesting property of this HMSC transformation is that it can be used for an arbitrary number of instances participating to a non-local choice.

Let us consider the non-local HMSC $H1$ of figure 5.17. Let $H2$ be a protocol that involves processes A, B plus an additional supervisor, which role is to query A and B for the next scenario to execute. The chosen scenario depends on the first answer returned by process A or B. The morphisms from $H1$ to an interface H_0 and from H_2 to H_0 are depicted as dotted arrows. The fibered product of H_1 and H_2 with interface H_0 produces HMSC H_3. In this new HMSC, if the first returned answer is $Choice(1)$, either by A (MSC C1) or by B (MSC C2), then the next scenario to perform is $M1$. Similarly, if the first returned answer is $Choice(2)$, either by A (MSC $C3$) or by B (MSC $C4$), then the next scenario to perform is $M2$. One can notice that $H3$ is a local-choice HMSC. Hence, the local choice problem was solved by insertion of a given protocol before M_1 and M_2.

Insertion of a HMSC into another one can hence be seen as a fibered product operation. Admittedly, this product is not trivial, and designing an interface for an insertion took as much time as designing H_3 from scratch. However, such insertion scheme can be automated and provided as a high-level instruction, and the interface can also be computed automatically.

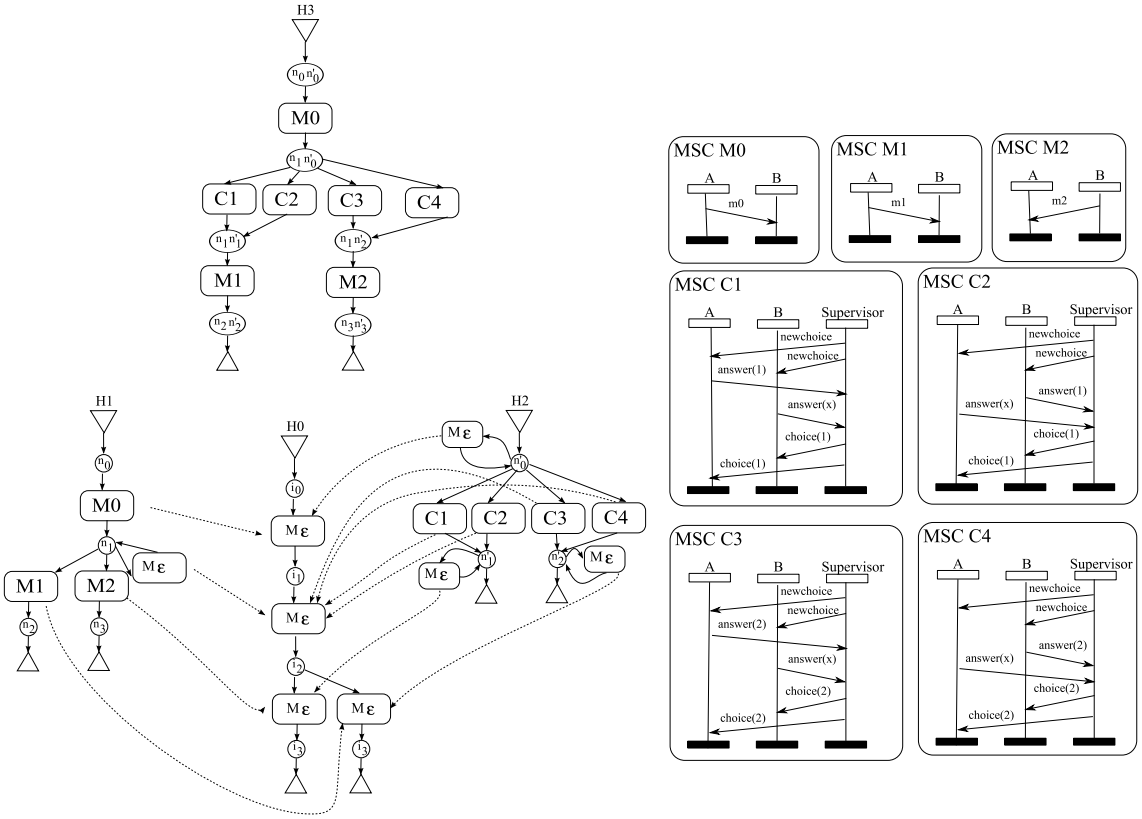


Figure 5.17: localization of a HMSC using a fibered product

Renaming and extension

The following examples illustrate two interesting features of fibered product. Very often in communication protocols, processes have symmetric roles, and exhibit isomorphic behaviors up to some renaming. Amalgamated sum allows for renaming of processes in a MSC. Consider for instance the example of Figure 5.18. MSC $M1$ depicts a simple interaction where A sends a message m to B. If we define a MSC M_0 over A, B with no events, and MSC morphisms $f =< f0, \perp, \perp >$, (where $f_0(A) = B, f_0(B) = A$, and $\perp$ denotes an empty morphism) from M_0 to M_1 and $g =< id, \perp, \perp >$ from M_0 to M_0, then we have $M_2 = M_1 \,{}_f +_g M_0$. In MSC M_2 the roles of A and B have been exchanged. We can use similar trick to rename messages, or to instantiate a protocol for a given set of processes out of a generic one. For instance, MSCs $C1, C2, C3, C4$ in Figure 5.17 can be generated from the same MSC pattern, simply by assigning instance names to roles defined in a generic behavior.

Of course, renaming does not need the full mechanism of MSC morphism, and could be achieved a more simple way. However, is is an element that can be used to define and instantiate a generic behavior over an arbitrary number of processes. Consider for instance the MSC $M1$ of Figure 5.19. This MSC can be composed with M_2, assuming a common instance $Sender$. Note that $M2$ can be obtained from $M1$ via a renaming. The amalgamated sum of $M1$ and $M2$ is the MSC $M3$, that depicts a scenario where process $Sender$ sends the same message to two processes $R1$ and $R2$ (no ordering is specified among sendings). Repeating this operation k times, one can obtain a scenario modeling a broadcast communication for a group of processes,

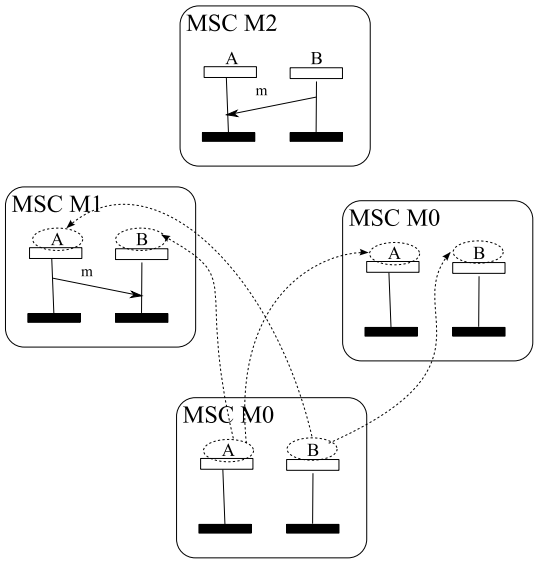


Figure 5.18: Exchanging roles with an amalgamated sum

starting only from a single generic MSC ($M1$) and a description of the composition of the broadcast group.

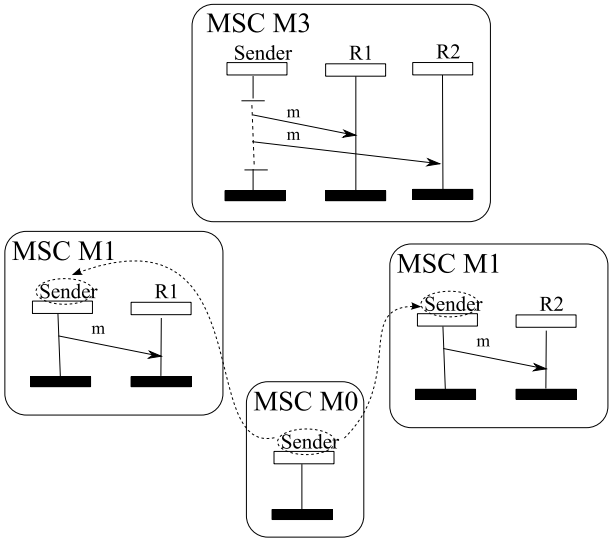


Figure 5.19

All these examples of fibered product and sums show that these operators can be useful to build models of systems from generic behavioral parameterized pieces, and a description of the systems components. This is frequently called self-modeling, and may be an appropriate answer to the lack of modeling during the development of real systems. Of course, it seems illusory to ask a designer to define MSC and HMSC morphisms. However, fibered product can be used as a formal ingredient to define high-level operators, defined in terms of sums or products, and which morphisms can be automatically computed, or defined with little help from end-users. The renaming and broadcast examples illustrate the kind of high-level operator that

could be provided this way.

5.4 Conclusion on fibered product

The fibered product described in this section is an answer to the undecidability of mixed product, and to the closure properties of monitored product. Indeed, as soon as events types are respected by MSC morphisms (sending events, reception events and atomic actions are respectively mapped to sending events, reception events and atomic actions of the target MSC) well-formed sums are MSCs. Furthermore, well-formedness is decidable. Similarly, fibered products of HMSCs are still HMSCs.

The definition of fibered product mainly relies on the definition of appropriate morphisms, which can be considered rather complicated at first sight. Furthermore, morphisms force some synchronization between MSCs, which can be considered as an arbitrary solution (remember that due to the meaning of sequential composition, there are several ways to obtain similar orderings between events, and hence that decomposition of scenarios into MSCs is not always meaningful). A clue to refine merging is to define HMSC morphisms not only as transition morphisms but rather as path morphisms, as is done in [20]. Note also that amalgamated sums define explicitly events that coincide in both views. This can be considered as a drawback, but relying on events labeling to detect similarities and assemble MSC languages immediately leads to compositions such as the mixed product, and to undecidability of important properties in a modular design framework. For instance, it seems essential to have procedures to detect whether a product of (c)HMSCs has an empty language or not. Furthermore, for most cases where fibered product can be applied, the morphisms are rather simple. They can often be computed automatically, and more elaborated fibered products can be provided as higher-level constructs, as shown for the localization example. Clearly, fibered product can be used as a design tool to create automatically large scenario specifications from a finite set of generic behavioral patterns, and some information on processes of a system. scenarios. These concrete scenarios could then be used for testing or diagnosis purposes.

6 Conclusion

Before concluding this chapter, it seems interesting to summarize properties of different HMSC operators considered in this chapter, and to recall the properties that they preserve. Among the considered properties, we are of course interested in global cooperation, locality of choices, regularity, divergence, and whether a composition of HMSCs remains finitely generated, existentially or universally bounded. For mixed and monitored product, considering global cooperation, locality, regularity and so on only makes sense in contexts where the composition of two HMSCs produces an MSC language that is still finitely generated, and preserve in some way the structure of the original HMSC. This is not always the case, as we have already seen.

The table 5.1 below summarizes the properties preserved by projection. For simplicity, and uniformity of the conclusion, we only consider as input HMSCs. However, safe CHMSCs usually enjoy the same properties as HMSCs, except of course for finite generation. The left column lists possible properties of a HMSC H, and the right column the corresponding property of its projection.

Properties of HMSC H	Properties of $\Pi(H)$
Local	local or non local
non Local	local or non local
regular	regular
non regular	regular or non regular
globally cooperative	g.coop or non-g.coop
non globally cooperative	g.coop or non-g.coop
divergent	divergent or non divergent
non divergent	non divergent
$\mathcal{L}(H)$ regular	$\mathcal{L}(\Pi(H))$ regular
$\mathcal{L}(H)$ not regular	$\mathcal{L}(\Pi(H))$ regular or non regular
Univ. bounded	univ. bounded
non Univ. bounded	univ.bounded or non univ.bounded
exist. bounded (by definition)	exist. bounded ($\Pi(H)$ is a safe cHMSC)
finitely generated (by definition)	fin.gen or non fin. gen

Table 5.1: Properties of HMSC projections

Furthermore, we can recall that it is decidable whether the projection of a HMSC is finitely generated, in which case it can be represented as another HMSC.

The table below summarizes the properties of HMSC products : mixed product $\|$, monitored product ($|||$), and fibered product $\times$.

$H1, H2$	$H_1 \parallel H_2$	$H_1 \mid\mid\mid H_2$	$H_1 \times H_2$
Local	local or non local	local or non local	local or non local
non Local	local or non local	local or non local	local or non local
regular	regular	regular	regular or non regular
non regular	regular or non regular	regular or non regular	regular or non regular
glob. cooperative	glob. coop. (when a model exists)	glob. coop. (when a model exists)	glob. coop or non glob. coop
non glob. cooperative	g.coop or non g.coop	g.coop or non g.coop	g.coop or non g.coop
divergent	divergent or non divergent	divergent or non divergent	divergent or non divergent
non divergent	non divergent	non divergent	non divergent
$\mathcal{L}(H)$ regular	$\mathcal{L}(H_1 \parallel H_2)$ regular	$\mathcal{L}(H_1 \mid\mid\mid H_2)$ regular	$\mathcal{L}(H_1 \times H_2)$ regular or non regular
$\mathcal{L}(H)$ not regular	$\mathcal{L}(H_1 \parallel H_2)$ regular or non regular	$\mathcal{L}(H_1 \mid\mid\mid H_2)$ regular or non regular	$\mathcal{L}(H_1 \times H_2)$ regular or non regular
$\forall$-bounded	$\forall$-bounded	$\forall$-bounded	$\forall$-bounded
non $\forall$-bounded	$\forall$-bounded or non $\forall$-bounded	$\forall$-bounded or non $\forall$-bounded	$\forall$-bounded or non $\forall$-bounded
$\exists$-bounded	$\exists$-bounded or non $\exists$-bounded	$\exists$-bounded or non $\exists$-bounded	$\exists$-bounded ($\frac{s1+s2}{2}$)
finitely generated	fin.gen or non fin. gen.	fin.gen or non fin. gen.	fin.gen

Table 5.2: Porperties of Products

We furthermore recall that, given two HMSCs H_1, H_2, it is undecidable whether $\mathcal{L}(H_1 \parallel H_2) = \emptyset$. This property is important, as it means that composition of H_1 and H_2 still has some meaning (it contains at least one behavior). One can not decide either if $\mathcal{L}(H_1 \parallel H_2)$ is existentially bounded. Moving to the more restricted setting

of monitored product, existential boundedness is decidable for safe cHMSCs (and hence for HMSCs), and emptiness is decidable for HMSCs. Mixed and monitored product do not necessarily preserve existential boundedness. For fibered products, emptiness may arise due to the fact that amalgamated sums are not well-formed, or if final states become unreachable after synchronization. Hence, emptiness is decidable. An existential bound is guaranteed by construction of the product (one can compute a tighter bound as the maximal existential bound over all MSC labeling the product).

At first sight, it may seem surprising that products may transform non regular or non- globally cooperative HMSCs into regular or globally cooperative products. However, one should recall that $M \in \mathcal{F}_{G_1 \| G_2}$ if it can be projected on the alphabet of G_1 and G_2 to obtain two MSCs $M_1 \in \mathcal{F}_{G_1}$ and $M_2 \in \mathcal{F}_{G_2}$. If a path ρ of G_1 can only be merged with path of G_2 that disagree on the ordering of common events that appear in M_ρ, then the behavior in path ρ simply disappears from the merged specification. Hence, mixed and monitored product can be used to select a subset of paths of a HMSC, which may change non-regular/non-local/divergent/non-globally cooperative specification to regular/local/non divergent/globally cooperative ones. Conversely, fibered product may amalgamate components from distinct processes without communications, hence MSCs labeling loops may become non (strongly) connected even if the operands were.

Last, note that though the nature of the proposed product may focus on semantics or (for monitored and mixed product) or syntax (for fibered product) of HMSCs, the differences in terms of modified properties is not obvious from the table. Non-divergence and universal bounds seem to be the only properties preserved by all operations. Monitored and mixed product preserve (assuming an equivalent (c)HMSC specification exists for the product) global cooperation, as one can only create new cycles in a product by synchronizing connected cycles of the operands. This property is usually an interesting property for model checking. However, mixed and monitored product do not preserve existential boundedness in general, which is a bad property with respect to implementation. Fibered product preserves existential boundedness, but at the cost of not preserving global cooperation. Specification that are not globally cooperative are not implementable as they are, as independent groups of processes are supposed to perform identical number of occurrences of HMSC loops without communication. If one considers a product as a way to refine a specification, loosing global cooperation during a product operation is a bad property of the operator.

Last comparing the cost of each operator in terms of the size of the built model and in terms of complexity of use, one can note that composing two specifications with mixed or monitored product does not require any additional information from end-users, but that the size of the specification built using a monitored product (and hence also a mixed product) is redhibitory. Fibered product is a way to master the size of the products. However, defining a fibered product means defining the morphisms that are used to synchronize MSCs and to amalgamated them, which is a tedious tasks. Monitored and mixed product do not require such an effort from the designer. An first attempt to build the morphisms automatically was proposed in [69]. We furthermore think that several High-level macros for HMSCs can be defined as fibered products which interfaces can be automatically computed.

Overall, none of the usual operations (projection, product) needed to design a partial order automata algebra has all desired properties. This a disappointing conclusion, but one should recall that HMSCs and their variants are models that define non-regular behaviors. So one could not expect HMSCs to exhibit the nice algebraic properties of automata. However, we believe that monitored product and fibered product may still find useful application to design large models out of a few behavioral elements, but only when the sizes of the models can be mastered in monitored product, and when morphisms for fibered composition can be computed automatically.

Chapter 6

Dynamic MSCs

La description est déjà prête, il faut seulement que vous combliez deux ou trois lacunes, pour que les choses soient en ordre; il n'y a pas d'autre but et aucun autre but ne saurait être atteint.

The record is already complete, there are only two or three omissions which you must fill in for the sake of order; There is no other object in view, and no other object can be achieved.

[Franz Kafka, Le Château(1926)]

1 Introduction

In chapters 3 and 4, we have seen two ways to extend expressiveness of partial order automata in a tractable way: splitting messages or allowing commutations to allow description of modern protocols such as sliding windows. This extension of HMSCs semantics allows for the definition of the same syntactics classes as for HMSCs, and hence allow for the definition of syntactic subclasses of the languages for which some problems (model checking, intersection vacuity, ...) are decidable. Chapter 5 has considered different ways to assemble HMSC specification, with disappointing results, as none of the compositions proposed preserve syntactic subclasses. Note that so far, we have only considered extensions of partial order automata for fixed sets of processes. MSCs were originally created in a telecommunication context, and were seen as descriptions of protocols. However, nowadays, many applications rely on threads, and most protocols are designed for an open world, where all the participating actors are not known in advance. Hence, integrating dynamic process creation in scenarios is needed to adapt the formalism to current sate of distributed systems.

This chapter is dedicated to the extension of HMSCs to allow dynamic process creation. This chapter mainly considers expressiveness issues, but realizability of dynamic scenario specifications is also considered in the chapter dedicated to implementation of scenarios

A first step towards MSCs over an evolving set of processes was made by Leucker, Madhusudan, and Mukhopadhyay [92]. Their *fork-and-join MSC grammars* allow for dynamic creation of processes and have good properties, such as decidability of MSO model checking. However, it remains unclear how to implement fork-and-join MSC grammars. In particular, a corresponding automata model with a clear

behavioral semantics based on MSCs is missing. Dynamicity of scenarios and their realizability are then two important issues that must be considered jointly.

The work described in this chapter is a joint work with B.Bollig, and was published in [26]. We introduce dynamicity in HMSCs using *dynamic MSC grammars* (DMG for short) as a specification language. It is inspired by the fork-and-join grammars from [92] but closer to an implementation. We keep the main idea of [92]: when unfolding a grammar, MSCs are concatenated on the basis of finitely many process identifiers. While, in [92], the location of identifiers can be changed by means of a very general and powerful split-operator, our grammars consider an identifier as a *pebble*, which can be moved *locally* within one single MSC. Then, rewriting rules append behaviors and new processes to the pebbled processes. The grammars proposed in this work are more "linear" than the grammars of [92], that is they assemble MSCs and produce processes from left to right along productions of a grammar, while fork-join MSC grammars allows more general recursive schemes. However, fork-join grammars were not designed to produce implementable specifications, and and implementation model was not proposed in [92]. We will show an implementation model, and consider realizability for dynamic MSCs in chapter 7.

2 MSC grammars

In order to introduce dynamic process creation in Message Sequence Charts, we need to describe behaviors without relying on process identities, but rather as interactions among processes playing a *role*.

The main principle of MSC grammars is to assemble anonymous partial orders while instantiating process names at concatenation time. To this extent, we will define MSC pieces, that contain messages, as usual, a threading mechanism that emphasizes thread creation (which can be seen as a particular *spawn* message) and birth of threads, represented as a particular *start* action.

2.1 dynamic MSCs, partial MSCs

Dynamic MSCS are simply a variant of MSCs allowing process creation. Thread creation was already a feature of the Z.120 standard, but was not addressed as a part of the language in theoretical studies before [92].

Definition 57 *A* dynamic MSC *(dMSC for short)is a tuple* $M = (\mathcal{P}, E, (<_p)_{p\in\mathcal{P}}, \mu, <_{\mathsf{s}}, \alpha, \varphi)$ *where*

(a) $\mathcal{P} \subseteq \mathbb{N}$ *is a nonempty finite set of* processes*,*

(b) $E := \bigcup_{p\in\mathcal{P}} E_p$*, where the* E_p *are disjoint nonempty finite sets of* events*,*

(c) $\alpha : E \to \{p!q(m), p?q(m), \mathsf{spawn}, \mathsf{start}\}$ *assigns a* type *to each event, and*

(d) $<_{\mathsf{p}}$*, and* μ *are binary relations on* E *with the same meaning as in MSCs (successor on the same process, message relation).*

(e) $<_{\mathsf{s}}$ *is a spawning relation. It is a binary relation on* E *that associates a spawning event (i.e. an event* e *such that* $\alpha(e) = spawn$ *with a start event (i.e. an event* f *such that* $\alpha(f) = start$*).*

As for MSCs, we require that $\leq := (<_{\mathsf{p}} \cup <_{\mathsf{s}} \cup \mu)^*$ is a partial order. The relation μ is a bijection from send events to receive events (that is, dMSCs are communication closed), and associates a send event to a receive on a different thread. For simplicity, we will also assume that dynamic MSCs are FIFO. The main change with respect to MSCs is the introduction a new event types (start and spawn), and of new constraints attached to these types. We require each start events to be the first event on a process $\alpha^{-1}(\mathsf{start}) = \{e \in E \mid \text{there is no } e' \in E \text{ such that } e' <_{\mathsf{p}} e\}$, and similarly, we require that each process has a start event as minimal event. Last, we require that $(E, \leq)$ has a unique minimal element, denoted by $start(M)$ (which is of course a start event). The relation $<_{\mathsf{s}}$ relates a spawn event $e \in E_p$ with the (unique) start action of a different thread $q \neq p$, meaning that p has created q. $<_{\mathsf{s}}$ induces a bijection between $\lambda^{-1}(\mathsf{spawn})$ and $\lambda^{-1}(\mathsf{start}) \setminus \{start(M)\}$, that is each process in a dynamic MSC was created during the depicted interaction, except for an original process that from which all other threads were created. Note that most of these requirements are notational conventions.

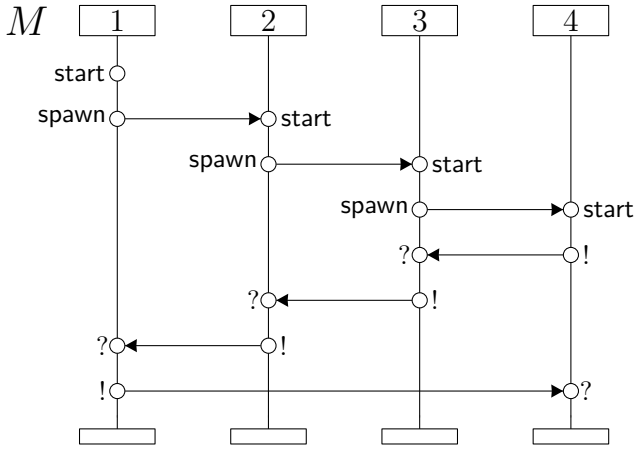


Figure 6.1: A dynamic MSC.

Figure 6.2, shows a dynamic MSC M with set of processes $\mathcal{P} = \{1, 2, 3, 4\}$. Processes 2,3, and 4 are created during the execution of M. MSC grammars that we define is this chapter will be used to assemble dMSCs out of a set of finite behavioral patterns. Though we want to design grammars which productions are dMSCs, we do not require each assembled pattern to be a dMSC, and work with *partial dMSCs*.

Definition 58 (pMSC) *Let $M = (\mathcal{P}, E, (<_p)_{p\in\mathcal{P}}, \mu, <_{\mathsf{s}}, \alpha, \varphi)$ be a dMSC and let $E' \subseteq E$ be a nonempty set satisfying $E' = \{e \in E \mid (e, \hat{e}) \in \mu \cup <_{\mathsf{s}} \cup \leq^{-1}$ for some $\hat{e} \in E'\}$ (i.e., E' is an upward-closed set containing only* complete *messages and spawning pairs). Then, the restriction of M to E' is called a* partial dMSC (PMSC)*. In particular, the new process set is $\{p \in \mathcal{P} \mid E' \cap E_p \neq \emptyset\}$. The set of* PMSCs *is denoted by $\mathbb{P}$, the set of dMSCs by $\mathbb{M}$.*

Partial MSCs suffixes of dMSCs. They do not necessarily have start events on each process. Consider Figure 6.2. It depicts the simple MSC I_p, with one start

event on process p. Moreover, M_1, M_2 are pMSCs, but not dMSCs. These patterns can not be defined over fixed process identities, as otherwise, we could not model process creation. We hence consider that pMSCs are defined over roles. Roles are defined alike processes in MSCs, but keeping in mind that they can be renamed at derivation time (i.e., when a pMSC pattern is appended during application of a rule, role are assigned an definitive identity). We will detail later how identities are managed during rewriting steps.

Defining behaviors over dynamically evolving sets of processes of arbitrary sizes has several implications. First, a process in $\mathcal{P}$ does not have the same meaning as in MSCs. A process $p \in \mathcal{P}$ should be interpreted as a way to assign a thread identity to a sequence of events. This identity is known only at runtime, so the fact that a process has identity 1,2,3, or 4 is less important than the fact that events located on process 1 and 2 (for instance) are events from distinct threads. The second implication of unbounded dynamicity is that to define behaviors, we will have to reason in terms of assembling of behavioral patterns, and possibly rename processes. Note that pMSCs are communication closed. Note also that as a suffix of a dMSC, a pMSC has at most one start event per process, and that this event should be minimal on the considered process. This is important as it means that one can not assemble any pair of patterns.

We next define a concatenation operator to append a partial MSC to a dMSC. Let $M = (\mathcal{P}, (E_p)_{p\in\mathcal{P}}, <_{\mathsf{p}}, <_{\mathsf{s}}, \mu, \lambda)$ be a PMSC. For $e \in E$, we recall that $\varphi(e)$ is the unique process $p \in \mathcal{P}$ such that $e \in E_p$. For every $p \in \mathcal{P}$, there are a unique minimal and a unique maximal event in $(E_p, \leq \cap (E_p \times E_p))$, which we denote by $\min_p(M)$ and $\max_p(M)$, respectively. We let $Proc(M) = \mathcal{P}$. By $Free(M)$, we denote the set of processes $p \in \mathcal{P}$ such that $\lambda^{-1}(\mathsf{start}) \cap E_p = \emptyset$. Finally, $Bound(M) = \mathcal{P} \setminus Free(M)$. Intuitively, free processes of a PMSC M are processes that are not initiated in M. In Figure 6.2, $Bound(\mathsf{I}_p) = \{p\}$, $Free(M_1) = \{1\}$, and $Free(M_2) = \{1, 2\}$.

Definition 59 (pMSC concatenation) *For $i = 1, 2$, let $M^i = (\mathcal{P}^i, E^i, (<_p)^i_{p\in\mathcal{P}}, \mu^i, <^i_{\mathsf{s}}, \alpha^i, \varphi^i)$ be* PMSCs. *Consider the structure $M = (\mathcal{P}, E, (<_p)_{p\in\mathcal{P}}, \mu, <_{\mathsf{s}}, \alpha, \varphi)$ where $E = E^1 \uplus E^2$ and $<_{\mathsf{p}} = <^1_{\mathsf{p}} \cup <^2_{\mathsf{p}} \cup \{(\max_p(M^1), \min_p(M^2)) \mid p \in \mathcal{P}$ with $E^1_p \neq \emptyset$ and $E^2_p \neq \emptyset\}$. The other sets and relations are defined as simple unions. If M is a* PMSC*, then the concatenation of M^1 and M^2 is defined, and denoted $M^1 \circ M^2$. We furthermore set $M^1 \circ M^2 := M$. Otherwise, $M^1 \circ M^2$ is undefined.*

Visually, concatenation of PMSCs corresponds to drawing identical processes one below the other. A concatenation of two pMSCs M_1, M_2 is undefined if a process p of M^2 contains a start event, and $E^1_p \neq \emptyset$. Let us consider the pMSCs I_p, M_1 and M_2 of figure 6.2. The concatenation $M_1 \circ M_2$ is defined, and consists of a pMSC in which process 1 spawns process 2, and process 2 then sends a message to process 1.

As mentioned in the definition, concatenation of patterns might be undefined. In particular, $M_1 \circ M_2$ is not defined if $Bound(M_1) \cap Bound(M_2) \neq \emptyset$. Consider the pattern M_1 of figure 6.2. Assembling twice this pattern produces a structure in which process 2 is created twice, and has two start events.

2.2 MSC grammars

We can now introduce *dynamic MSC grammars* (DMGs). They are inspired by the grammars from [92], but take into account that we want to implement them in terms

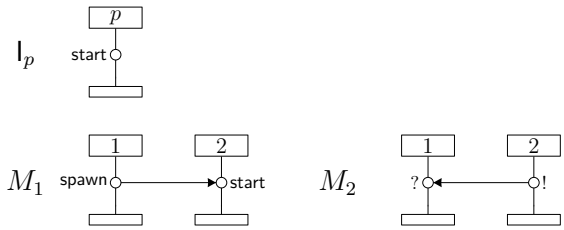


Figure 6.2: (Partial) message sequence charts

of communicating threads. This means in particular that all processes occurring in a specification will be spawned from an initial original process. Roughly speaking, MSC grammars can be seen as graph grammars that append pMSCs, and which productions are dMSCs. We keep the main idea of [92] and use process identifiers to tag active processes in a given context. Roughly speaking, MSC grammars append successively finite pieces of behaviors, and process identifiers are used to remember some processes to which such piece of behavior can be appended.

Definition 60 *Let* Π *be a nonempty and finite set of* process identifiers. *A* named MSC *over* Π *is a pair* (M, ν) *where* M *is an MSC and* $\nu : \Pi \to Proc(M)$ *assigns a process identifier to some processes of* M. *An* in-out PMSC *over* Π *is a pair* (M, μ) *where* M *is a* PMSC *and* $\mu : \Pi \rightharpoonup Free(M) \times Proc(M)$ *is a partial mapping. In the sequel, we will denote by* $\mathrm{n}\mathbb{M}$ *the set of named MSCs over* Π, *and by* $\mathrm{m}\mathbb{P}$ *the set of in-out* PMSCs.[1] *We will also let* $\mathfrak{M}$ *range over named MSCs and* $\mathcal{M}$ *over in-out* PMSCs.

Intuitively, a named MSC is an MSC M plus an assignment of process identifiers to processes appearing in M. Processes that are not assigned any identifier will not be appended any piece of behavior by future rewritings of non-terminals of the grammar. As for word grammars, MSC grammars will be used to *derive* a final production (namely a dMSC) from an initial axiom, and using a set of rules. As usual, a rule rewrites a non terminal N into an expression Exp, defined over non-terminals and named MSCs. More formally, letting $\mathcal{N}$ be a set of non-terminals, and Π be a set of process identifiers, an *expression* over $\mathcal{N}$ and Π is a sequence $expr \in (\mathrm{m}\mathbb{P} \cup \mathcal{N})^*$ of the form $u_0.(M_1, \mu_1).u_1 \ldots (M_k, \mu_k).u_k$, $k \geq 1$ and $u_i \in \mathcal{N}^*$, such that $M(expr) := M_1 \circ \ldots \circ M_k \in \mathbb{P}$. We let $Proc(expr) := Proc(M(expr))$, $Free(expr) := Free(M(expr))$, and $Bound(expr) := Bound(M(expr))$. Intuitively, free processes in the right part of a rule should have been defined before the application of the rule, and bound processes are created and assigned an identity during application of the rule.

Let us consider the rule *Rule*2 in Figure 6.3. It is the graphical representation of a rule that rewrites a non-terminal called A. This rule is of the form $(A, \mathcal{M}_1.A, f)$, where $\mathcal{M}_1$ is a named MSC, with two processes (say $\{1, 2\}$), that migrate identifier π_2 from process 1 to process 2. Mapping f is only defined for process 1, and we have $f(1) = \pi_2$, which means that process 1 is the process carrying identifier π_2 when this rule is applied. In the rest of this chapter, we will indicate mapping f by attaching a set of labels to each process line. We will define identifiers migration by labeling

[1] We omit the index Π, which will always be clear from the context.

messages or spawn relation by the label that is attached to a new process (in our example, π_2 labels the spawn relation in $\mathcal{M}_1$.

Definition 61 *A* dynamic MSC grammar *(DMG for short) is a quadruple* $G = (\Pi, \mathcal{N}, S, \longrightarrow)$ *where* Π *and* $\mathcal{N}$ *and are nonempty finite sets of* process identifiers *and* non-terminals*,* $S \in \mathcal{N}$ *is the* start non-terminal*, and* $\longrightarrow$ *is a finite set of* rules*. A rule is a triple* $r = (A, expr, f)$ *where* $A \in \mathcal{N}$ *is a non-terminal,* $expr$ *is an expression over* $\mathcal{N}$ *and* Π *with* $Free(expr) \neq \emptyset$*, and* $f : Free(expr) \to \Pi$ *is injective. We may write* r *as* $A \longrightarrow_f expr$*.*

The size of a DMG is denoted $|G|$, and is defined as $|G| = |\Pi| + \sum_{A \longrightarrow_f expr} (|expr| + |Proc(expr)|)$, where $|expr|$ denotes the length of $expr$ as a word. We also set $Proc(G) := \bigcup_{A \longrightarrow_f expr} Proc(expr)$.

Let us give an intuitive interpretation of a rule $r = (A, expr, f)$. Application of rule r replace a non-terminal A, that has to be appended to a named MSC (this rewriting mechanism is formalized hereafter). A is hence replaced by $expr$. All processes in $Bound(expr)$ should be assigned a new fresh identity, but every free process p of $expr$ is assigned an existing identity, more precisely the identity of the process carrying identifier $f(p)$.

Let us consider the example grammar of Figure 6.3. This grammar defines sets of behaviors in which an arbitrary number of processes are created sequentially (process 1 creates process 2, ..., process $n-1$ creates process n) and then the last created process sends a message m to the first process. The language of this grammar contains all similar behaviors for an arbitrary number $n > 1$ of processes.

2.3 Semantics of Dynamic MSC grammars

The semantics of a DMG $G = (\Pi, \mathcal{N}, S, \longrightarrow)$ is defined as the set of dMSCs that can be derived from the axiom S using rules in $\longrightarrow$. Applying a rule is called a rewriting. It applies on a configuration, and produces a new configuration of the grammar.

Definition 62 *A* configuration *of DMG* $G = (\Pi, \mathcal{N}, S, \longrightarrow)$ *is a pair* $(\mathfrak{M}, \beta)$ *where* $\mathfrak{M} \in \mathrm{n}\mathbb{M}$ *and* $\beta \in (\mathrm{m}\mathbb{P} \cup \mathcal{N})^*$*. If* $\beta = \varepsilon$*, then the configuration is said to be* final*. Let* $Conf_G$ *be the set of configurations of* G*. A configuration is* initial *if it is of the form* $((I_p, \nu), S)$ *for some* $p \in \mathbb{N}$*, where* I_p *is the MSC depicted in Figure 6.2 and* $\nu(\pi) = p$ *for all* $\pi \in \Pi$*.*

In a configuration $(\mathfrak{M}, \beta)$, the named MSC $\mathfrak{M}$ represents the scenario that has been executed so far, and β is a sequence of non-terminals that will be evaluated later and in-out PMSCs that will be appended later, proceeding from left to right. Note that in a configuration, process identities are fixed as soon as a pMSC appears in β.

A *derivation* of a DMG is a sequence of configurations $(\mathfrak{M}, \beta)$. A derivation of a DMG is obtained by application of rewriting rules and concatenations, following the semantics defined hereafter. A derivation is final if it reaches a final configuration. We distinguish two kinds of steps in derivations:

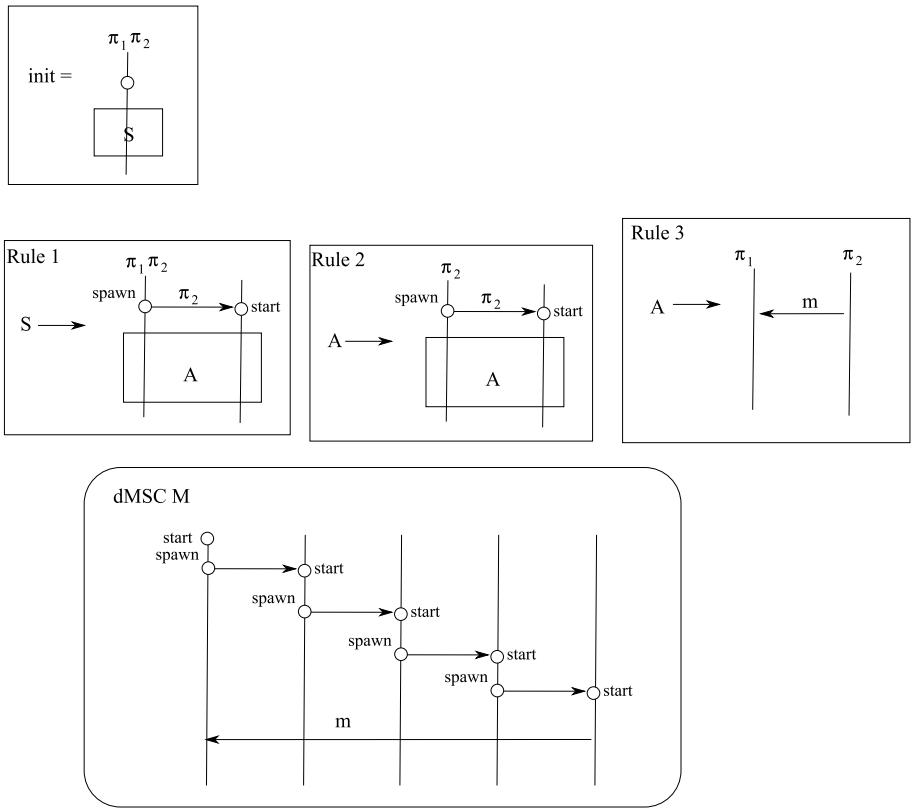


Figure 6.3: A simple MSC grammar, and one of its productions

Terminal concatenation If $\beta = \mathcal{M}.\gamma$ for some in-out PMSC $\mathcal{M}$, then the next configuration is $(\mathfrak{M} \circ \mathcal{M}, \gamma)$. However, the concatenation $\mathfrak{M} \circ \mathcal{M}$ is defined only if $\mathfrak{M}$ and $\mathcal{M}$ are compatible. Formally, we define a partial operation $\_\circ\_ : \mathrm{n}\mathbb{M} \times \mathrm{m}\mathbb{P} \rightharpoonup \mathrm{n}\mathbb{M}$ as follows: Let $(M_1, \nu_1) \in \mathrm{n}\mathbb{M}$ and $(M_2, \mu_2) \in \mathrm{m}\mathbb{P}$. Then, $(M_1, \nu_1) \circ (M_2, \mu_2)$ is defined if $M_1 \circ M_2$ is defined and contained in $\mathbb{M}$, and, for all $\pi \in \Pi$ such that $\mu_2(\pi) = (p, q)$ is defined, we have $\nu_1(\pi) = p$. If defined, we set $(M_1, \nu_1) \circ (M_2, \mu_2) := (M, \nu)$ where $M = M_1 \circ M_2$, $\nu(\pi) = \nu_1(\pi)$ if $\mu_2(\pi)$ is undefined, and $\nu(\pi) = q$ if $\mu_2(\pi) = (p, q)$ is defined. This means that a configuration $(\mathfrak{M}, \mathcal{M}.\gamma)$ may have no successor, and hence from this configuration, no sequence of rewritings and concatenations can reach a final configuration. We will call such configuration a deadlocked configuration.

Terminal concatenation allows for the definition of a relation $\overset{\mathrm{e}}{\Longrightarrow}_G$ over sets of configurations. For configurations $\mathcal{C} = (\mathfrak{M}, \mathcal{M}.\gamma)$ and $\mathcal{C}' = (\mathfrak{M}', \gamma)$, we let $\mathcal{C} \overset{\mathrm{e}}{\Longrightarrow}_G \mathcal{C}'$ if $\mathfrak{M}' = \mathfrak{M} \circ \mathcal{M}$ (in particular, $\mathfrak{M} \circ \mathcal{M}$ must be defined).

Non-terminal replacement Now consider a configuration $(\mathfrak{M}, A.\gamma)$, where A is a non-terminal. Replacing A with a sequence *expr* includes a renaming of processes to make sure that free processes appearing in *expr* that have identifier π have the same name as an existing process of $\mathfrak{M}$ identified as π. In particular, processes that occur free in *expr* take identities of processes from $\mathfrak{M}$. This is achieved by renaming processes.

A *renaming* is a bijective mapping $\sigma : \mathbb{N} \to \mathbb{N}$. For an in-out PMSC $\mathcal{M} =$

(M, μ) with $M = (\mathcal{P}, E, (<_p)_{p\in\mathcal{P}}, \mu, <_{\mathsf{s}}, \alpha, \varphi)$, we let $\mathcal{M}\sigma = (M\sigma, \mu\sigma)$ where $M\sigma = (\sigma(\mathcal{P}), E, (<_{\sigma^{-1}(p)})_{p\in\sigma(\mathcal{P})}, \mu, <_{\mathsf{s}}, \alpha, \sigma(phi))$ and $\mu\sigma(\pi) = \sigma(\mu(\pi))$ for $\pi \in \Pi$. For a rule $r = (A, expr, f)$ with $expr = u_0.\mathcal{M}_1.u_1 \ldots \mathcal{M}_k.u_k$, we set $r\sigma$ to be the rule $(A, expr\sigma, f\sigma)$ where $expr\sigma = u_0.\mathcal{M}_1\sigma.u_1 \ldots \mathcal{M}_k\sigma.u_k$ and $f\sigma(q) = f(\sigma^{-1}(q))$ for $q \in Free(expr\sigma)$.

Non-terminal replacement allows for a set of relations $\{\overset{r}{\Longrightarrow}_G\}_{r\in\longrightarrow}$ over configurations indexed by rules $r \in \longrightarrow$. For configurations $\mathcal{C} = (\mathfrak{M}, A.\gamma)$ and $\mathcal{C}' = (\mathfrak{M}, expr.\gamma)$, $\mathfrak{M} = (M, \nu)$, and $r \in \longrightarrow$, we let $\mathcal{C} \overset{r}{\Longrightarrow}_G \mathcal{C}'$ if there is a renaming σ such that $r\sigma = (A, expr, f)$, $\nu(f(p)) = p$ for all $p \in Free(expr)$, and $Proc(M) \cap Bound(expr) = \emptyset$.

Semantics The semantics of G is given as the set of (named) MSCs appearing in final configurations that can be derived from an initial configuration by means of relations $\overset{r}{\Longrightarrow}_G \subseteq Conf_G \times Conf_G$ (for every rule r) and $\overset{\mathrm{e}}{\Longrightarrow}_G \subseteq Conf_G \times Conf_G$.

We define $\Longrightarrow_G$ to be $\overset{\mathrm{e}}{\Longrightarrow}_G \cup \bigcup_{r\in\longrightarrow} \overset{r}{\Longrightarrow}_G$. The dMSC language of G is the set $\mathcal{F}_G := \{M \in \mathbb{M} \mid \mathcal{C}_0 \Longrightarrow_G^* ((M, \nu), \varepsilon)$ for some initial configuration $\mathcal{C}_0$ and $\nu\}$.

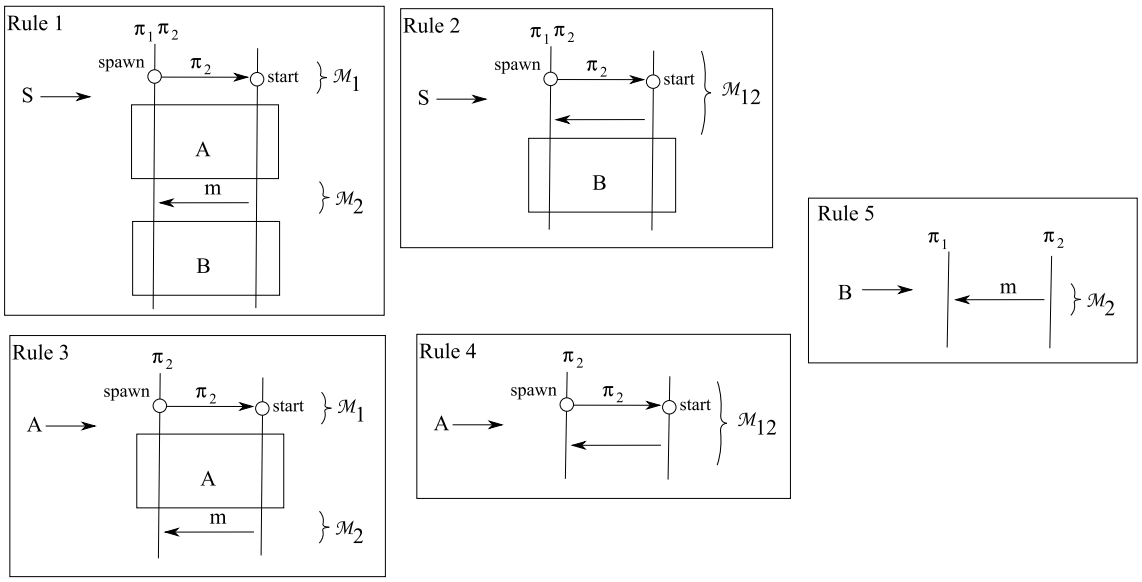


Figure 6.4: A dynamic MSC grammar

Let us illustrate the formal definition of MSC grammars and derivations through a second example. Figure 6.4 depicts a DMG with non-terminals $\mathcal{N} = \{S, A, B\}$, start symbol S, process identifiers $\Pi = \{\pi_1, \pi_2\}$, and five rules. Any rule has a left-hand side (a non-terminal), and a right-hand side (a sequence of non-terminals and PMSCs). In a derivation, the left-hand side can be replaced with the right-hand side. This replacement, however, depends on a more subtle structure of a rule. The bottom left one, for example, is actually of the form $A \longrightarrow_f expr$ with $expr = \mathcal{M}_1.A.\mathcal{M}_2$, where f is a function that maps the first process of $expr$, which is considered *free*, to the process identifier π_2. This indicates where $expr$ has to be inserted when replacing A in a configuration. To illustrate this, consider a derivation as depicted in Figure 6.5, which is a sequence of configurations, each consisting of an upper and a lower part. The upper part is a *named* MSC [92], an MSC where some

processes are tagged with process identifiers. The lower part, a sequence of PMSCs and non-terminals, is subject to further evaluation. In the third configuration, which is of the form $(\mathfrak{M}, A.\beta)$ (with named MSC $\mathfrak{M}$), replacing A with *expr* requires a renaming σ of processes in *expr*: the first process of *expr*, tagged with π_2, takes the identity of the second process of $\mathfrak{M}$, which also carries π_2. The other process of *expr* is considered newly created and obtains a fresh identity. Thereafter, A can be replaced with $expr\sigma$ so that we obtain a configuration of the form $(\mathfrak{M}, \mathcal{M}.\gamma)$, $\mathcal{M}$ being a PMSC. The next configuration is $(\mathfrak{M} \circ \mathcal{M}, \gamma)$ where the concatenation $\mathfrak{M} \circ \mathcal{M}$ is simply performed on the basis of process names and does not include any further renaming. Process identifiers might migrate, though. Actually, $\mathcal{M}$ is a pair (M, μ) where M is a PMSC and μ partially maps process identifiers π to process pairs (p, q), allowing π to change its location from p to q during concatenation (cf. the fifth configuration in Figure 6.5-bottom right, where π_2 has moved from the second to the third process).

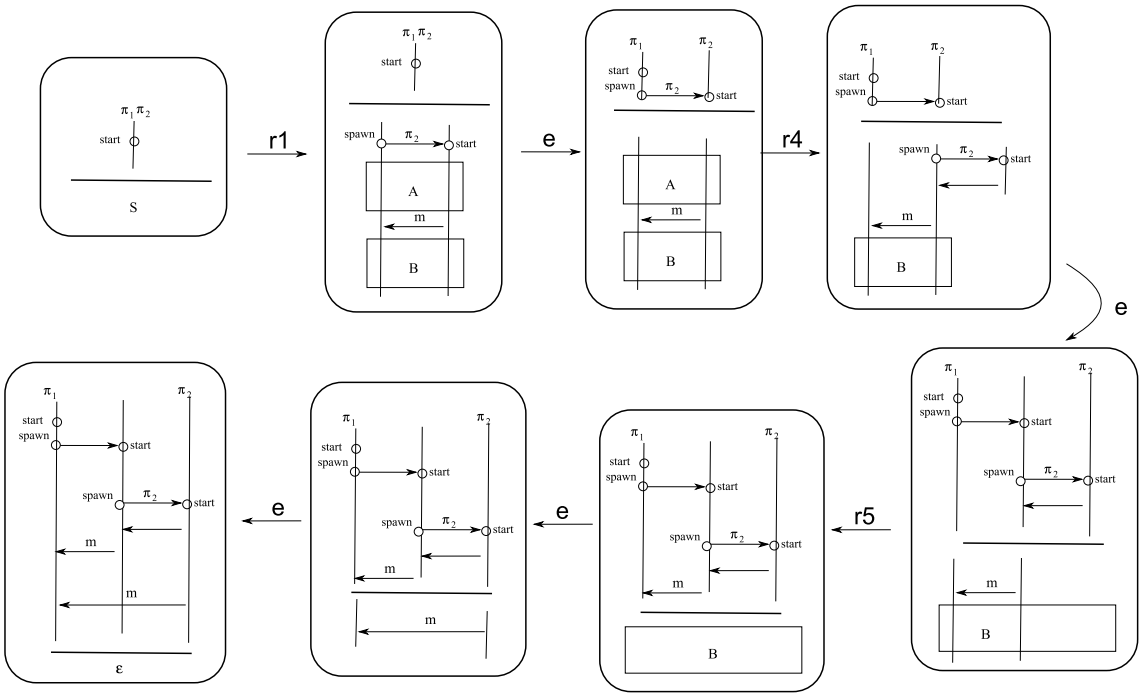


Figure 6.5: A derivation

To conclude this section, let us formalize the DMG $G = (\Pi, \mathcal{N}, S, \longrightarrow)$ depicted in Figure 6.4. Given the PMSCs M_1 and M_2 from Figure 6.2, we let $\mathcal{M}_1 = (M_1, \mu_1)$, $\mathcal{M}_2 = (M_2, \mu_2)$, and $\mathcal{M}_{12} = (M_1 \circ M_2, \mu_1)$ be in-out PMSCs with $\mu_1(\pi_1), \mu_2(\pi_1), \mu_2(\pi_2)$ undefined and $\mu_1(\pi_2) = (1, 2)$. Then, $\longrightarrow$ is composed of the rules

$$S \longrightarrow_{f_S} \mathcal{M}_1.A.\mathcal{M}_2.B \qquad S \longrightarrow_{f_S} \mathcal{M}_{12}.B \qquad B \longrightarrow_{f_B} \mathcal{M}_2$$
$$A \longrightarrow_{f_A} \mathcal{M}_1.A.\mathcal{M}_2 \qquad A \longrightarrow_{f_A} \mathcal{M}_{12}$$

where $f_S(1) = f_B(1) = \pi_1$ and $f_A(1) = f_B(2) = \pi_2$. A sequence of configurations starting in I_p and linked by the relation $\Longrightarrow_G^*$ is graphically depicted in Figure 6.5. In any configuration, the part above the first non-terminal (if there is any) illustrates its named MSC.

2.4 Comparison of DMGs, HMSCs and existing dynamic models

Let us now compare the respective expressive power of HMSCs and DMGS, and fork-join grammars. Strictly speaking, HMSCs and DMGS are incomparable, as we have required any production of a DMG to be a dynamic MSC with a single minimal start event, and as HMSC do not start each process with a start action. However, this is mainly a notational convention to represent threads, avoid duplicate process creation during dMSC construction.

As grammar rules can refer to MSCs, and as a grammar is more powerful that a finite state machine, one can easily create a grammar that simulates a HMSC H. For every HMSC H over a set of processes $\mathcal{P}$, one can design a dynamic MSC grammar G_H with process identifiers $\Pi = \{\pi_1, \ldots \pi_p\}$ which derivations all start with a rule spawning all processes in $\mathcal{P}$, followed by sequences of rewritings that produce some MSC in $\mathcal{L}(H)$. Figure 6.6 is such a translation for a simple HMSC. However, HMSCs define behaviors over a fixed set of processes; Hence, we can compare both models and claim that Dynamic MSC grammars are strictly more expressive than HMSCs.

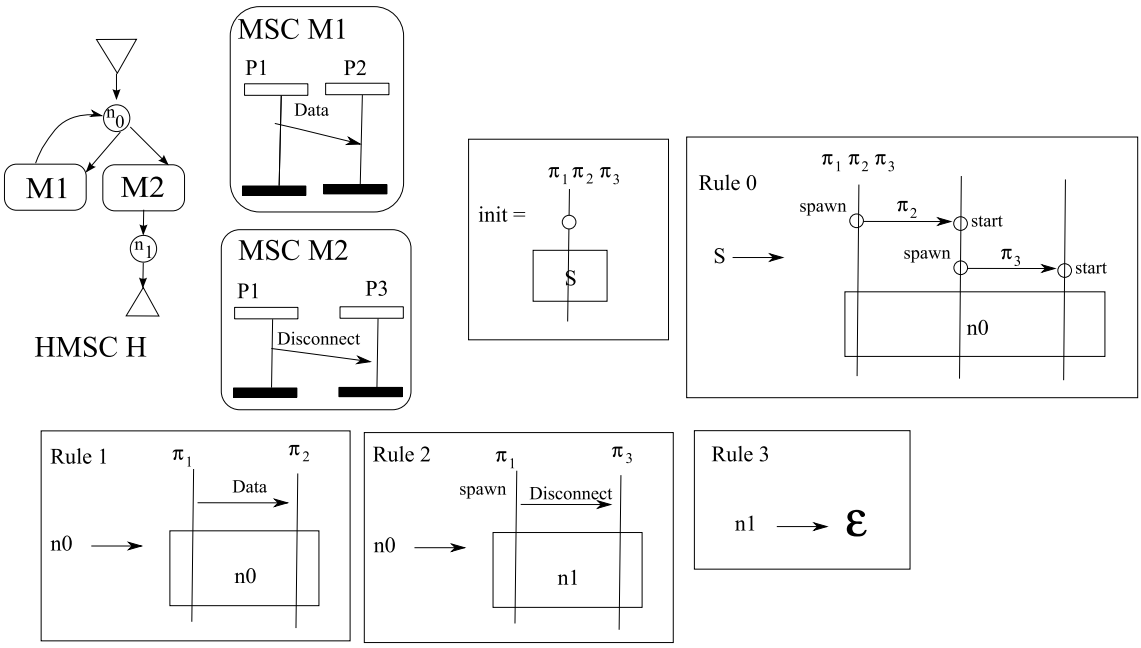


Figure 6.6: A HMSC H and a grammar G_H that simulates it

Let us now compare the expressiveness of DMGS w.r.t. the fork-join grammars of [92]. Some MSC languages of fork-join grammars can not be implemented by DMGs. For instance, partial orders of the form of Figure 6.7 in which pairs of newly created processes send messages to the same pair of processes can be easily modeled with a fork-join grammar, but not with a DMG. In fork-join grammars, named MSCs are assembled according to process identifiers, but a specific *split* operator allows to distribute disjoint subsets of operators overs subexpressions. For instance, in the expression $M.split_{\{A,B\}\{C,D\}}(N, N').M'$, in which M, M', N, N' are named MSCs, the behavior started on A in M continues as described in N, and if a process is labeled A in N', it is considered as a new process. This allows for

instance creation of processes without explicit spawn action. This is one of the difference with our grammars (in DMGs new processes are necessarily spawned by existing ones). Furthermore, in DMGs, when a non-terminal is replaced by the corresponding expression, identities of processes in terminal MSCs are fixed once for all. In fork-join grammar, a terminal named MSC is appended with different processes identities depending on how a preceding subexpression is evaluated. Note that in fork-join grammars, processes no not explicitly "migrate" as with DMGs. However, this is not limiting, and recursive patterns such as the one in figure 6.3 can be easily defined by fork-join grammars. We think that DMGs languages can be produced by fork-join grammars at the cost of using more process identifiers to "store" the identity of processes before developing a non-terminal. We hence conjecture that DMGs are strictly less expressive than fork-join grammars.

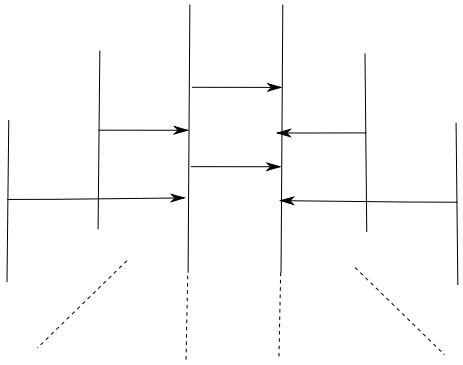


Figure 6.7: A set of behaviors that can not be modeled by a DMG.

3 Properties of DMGs, verification

In this section, we address the properties of Dynamic MSC grammars. First of all, as DMGs embed the expressive power of HMSCs, all properties that are undecidable for HMSCs (regularity, intersection emptiness, language inclusion,....) are also undecidable for DMGS. However, many decidable properties of HMSCs remain decidable for DMGs.

Before considering properties of DMGs, we can note several syntactic properties of the model.

- In a configuration of the form $(\mathfrak{M}, \beta)$ where $\mathfrak{M} = (M, \nu) \in \mathrm{n}\mathbb{M}$ and $\beta \in (\mathrm{m}\mathbb{P} \cup \mathcal{N})^*$, every process such that $\nu(\pi_i) = p$ contains a start event. This remark is important, as it allows to decide without considering M whether some concatenation $\mathfrak{M} \circ \mathcal{M}$ is defined or not.

- In a configuration of the form $(\mathfrak{M}, \beta)$ where $\mathfrak{M} = (M, \nu) \in \mathrm{n}\mathbb{M}$ and $\beta \in (\mathrm{m}\mathbb{P} \cup \mathcal{N})^*$, if $\beta = \mathcal{M}.\beta'$ for some $\mathcal{M} = (M', \mu)$, then $\mathfrak{M} \circ \mathcal{M}$ is defined if and only if for every process identifier π_i, if $\mu(\pi_i) = (p, q)$, then $\nu(\pi_i) = p$, and furthermore, $M \circ \mathcal{M}'$ is defined (i.e. each process in M, M' has one single minimal start event. Clearly, both conditions can be checked without knowing M.

- let $A \longrightarrow_f expr$ be a rule of a grammar, and $(\mathfrak{M}, A.\beta)$ a configuration. Let $expr = \mathcal{M}_1.u.\mathcal{M}_2$. Applying rule r means choosing a renaming of processes in $\mathcal{M}_1, \mathcal{M}_2$, and replacing A by $\mathcal{M}_1.u.\mathcal{M}_2$. At the time of rewriting, the identity of processes in both $\mathcal{M}_1, \mathcal{M}_2$ are chosen. If rules are written correctly, one should expect f to be compatible with μ_1, but the fact that $\mathcal{M}_2$ can be appended to a configuration may still depend on how u is rewritten. That is, a derivation of a grammar is not a derivation corresponding to G seen as a word grammar, and process identifiers mappings may prevent application of some rules to lead to final configurations. Consider for instance the simple grammar of Figure 6.8. After application of rule $(r1)$ to the axiom, we reach a configuration $C = ((M, \nu), \beta)$, where (M, ν) is a named MSC containing tow processes (say $1, 2$), a creation from 1 to 2. We also have $\nu(\pi_1) = 1$ and $\nu(\pi_2) = 2$, and $\beta = A$. After application of rule $r2$, we obtain a configuration $C' = ((M', \nu'), \beta')$, where M' is the MSC M followed by a message from 1 to 2 and an answer from 2 to 1. Moreover, we have $\nu'(\pi_1) = 2$,$\nu'(\pi_2) = 1$, and $\beta' = B.(N, \mu)$, where N is an MSC containing a simple message exchange from process 1 to process 2, and $\mu(\pi_1) = (1, 1)$ $\mu(\pi_1) = (2, 2)$. This configuration is represented in Figure 6.9. Note that between C and C', processes 1 and 2 have exchanged their identities. Hence, if we apply rule $r3$, we reach a configuration C'' that is not a final configuration, and from which the remaining in-out MSC (N, μ) can not be appended to reach a final configuration. On the other hand, applying rule $r4$ swaps again the identifiers of processes, and allows for the replacement of non-terminal B using rule $r3$, and then for the concatenation of (N, μ). Hence, any sequence of rule applications of the form $r1.r2.(r4.r2)^{2.n+1}.r3$ where $n \in \mathbb{N}$ leads to a final configuration, but sequences of the form $r1.r2.(r4.r2)^{2.n}.r3$ contain deadlocked configurations, and hence can never reach a final configuration.

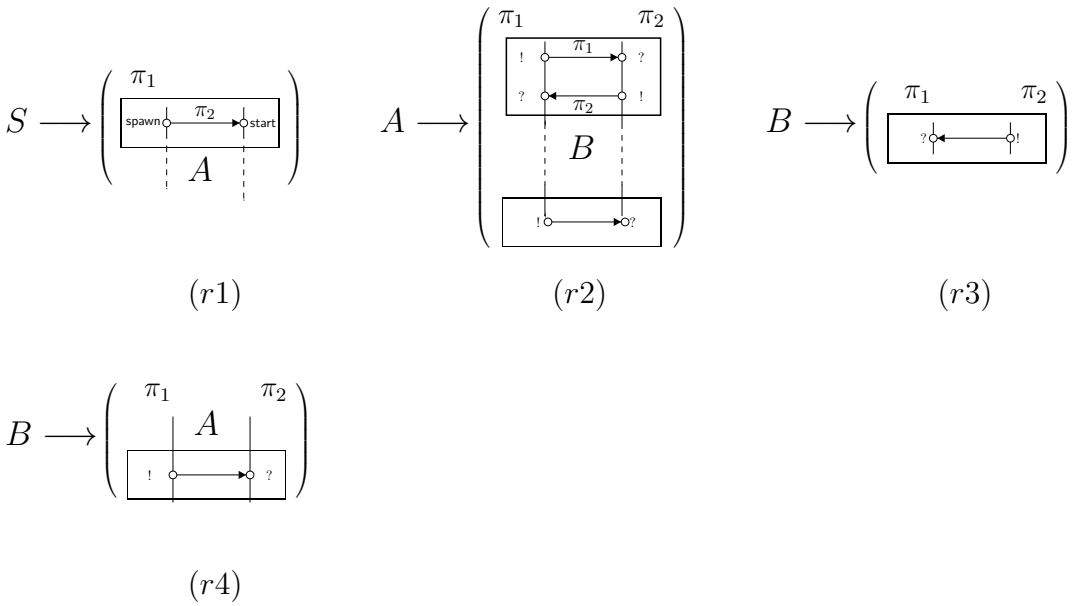


Figure 6.8: A dynamic MSC grammar which derivations can deadlock.

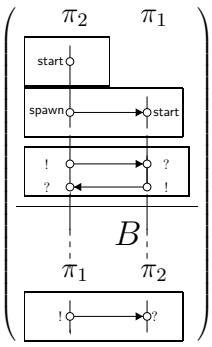


Figure 6.9: A derivation for the grammar of Figure 6.8

This last remark highlights one important fact: while in HMSCs, all path of the underlying automaton can be associated an MSC obtained by concatenation of MSC along the transitions of the path, some derivations of an MSC grammars my never reach a final configuration. Even worse, one can design a DMG G such that $\mathcal{L}(G) = \emptyset$. Consider for instance the example of Figure 6.8, and remove rule $r4$. Then, the only possible derivation is the sequence of rules $r1.r2.r3$, and we have already seen that one can not reach a final configuration using this sequence of rewritings. Hence, there is no sequence of rule applications leading to a well-defined concatenation of μ-MSCs.

For a specification formalism, this is particularly harmful, as one can design a model that has no meaning. We hence define the emptiness problem as follows: given an MSC grammar G, check whether $\mathcal{L}(G) = \emptyset$.

Theorem 39 *The emptiness problem for Dynamic MSC grammars is decidable in* $EXPTIME$.

The idea of the proof is to build a tree automaton A_G that recognizes derivation trees for a DMG G. However, states of this tree automaton contain labels associated to terminal MSCs, non-terminal letters, plus some information on how processes should migrate during rewriting of a non-terminal. The construction of the tree automaton leads to an exponential blowup is the size of the initial grammar. Then checking emptiness of $\mathcal{A}_G$ can be done in $O(|Q|.(|Q|.\delta))$. A more detailed proof sketch can be found in appendix.

An immediate corollary of this result is that one can also decide whether all derivations of an MSC grammar lead to a $\mathfrak{M}$. This can be achieved by building a tree automaton $\mathcal{A}_G^{word}$ that recognizes parse trees of G seen as a word grammar, and use the automaton $\mathcal{A}_G$ built in theorem 39. Then checking that $\mathcal{L}^{word}(\mathcal{A}_G) \subseteq \mathcal{L}(\mathcal{A}_G)$ can be performed in $EXPTIME$.

4 Conclusion

In this chapter, we have introduced a dynamic scenario model, namely Dynamic MSC grammars. DMGs are strictly more powerful than HMSCs, and we conjecture that they are strictly less powerful than the fork-join grammars proposed by [92]. In the next chapter, we will propose an implementation model for dynamic MSC

grammars, and show that for a subset of the language, one can decide whether there exists a dynamic implementation for a given DMG.

Verification problems for MSC grammars inherit all undecidable properties of HMSCs. However, one can decide the vacuity of the language generated by a grammar. The technique relies on the construction of tree automata, that which size can be exponential in the size of the original grammar. An interesting issue would be to consider standards decision problems for subclasses of DMGs, and see if usual subclasses (regular, globally cooperative, ...) defined for HMSCs can also be defined for this model.

Several other issues for DMGs remain open. First, we would like to study regular MSC grammars. Regular sets of MSCs over a fixed number of processes have already been studied [74] (a set of MSCs is called regular if the associated linearization language is a regular language). We would like to define a robust notion of regularity that takes thread creation into account. Preferably, any regular set of dynamic MSCs should have an implementation in terms of a dynamic communicating automaton. Note however that the linearizations of a set of (dynamic) MSCs are words over an infinite alphabet. Techniques borrowed from [93] might help in the characterization of dynamic MSC languages.

Chapter 7

Implementation

Le désordre simulé suppose une discipline parfaite.
Simulated disorder postulates a perfect discipline.
[Lao Tseu]

1 Introduction

MSCs and their variants are usually described as intuitive, easy to learn models, etc. However, one can see from the bibliography on MSCs, sequence diagrams, etc, that scenario models are often used to design simple finite use-cases, and less frequently complete protocols in industrial contexts. HMSCs were successfully used as a model to find bugs in cell-phones initialization phases at Motorolla [18] but up to now, we are not aware of software production that started from HMSCs to derive running code. We are convinced that automatic implementation techniques are a key element to promote scenarios usage during software production. Of course, HMSCs are frequently designed at a high level of abstraction, but even though, they can be used to model intuitively and formally control flows of distributed programs.

Previous chapters have mentioned decidability issues, expressiveness problems and possible extensions of the language to overcome them, and subclasses of scenario models to allow verification. However, verification of scenarios can be performed on models that can not yet be implemented on a distributed architecture. We are convinced that implementability of a model, and synthesis problems are other important factors to consider. Roughly speaking, implementability consists in deciding whether a formal model can be implemented on a network of machines, and the synthesis problem consists in generating a distributed program for these machines.

This chapter addresses implementability and synthesis issues for HMSCs and dynamic HMSCs. Implementability and synthesis are practical concerns: when automatic and correct implementation solutions exist for a formal model, the time spent on a formal models benefits to the code design phase. Designing a formal model for a future "real" system is a heavy and costly task. If no code can be generated from a specification, then all verifications performed on a formal model are hindered by the fact that the coding phase may end up with a program that does not respect the specification, and hence may not satisfy the good properties of the model. Within this context, there are good chances that a formal model of

a system will never be designed. The same arguments can be considered from a more theoretical point of view: does it make sense to study properties of models that are connected in any way to real distributed system (and continue to call them "models of distributed systems")? How far is a class of scenarios from real running application (and hence how pertinent is a verification solution for this subclass) ?

A good way to reconciliate verification and implementation is to define (automatic) code synthesis techniques that generate a code skeleton, and preserve properties of a specification. However, synthesis should output programs that exhibit exactly the same behavior as the original specification. This is usually not the case for HMSCs : either the synthesis process needs to add new messages and synchronizations to ensure correct scheduling of all events appearing in the HMSC behaviors, or the generated implementation uses exactly the same sets of events and messages, and the synthesized behaviors allow more executions that in the original specification. Solutions to overcome this problem are usually:

- to restrict to subclasses of the considered scenario language that guarantee correct synthesis, or
- to consider additional behaviors as part of the semantics of the specification (hence changing the semantics of the model). We will show later that additional behaviors usually appear when two branches of a choice can be initiated by distinct processes. The *implementation semantics* proposed in the literature consists in including so called *implied scenarios*, that are shuffles of the scenarios depicted by conflicting branches [134, 136]. This implementation semantics corresponds to an implementation of a HMSC by communicating machines synthesized by projection of the original specification.

We think that the second option is less appealing, at least when considering HMSC specifications. The first reason is that the behaviors of an HMSC specification with "additional behaviors" is larger than the set of intended specified behaviors. Calling $\mathcal{L}^{impl}(H)$ the "implementation" semantics of H, that is the set of linearizations defined by communicating processes that implement H, we necessarily have $\mathcal{L}(H) \subseteq \mathcal{L}^{impl}(H)$. The usual Interpretation of an implementation for HMSCs is that the implementation model is a communicating finite state machine. However, we can easily prove the following properties:

- $\mathcal{L}^{impl}(H)$ is not always the language of a safe CHMSC.
- $\mathcal{L}^{impl}(H) \setminus \mathcal{L}(H)$ is not always the language of a safe CHMSC.
- One can not decide whether $\mathcal{L}^{impl}(H) = \mathcal{L}(H)$ [12]

The first point may raise verification problems. We have seen in former chapters that verification of globally cooperative (C)HMSCs is decidable, but that verification of simple problems outside the class of safe CHMSCs is in general impossible. We will show in the next sections that choosing an implementation semantics for HMSCs and their extensions leads to undecidability. The second point means that, the language of additional behaviors is not representable by a safe CHMSC either. Hence, one can not in general represent the additional behaviors as a separate model, and

analyze the properties of the additional behaviors. Combined with the third point, we can deduce that even if a formal model allows for a finite representation of $\mathcal{L}^{impl}(H) \setminus \mathcal{L}$, then one can not decide whether this model has an empty set of behaviors or not. Hence, considering HMSCs equipped with an implementation semantics is a dangerous solution. It introduces a discrepancy between between the modeled behavior and the supposed semantics (this situation can be compared to the race problem introduced in chapter 2). Second, the implementation semantics can not be represented by models with decidable properties, even when the original model was a simple HMSC. Last, discrepancies between can not be represented nor analyzed (and in particular lack of such differences) and there exist no algorithm computing a "closure of a cHMSC by implementation".

To conclude on the implementation semantics, the properties listed above show that a designer relying on this semantics has no way to know if a designed model has good properties or even behaves as expected. In a design process, in which an abstract model is mapped onto a real architecture, mastering the discrepancies between the specification an an implementation is a key issue, and implementation semantics does not offer this possibility, as all properties that could help deciding if a model conforms to users expectations are undecidable.

In this chapter, we adopt another point of view: instead of considering partial order automata with their semantics up to implementation, we want to consider HMSCs for which an equivalent implementation exists. For this, we can not consider the semantic properties of a model, as one can not decide in general if a HMSC has an equivalent CFSM implementing it [12]. Hence, we propose to consider syntactic subclasses of HMSCs for which an equivalent implementation always exists. We will consider in particular the class of local-choice HMSCs, for which simple syntactic restrictions allow an implementation by communicating automata obtained by a simple projection. Without these restrictions local-choice HMSCs can be implemented by a variant of CFSM that maintains vectorial clocks. For Dynamic MSC grammars, we will show that a subclass of "local DMGs" can be implemented by a variant of CFSMs that allows for the creation of threads.

The work presented hereafter on implementation of HMSCs is a joint work with Claude Jard, Rouwaida Abdallah, and the work on implementation of DMGs is a joint work with Benedikt Bollig.

2 A canonical model for implementation ?

Most of works on HMSCs implementation assume that the implementation is done with communicating finite state machines (CFSM) introduced by Brand & Zafiropulo [32]. For the sake of completeness, we introduce this well-known model hereafter. CFSMs are a natural implementation model for several reasons: first, HMSCs describe the behavior of independent agents, which continuously run sequences of communication events and atomic actions. Hence, if we want to respect the independence of agents in the architecture depicted in an HMSC, an implementation model must allow for the definition of parallel components. Models such as Petri nets or networks of automata communicating via shared actions fulfill these requirements. However, scenario models clearly depict agents that communicate asynchronously, which rules out communications using shared actions. One can also notice that

HMSCs semantics can enforce messages between a pair of processes to respect FIFO ordering, which can not be enforced by Petri nets. In fact, it has been shown that synthesis of Petri nets from HMSCs usually produces an over approximation of the initial HMSC language [33]. All these considerations call for the use of CFSMs as target architecture.

Similarly, synthesis of CFSM is very often interpreted as a projection of the HMSC on each of its processes. Let us recall that for models such as HMSCs and CHMSCs, the language $\bigcup_{M \in \mathcal{L}(H)} \pi_p(M)$ is a regular language, and can hence be seen as the language of a finite state machine which transitions are labeled by communication events or atomic actions. The CFSM obtained by projection of a HMSC/CHMSC on each of its processes is often called the *canonical implementation*, and realizability of some HMSC is usually defined w.r.t. this canonical model [11,95]. However, we will see in sections 4 that extensions of CFSMs can also be used as implementation model for larger decidable syntactic subclasses of HMSC, and even of dynamic MSCs.

Let us now introduce formally Communicating Finite State Machines (CFSM) [32]. A CFSM $\mathcal{A}$ is a network of finite state machines that communicate over unbounded, non-lossy, error-free and FIFO communication channels. We will write $\mathcal{A} = \parallel_{i \in \mathcal{P}} A_i$ to denote that $\mathcal{A}$ is a network of machines describing the behaviors of a set of machines $\{A_i\}_{i \in \mathcal{P}}$. A communication buffer $B_{(i,j)}$ is associated to each pair of instances $(i,j) \in \mathcal{P}^2$. Buffers will implement messages exchanges defined in the original HMSC. More formally, we can define a communicating automaton as follows:

Definition 63 *A* communicating automaton *associated to an instance p is a tuple $A_p = (Q_p, \delta_p, \Sigma_p, q_{0,p})$ where Q_p is a set of states, $q_{0,p}$ is the initial state, Σ_p is an alphabet with all letters of the form $p!q(m)$ $p?q(m)$ or $p(a)$, symbolizing message sending to a process q, reception from a process q, an atomic action a executed by process p, or a silent move ε. The transition relation $\delta_p \subseteq Q_p \times \Sigma_p \times Q_p$ is composed of triples (q, σ, q') indicating that the machine moves from state q to state q' when executing action σ. A* CFSM $\mathcal{A} = \parallel_{i \in I} A_i$ *is a composition of communicating automata.*

Figure 7.1: Two communicating machines

Note that our definition of CFSM does not contain final states. The reason is that we will consider implementation of MSC languages that are closed by prefix.

Figure 7.1 describes a CFSM composed of two finite state machines A_{Client} and A_{Server}. The initial states of these two machines are denoted by a dark incoming arrow. Each run of a set of communicating machines defines a prefix, that can be built incrementally starting from the empty prefix, and appending one executed event after the other (i.e. it is built from a total ordering of all events occurring on the same process, plus a pairing of messages sendings and receptions). Then, the language $\mathcal{L}(\mathcal{A})$ of a set of communicating machines is the set of all prefixes associated to runs of $\mathcal{A}$.

Let us formalize the semantics of a network of communicating automata. A *configuration* of a network of automata $\mathcal{A} = \parallel_{i \in \mathcal{P}} A_i$ is a pair $C = (L, W)$ where L is a sequence of sates $q_1 \dots q_{|\mathcal{P}|}$ depicting the local state of each communicating machine, and $W = \{w_{11}, \dots w_{1|\mathcal{P}|}, w_{21}, \dots w_{2|\mathcal{P}|}, \dots w_{|\mathcal{P}||\mathcal{P}|}\}$ is a set of $|\mathcal{P}|^2$ words depicting the contents of message buffers. Each w_{ij} is a sequence of message names, and depicts the contents of the queue from A_i to A_j. Then, the behavior of $\mathcal{A}$ is defined as follows:

- all machines start from their initial states with all communication buffers empty, that is the initial configuration is $C_0 = (L_0 = q_{0,1}. \dots . q_{0,|\mathcal{P}|}, W_0 = \{\varepsilon, \dots \varepsilon\})$.

- From a configuration C, a machine A_p can send a message m to a machine A_q if A_p is in local state q_p, there exists a transition $(q_p, p!q(m), q'_p)$ in A_p. Executing this action $p!q(m)$ simply appends m to the buffer $w_{p,q}$ from p to q and changes A_p's local state to q'_p in the configuration. Hence, if $C = (L, W)$ with $L = q_0 \dots q_p \dots q_{|I|}$ and $W = \{w_{11}, \dots w_{p,q} \dots w_{|\mathcal{P}||\mathcal{P}|}\}$, executing $p!q(m)$ results in a configuration $C' = (L', W')$ with $L' = q_0 \dots q'_p \dots q_{|\mathcal{P}|}$ and $W' = \{w_{11}, \dots w_{p,q}.m \dots w_{|\mathcal{P}||\mathcal{P}|}\}$ Local actions of communicating automata change the local state of a single machine and leave the buffer contents unchanged.

- From a configuration C, A_p can receive a message m from process q, if A_p is in local state q_p, there exists a transition $(q_p, p?q(m), q'_p)$ in A_p, and the first letter of $w_{q,p}$ is m (which means that m is the first message that has to be received in the queue from q to p). Executing this action $p?q(m)$ simply removes m from the buffer $w_{p,q}$ from p to q and changes A_p's local state to q'_p in the configuration. Hence, if $C = (L, W)$ with $L = q_0 \dots q_p \dots q_{|\mathcal{P}|}$ and $W = \{w_1 1, \dots w_{p,q} = m.w \dots w_{|\mathcal{P}||\mathcal{P}|}\}$, executing $p?q(m)$ results in a configuration $C' = (L', W')$ with $L' = q_0 \dots q'_p \dots q_{|\mathcal{P}|}$ and $W' = \{w_{11}, \dots w_{p,q} = w \dots w_{|\mathcal{P}||\mathcal{P}|}\}$.

This way, CFSMs define sequences of actions $\sigma_1 . \dots . \sigma_k$ that can be executed by their local components from their initial states. Each action moves the communicating machines from one configuration to another. However, CFSM are concurrent models, and their executions can be represented in a non-interleaved way by MSC prefixes.

In the next sections, we will consider incomplete executions of MSCs in which some messages have been sent and not yet received. This notion of incomplete execution is captured by the definition of *pieces* and *prefixes*.

Definition 64 (prefix, suffix, piece of MSCs) *Let $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \varphi)$ be a MSC. A* prefix *of M is a tuple $(E', (<'_p)_{p\in\mathcal{P}}, \alpha', \mu', \varphi')$ such that E' is a subset of E closed by causal precedence (i.e. $e \in E' \wedge f \leq e \Longrightarrow f \in E'$) and $<'_p, \alpha', \mu', \varphi'$ are restrictions of $<_p, \alpha, \mu, \varphi$ to E'. A* suffix *of M is a tuple $(E', (<'_p)_{p\in\mathcal{P}}, \alpha', \mu', \varphi')$ that E' is closed by causal succession (i.e. $e \in E' \wedge e \leq f \Longrightarrow f \in E'$) and $<'_p, \alpha', \mu', \varphi'$ are restrictions of $<_p, \alpha, \mu, \varphi$ to E'. A* piece *of M is the restriction of M to a set of events $E' = E \setminus X \setminus Y$, such that the restriction of M to X is a prefix of M and the restriction of M to Y is a suffix of M.*

Note that prefixes, suffixes and pieces are not always MSCs, as their message mappings m are not necessarily bijections from sending events to receiving events. In the rest of the chapter, we will denote by $Pref(M)$ the set of all prefixes of a MSC M, and for an MSC language L, $Pref(L)$ will denote the prefix closure of L. We will denote by O_ε the empty prefix, i.e. the prefix that contains no event. For a particular type of action a, we will denote by O_a a piece containing a single event of type a. The examples of Figure 7.2 shows a MSC M involving three processes P, Q, R, a prefix Pr, a suffix S, and a piece Pc. Observe that Pc is obtained by erasing Pr and S from M. Note also that Pr, S and Pc contain incomplete messages.

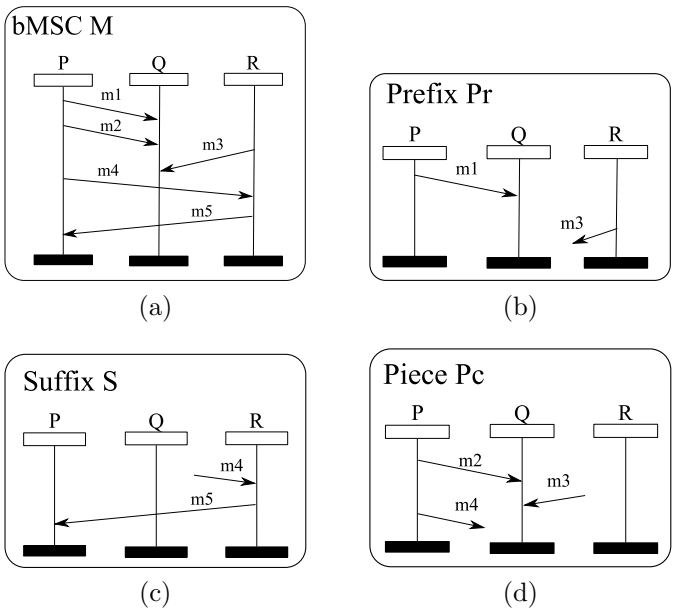


Figure 7.2: A MSC (a), a prefix (b), a suffix (c) and a piece (d)

Prefixes and pieces of MSCs can be concatenated as MSCs, but with an additional phase that rebuilds the message mappings. Let O_1 be a prefix of a MSC, and O_2 be a piece of MSC. Then, the concatenation of O_1 and O_2 is denoted by $O_1 \circ O_2 = (E, \leq, C, \varphi, t, m)$, where $E, \leq, C, \varphi$, and t are defined as for sequential composition of MSCs (see definition 6 in chapter 1) and m is a function that associates the n^{th} sending event from p to q to the n^{th} reception from p on q for every pair of processes $p, q \in \mathcal{P}$. Note that this sequencing is not defined if for some p, q, n, the types of the n^{th} sending and reception do not match, that is one event is of the form $p!q(m)$ and the other one $q?p(n)$ with $n \neq m$. In particular, we will denote by $O \circ \{e\}$ the prefix obtained by concatenation of a single event e to a prefix O.

Definition 65 *Let $\mathcal{A} = \parallel_{i\in\mathcal{P}} A_i$ be a CFSM. The language of $\mathcal{A}$ is denoted by $\mathcal{F}(\mathcal{A})$ and is the set of MSC prefixes defined inductively as follows :*

- *the prefix associated to an empty sequence of actions is the empty prefix O_ε,*
- *the prefix associated to a sequence of actions $\sigma_1 \dots \sigma_k.\sigma_{k+1}$ of $\mathcal{A}$ is the prefix $O \circ \{e\}$ where e is an event labeled by σ_{k+1} and O is the prefix associated to $\sigma_1 \dots \sigma_k$.*

3 realizability of HMSCs

Before entering the details of implementation, let us recall some well known results on the the implementability of HMSCs. Consider for instance the MSCs of Figure 7.3 (example borrowed from [11]). The MSC $M1$ describes a behavior in which message m is received before message n on process $P2$ and similarly on $P3$. The MSC $M2$ describes a behavior in which message n is received before message m on process $P2$ and similarly on $P3$. As $P2$ and $P3$ do not communicate, in an implementation that uses only messages m and n, one can not avoid the implied behavior $M_{implied}$ in which m is received before n on $P2$ and n is received before m of $P3$. A similar scenario with the converse orders on $P2$ an $P3$ is also unavoidable.

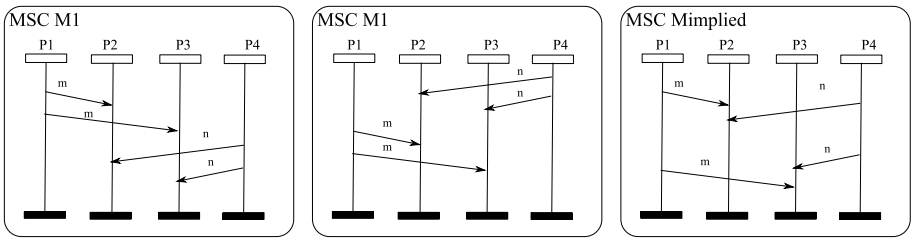


Figure 7.3: Two MSCs, and an additional implied scenario

Definition 66 *Let M be a MSC. The projection of M on process $p \in \mathcal{P}$ is denoted $\Pi_p(M)$ A set of MSCs L* weakly implies *a MSC M if for every process p of $\mathcal{P}$, there exists an MSC $M_p \in L$ such that $\Pi_p(M) = \Pi_p(M_p)$. The set L^w is the set of all MSCs that are implied by MSCs in L. A set of MSCs L is* weakly realizable *iff $L = L^w$.*

Being weakly realizable for an MSC language is close to being implementable: it guarantees that there exists a set of communicating finite machines that implements all local (process wise) behaviors of MSCs in L, and that their composition produces behaviors that are also in L. Note also at this point that realizability in expressed in terms of projections, which are regular for finite MSC languages, HMSC and CHMSC languages. Hence, the implementation model for a scenario specification is frequently assumed to be the set of communicating machines obtained by projection of the specification on each of its processes, and is called the *canonical implementation*. Realizability then amounts to checking whether the canonical implementation of an MSC language realizes exactly the specification. One can easily see that weak

realizability for a finite set of MSCs L can be checked in $O(m.|L|^{|\mathcal{P}|+1})$, where m is the size of the largest MSC in L. Indeed, a weak-FIFO MSC M can be represented by the set of its projections $(\pi_1(M), \ldots \pi_{|\mathcal{P}|}(M))$. Checking that a finite MSC language L is weakly realizable consists in building all possible combinations $(\pi_1(M_{i1}), \ldots \pi_{|\mathcal{P}|}(M_{i_{|\mathcal{P}|}}))$, and then checking that the obtained structure is an MSC and more precisely a MSC from L. Solving this problem is also in co-NP [10], as one can build a combination of projections in polynomial time and return a negative answer as soon as a witness is found. However, the decidability for finite MSC languages does not extend to HMSCs.

Theorem 40 ([11]) *Let H be a regular HMSC. Then checking whether $\mathcal{F}_H$ is weakly realizable is undecidable.*

The proof of this theorem is again based on an encoding of a PCP variant, and can be found in [11]. This result is very strong, as it holds even for regular HMSCs. A weaker notion of realizability called *safe realizability* was also proposed [11,94,95]. Safe realizability of an MSC language L asks that for every MSC M obtained as a combination of projections of MSCs in L, there exists a MSC $M' \in L$ such that M is a prefix of M'. In terms of implementation, this is equivalent to requiring that the canonical implementation of the considered MSC language L is deadlock free. Safe realizability is decidable (and EXPSPACE complete) for regular and globally cooperative HMSCs [94]. For a finite MSC language L, it can be performed in $O(|L|^2.|\mathcal{P}| + r.|\mathcal{P}|)$, where r is the number of events in MSCs of L [10]. However, safe realizability remains undecidable in the general case.

From a software engineering perspective, these undecidability results are bothering to start building software from HMSCs. Starting from a HMSC specification that satisfies a given property φ, an algorithm synthesizing CFSM, or distributed programs may produce communicating machines that implement all the specified behaviors, plus additional, undesired, behaviors that violate property φ. More bothering, one can not decide in general if the synthesized machines contain unspecified behaviors.

To overcome this problem, the software engineering community has proposed to test the synthesized machines with respect to the original HMSC. Indeed, for a given MSC M over a set of processes $\mathcal{P}$, there exists an algorithm to test if $M \in \mathcal{F}_H$ that runs in $O(|H|.|M|^{|\mathcal{P}|})$ [12]. So the approach proposed for instance in [136] to eliminate implied scenarios is to use the following loop:

1. Synthesize a canonical implementation $\mathcal{A}_H$ from H
2. run $\mathcal{A}_H$ to obtain an MSC $M \notin Pref(\mathcal{F}_H)$
3. if some implied MSC M was found, add it to H and go to 2
4. if not, then stop, $\mathcal{F}(\mathcal{A}_H) = Pref(\mathcal{F}_H)$

The objectives of such simulation loops is to change the original specification in such a way that at the end of the algorithm, $Pref(\mathcal{F}_H) = \mathcal{F}(\mathcal{A}_H)$. However, following the undecidability results above, one can not decide whether this equality holds. Hence, in general, this loop is only a semi-algorithm. Other tricky issues are

that finding $M \notin \mathcal{F}_H$ is not a tractable problem in general, except in the considered H is regular (this is the solution proposed in [135]). Indeed, finding such M amounts to solving the realizability question. The last tricky issue it how to add M to an existing specification. Of course, one can easily add a new branch for M in an existing HMSC, but there might be an infinite number of implied scenarios for a given HMSC H. Even worse, the implied semantics is not necessarily representable as another HMSC. So, closure by addition of implied behavior works only in very restricted contexts.

In the rest of this chapter, we will adopt a different approach. We will restrict synthesis to sub-classes of HMSCs for which correct synthesis is guaranteed, or at least can be achieved by slight adaptations of the canonical target model.

4 Controlled implementation of HMSCs

In this section, we revisit the problem of program synthesis from specifications described with local HMSCs. We assume as synthesis scheme the usual algorithm that builds one machine per process by projecting the original HMSC on this process (the *canonical implementation* of H).

We show that in the subclass of local HMSCs, differences that arise between a HMSC and its canonical implementation are due to loss of ordering among messages. We then show that the exact set of behaviors specified by a local HMSC can be preserved by addition of communication controllers, that intercept messages to add stamping information before resending them, and deliver messages to processes in the order described by the specification.

Note that in a very general setting, restricting a technique to a syntactic subclass of HMSCs to obtain decidability is acceptable from a software engineering point of view only if the considered subclass is reasonably expressive. Restricting to bounded HMSCs [14], for instance, reduces the expressive power of HMSCs to that of finite automata, which is in general too restrictive for the kind of asynchronous applications that are usually modeled with scenarios. Our goal in this section is to propose an implementation mechanism for a class of HMSCs that is general enough. Canonical implementation by projection has been proved correct for subclasses of HMSCs : *reconstructible* HMSCs of [79] is a subclass of local HMSCs that impose restrictions on the way loops and choices are used. The solution in [56] starts form local-choice HMSCs, and derives a distributed implementation but with potential deadlocks.

Other implementation techniques implement exactly the language of a bounded HMSC, but with deadlocks [111], or avoid deadlocks but need several initial states in the synthesized machines [19]. In our opinion, deadlocks should not be allowed in an implementation of an HMSC: it amounts to deciding at runtime that an ongoing execution is not valid, as it does not belong to the specification. For this reason, the definition of CFSM in previous section does not contain accepting states. In fact, we do not question correctness of synthesis techniques that use deadlocks, but rather practical use of deadlocking CFSMs. Considering the synthesized CFSM as a program that should run on a network means in general that any sequence on actions of the implementation is part of the implemented behavior, while deadlocked runs are simply removed from the semantics of a CFSM. brings a different light to synthesis.

There are solutions to invalidate a distributed run of a system at runtime (hence implement an equivalent of non-terminating runs removal in the theoretical model), but it is a difficult task, implemented by complex compensation and backtracking mechanisms. Clearly, such techniques are out of the scope of this chapter.

Coming back to the definitions of realizability of this chapter, this means that the correct notion of realizability should be *safe realizability* (which is undecidable in general). We hence consider that a distributed system implements an HMSC if the *prefix closure* of the behaviors of the original specification and of the synthesized machines are identical.

In this section, we propose an implementation mechanism for local message sequence charts, that is HMSC specifications that do not require distributed consensus to be implementable. The proposed technique is to project an HMSC on each process participating to the specification, without additional contents to message. It is well known that this solution produces programs with more behaviors than in the specification [79]. Discrepancies between the original and synthesized model can be avoided by adding message content to synthesized automata, that avoid any ambiguity when a local machine has to take a decision. Indeed, for local-choice specification, the first process to take a decision can choose a particular accepting path of the original HMSC, and this decision is then attached to all messages. All processes conform to this initial choice. Even if correct, this solution has many drawbacks: first, the possible number of tags attached to messages in not bounded, and the resulting model can hence be considered as a CFSM with infinite state space (each CFSM starting with an unbounded number of transitions fixing the chosen path). Second, even if this techniques can be used to guarantee language equality, choices are anticipated. Such solution of *anticipated choices* is not acceptable from a practical point of view, for instance if choices model users decision. Hence, we will mainly consider solutions for which choices are performed *online* during the execution of the CFSM.

Hence, the solution proposed hereafter respects the architecture of the original HMSC, works with finite state machines (but with infinite sets of tags for messages), and avoids deadlocks. The projections of the original HMSCs are composed with local controllers, that intercept messages between processes and tag them with sufficient information to avoid the additional behaviors that appear in the sole projection. The main result of this work is that the projection of the behavior of the controlled system on events of the original processes is equivalent (up to a renaming) to the behavior of the original HMSC.

4.1 The synthesis problem

The objective of the synthesis algorithm from an HMSC H is to obtain a CFSM $\mathcal{A}$ that behaves exactly as H. Let $S(.)$ be a synthesis algorithm that takes as input a HMSC and outputs a CFSM. We will say that $S(.)$ is *correct* if for every HMSC H we have

$$Pref(\mathcal{F}_H) = \mathcal{F}(S(H))$$

.

We have assumed some restrictions on the scenarios that we implement. Some of them are introduced for the sake of readability, and some of them are essential

to ensure a solution to the synthesis problem. We consider that HMSCs are deterministic, and that two MSCs labeling distinct transitions of a local HMSC start with distinct messages. We use this assumption to differentiate branches at runtime. We could achieve a similar result by introducing additional tags during synthesis. However, this mild restriction simplifies the notations and proofs.

MSCs also allow behaviors with *message overtaking*, i.e. in which messages are not received in the order of their emission. In this paper, we consider only FIFO architectures as a target for synthesis. This is hence a natural restriction to consider that all MSCs are FIFO. However, synthesis technique could easily be adapted to allow overtaking. This requires some modifications to the communication architecture, to allow automata to look at the whole contents of their buffers rather that consuming the first incoming message. Such semantics exists for instance in extended automata models such as SDL.

An obvious solution is to project the original HMSC on each instance, that is if H is defined over a set of instances $\mathcal{P}$, we want to build a CFSM $\mathcal{A}_{\mathcal{H}} = \parallel_{i \in \mathcal{P}} A_i$ such that $Pref(\mathcal{F}_H) = \mathcal{F}(\mathcal{A}_H)$.

The principle of projection is to copy the original HMSC on one particular instance, and to remove all the events that do not belong to the considered instance. This operation preserves the structure of the HMSC automaton: Starting from an automaton labeled by MSCs, we obtain an automaton labeled by (possibly empty) sequences of events located on the considered instance. This object can be considered as a finite state automaton by adding intermediary states in sequences of events (recall that projections of an HMSC language of a single process is a regular language). Empty transitions can be removed by the usual ε-closure procedure for finite state automata (see for instance chapter 2.4 of [80]).

Definition 67 (Projection) *Let us consider an HMSC $H = (N, \to, \mathcal{M}, n_0)$. The set of events of a MSC M is denoted by E_M, and the set of events of M located on instance i by E_{Mi}. The set E_{Mi} is totally ordered by $\leq_i$. We denote its elements by $e_1, \cdots, e_{|E_{Mi}|}$. The finite state automaton A_i, result of the projection of H onto the instance i is $A_i = (Q_i, \to_i, E_i \cup \{\varepsilon\}, n_0)$. We encode states of A_i as tuples $(n, M, n', i) \in N \times \mathcal{M} \times N \times \mathbb{N}$, the first three components designating an HMSC transition and the last one designating the index of the last event executed by A_i in M during this transition, or simply as a reference to an HMSC node n (designating configuration in which A_i has not yet started the execution of a MSC from n). We then have $Q_i = \{n\} \cup \{(n, M, k, n') \mid (n, M, n') \in \longrightarrow \wedge k < |E_{M_i}|\}$, and $E_i = \bigcup_{M \in \mathcal{M}} E_{Mi}$. We can then define the transition relation $\longrightarrow_i$ as*

$$
\begin{array}{rl}
\longrightarrow_i & = \{(n, \varepsilon, n') \mid \exists (n, M, n') \in \longrightarrow \wedge |E_{M_i}| = 0\} \\
\cup & \{(n, t(e_1), n') \mid \exists (n, M, n') \in \longrightarrow \wedge |E_{M_i}| = 1\} \\
\cup & \{(n, t(e_1), (n, M, n', 1)) \mid (n, M, n') \in \longrightarrow \wedge |E_{M_i}| \geq 2\} \\
\cup & \{((n, M, n', k-1), t(e_k), (n, M, n', k)) \mid (n, M, n') \in \longrightarrow \wedge 2 \leq k < |E_{M_i}|\} \\
\cup & \{((n, M, n', k-1), t(e_k), n') \mid (n, M, n') \in \longrightarrow \wedge k = |E_{M_i}|\}
\end{array}
$$

Note that the synthesized machines do not use additional message contents, and communicate only using events that were originally specified in the original HMSC. The synthesis by projection from the HMSC of Figure 7.4 produces the CFSM of Figure 7.5. Note that as instance D is not active in MSC $M1$, there is an ε-transition in the automaton associated to D.

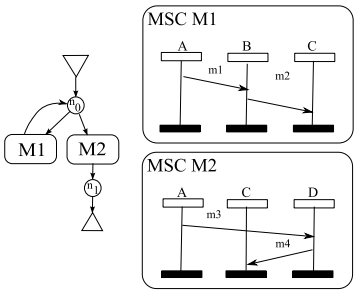


Figure 7.4: An example of HMSC

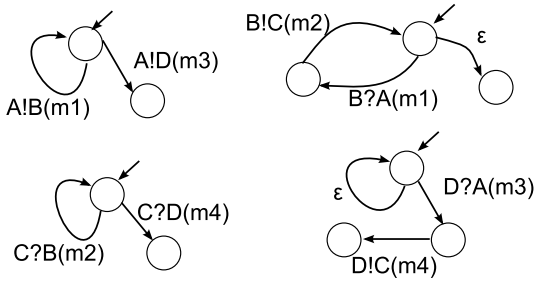


Figure 7.5: The instance automata projected from the HMSC of Fig. 7.4.

4.2 implementation problems

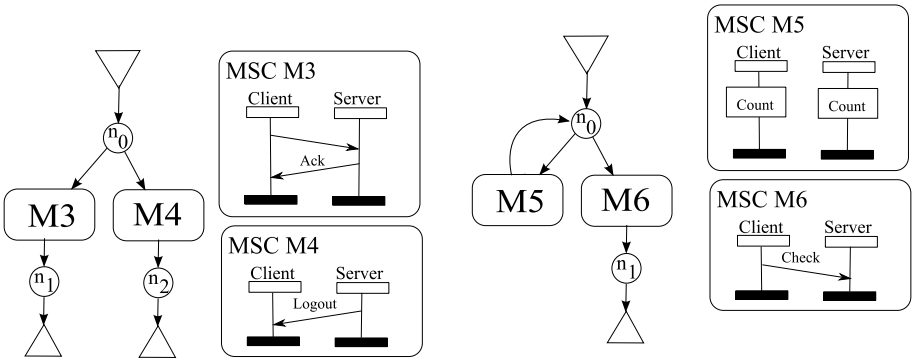


Figure 7.6: An example of non-local HMSC *a*) and a non-gc HMSC *b*)

We can now state the main difficulty when moving from HMSCs to local machines. In an HMSC, the possible executions are built by concatenating MSCs one after another. Hence in an execution of an HMSC, all processes conform to a single sequence of MSCs collected along a path. In a CFSM setting, when two processes have to take a decision to perform scenario M_1 or M_2, they can of course take concurrently the same decision, but conversely, one instance can decide to perform scenario M_1 while the other instance decides to perform M_2. Consider for instance the HMSC of Figure 7.6-a. The synthesis from the HMSC of Figure 7.6-a) produces the CFSM of Figure 7.1. In this model, the CFSM can behave as specified in scenarios M_1 and M_2. However, A_{client} can also decide to send a *Data* message while A_{Server} sends a logout message. This situation was not specified in the HMSC of Figure 7.6-a)(it is an implied scenario), and can lead to a deadlock of the system. So, the CFSM of Figure 7.1 cannot be considered as a correct implementation. The

HMSC in Figure 7.6-b also raises problems. According to the specification, processes *Client* and *Server* must execute the same number of local action "'count"' before exchanging message *Check*. Obviously, this last specification is not local nor globally cooperative.

In general, the projection of an HMSC on its instances can define more behaviors than the original specification, but can also deadlock. Hence, synthesis by projection on instance is not correct for any kind of HMSC. It was proved in [79] that the synthesized language contains all runs of the HMSC specification. One may think that using globally cooperative and local HMSCs only, all implementation problems are solved. However this is not the case, as show in the following theorem.

Theorem 41 ([79]) *Let H be an HMSC and let $\mathcal{A}$ be the CFSM obtained by projection of H on its instances. Then $Pref(\mathcal{F}_H) \subseteq \mathcal{F}(\mathcal{A}_H)$. Furthermore, there exist local HMSCs such that $Pref(\mathcal{F}_H) \subset \mathcal{F}(\mathcal{A}_H)$.*

In the rest of the paper, we will only consider local HMSCs. However, this is not sufficient to ensure correctness of synthesis. Let us consider the projection of H in Figure 7.4 on all its instances given in Figure 7.5. A correct behavior of H is shown in Figure 7.7-a), while a possible but incorrect behavior of the synthesized automata is shown in Figure 7.7-b). We can see that message m_2 sent by machine B to machine C can be delayed and arrive later than message m_4 from machine D, hence mixing two different scenarios. Machine C does not have enough information to decide to delay the reception of m_4, and hence may violate the HMSC semantics.

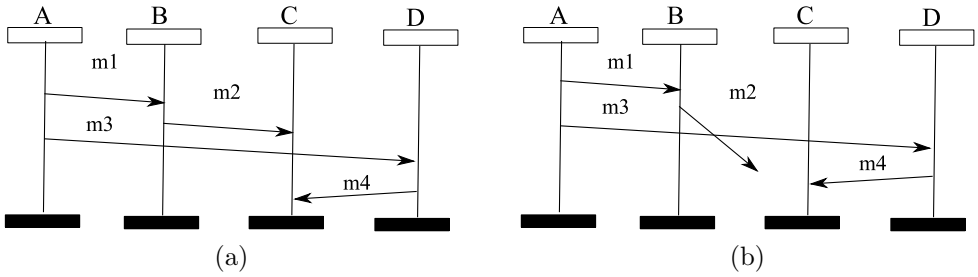


Figure 7.7: *a*)A correct behavior of the HMSC of Fig. 7.4, and *b*) a possible distortion due to the loss of information on projected instances.

This example proves that in general, even for local HMSCs, the synthesis by projection is not correct. Problems arise when an instance does not have enough information on the sequences of choices that have occurred in the causal past of a message reception event. In some sense, the projection of an HMSC on local components breaks the global coordination between deciding instances and the other instances in the system. In [79], a subclass of local-choice HMSCs for which synthesis by projection is correct was identified. This class is called *reconstructible HMSCs*.

Definition 68 *Let H be a local HMSC and c be a choice node of H. Let ρ be a cyclic path starting from c, and ρ' be any acyclic path starting from c. Let H_c be the HMSC with two nodes c, c', two transitions (c, O_ρ, c) and (c, O'_ρ, c'). Let $\mathcal{A}_c$ be the CFSM obtained by projection from H_c. We will say that c, ρ, ρ' is a* sequence-loss witness *iff $Pref(\mathcal{F}_{H_c}) \neq \mathcal{F}(\mathcal{A}_c)$.*

We will say that an HMSC is *reconstructible* if and only if it is local and has no sequence-loss witnesses. One does not have to simulate all runs of communicating automata in $\mathcal{A}_c$ to detect that $\mathcal{L}(H_c) \neq \mathcal{L}(\mathcal{A}_c)$. Reconstructibility of a HMSC H can be decided as a simple criterion on simple cycles and acyclic paths of H, and is in co-NP. Indeed, sequence losses can be detected by checking if the sequential ordering of events along a non-deciding instance in prefix $O_\rho \circ O_{\rho'}$ can be lost during projection. To avoid technical details, will not show in this paper how the sequence losses can be found from $O_\rho \circ O_{\rho'}$, but rather illustrate the approach on an example. We refer interested readers to [79] for formal details. Let us consider the example of Figure 7.4, with a single choice node n_0, and the path $(n_0, M_1, n_0).(n_0, M_2, n_1)$. According to the semantics of HMSCs, reception of messages m_2 and m_4 on instance C should occur in this order in a correct implementation of the example. Now let us consider the automata obtained by projection of H on instances, as in Figure 7.5. After executing $A!B(m1).B?A(m1).B!C(m2).A!D(m3).D?A(m3).D!C(m4)$, the CFSM is in configuration $(L = q_{1,A}.q_{0,B}.q_{0,C}.q_{2,D}, W = \{\varepsilon, \ldots w_{BC} = m2, w_{DC} = m4, \ldots \varepsilon\})$. From this configuration, the automaton corresponding to instance C can receive $m2$, which is the expected behavior, or conversely receive $m4$ which is wrong according to the choices that were performed by instance A. Hence $n_0, (n_0, M_1, n_0), (n_0, M_2, n_1)$ is a sequence loss witness. This can easily be seen from $M_1 \circ M_2$: If one removes the ordering between the reception of $m2$ and the reception of $m4$, there is no way to infer this ordering from remaining causalities.

One important fact is that synthesis by projection is correct for the subclass of reconstructible HMSCs.

Theorem 42 ([79]) *Let H be a local and reconstructible HMSC, and $\mathcal{A}_H$ be the CFSM obtained from H by projection. Then, $Pref(\mathcal{F}_H) = \mathcal{F}(\mathcal{A}_H)$.*

According to theorem 42, the communicating automata synthesized from reconstructible HMSCs are correct implementations. However, we show in the next section, that all local HMSCs can be implemented with the help of additional controllers. This allows for the following synthesis approach: first check if an HMSC is reconstructible. If the answer is yes, then synthesize the CFSM by simple projection as proposed in section 4.1. If the answer is no, then synthesize the CFSM with their controllers, as proposed in section 4.3.

4.3 Implementing HMSCs with message controllers

The class of reconstructible HMSCs shown in section 4.1 is contained in the class of local HMSCs. This subclass is quite restrictive (for instance, the HMSC of Figure 7.4 is not reconstructible, and hence can not be implemented by a simple projection). Note also that the difference between the languages of an HMSC and of the synthesized machines comes from the fact that some communicating automata consume a wrong message instead of waiting for the arrival of the message specified by the HMSC. This problem is clearly avoided if a path of the implemented HMSC is chosen from the initial configuration, but as already explained, this solution is not satisfactory from a practical point of view if one sees semantics of choices as a decision that is taken at the last moment (i.e. when a minimal event on a branch is executed).

A solution to avoid discrepancies between a local HMSC specification and its canonical implementation is to instrument the synthesized machines with controllers. We associate a local controller to each communicating machine that can tag messages and delay their delivery. As synthesis fails because of reception of messages in the wrong order, each controller will receive messages destinated to the machine it controls, and decide whether it should deliver it to the controlled machine, or delay its delivery. This decision is taken depending on additional information carried by messages, namely a vector clock. Vector clocks is a well known mechanism, and helps keeping track of global progress in distributed systems.

This new mechanism allows for the implementation of *any* local HMSC H, without syntactic restriction, and with online choices. The architecture is as follows: For each process, we compute an automaton, as shown in previous section by projection of H on each of its instances. The projection is the same as previously, with the slight difference that the synthesized automaton communicates with its controller, and not directly with other processes. To differentiate, we will denote by $K(A_i)$ the "controlled version" of A_i, keeping in mind that A_i and $K(A_i)$ are isomorphic machines. Then, we add to each automaton $K(A_i)$ a controller C_i, that will receive all communications from $K(A_i)$, and tag them with a stamp. In every automaton $K(A_i)$ we replace each transition of the form $\big((n_1, M_1, k, n_2), p!q(m), (n_3, M_2, k', n_4)\big)$ (respectively $\big((n_1, M_1, k, n_2), p?q(m), (n_3, M_2, k', n_4)\big)$) in A_i, by a transition of the form $\big((n_1, M_1, k, n_2), p!C_p(q, m, b), (n_3, M_2, k', n_4)\big)$ (respectively $\big((n_1, M_1, k, n_2), p?C_p(q, m, b), (n_3, M_2, k', n_4)\big)$), where b indicates the branch to which the sending or the reception belongs. A controller C_i can receive messages of the form (q, m, b) from his controlled process $K(A_i)$. In such cases, it tags them with a clock (the contents of this clock is defined later in this section), and sends them to controller C_q. Similarly, each controller C_i will receive all tagged messages destinated to $K(A_i)$, and decide with respect to its tag whether a message must be sent immediately to $K(A_i)$ or delayed (i.e. left intact in buffer). Automata and their controllers communicate via FIFO channels, which defines a total ordering on message receptions or sendings. Controllers also exchange their tagged messages via FIFO buffering. In this section, we first define the distributed architecture and the tagging mechanism that will allow for preservation of the global specification. We then define control automata and their composition with synthesized automata. We then show that for local HMSCs the controlled local system obtained by projection simulates the original specification.

Distributed architecture

We consider the $n = |\mathcal{P}|$ automata $\{K(A_i)\}_{1 \leq i \leq n}$ obtained by projection of the original HMSC on the different instances, and a set of controllers $\{C_i\}_{1 \leq i \leq n}$. Each communicating automaton $K(A_i)$ is connected via a bidirectional FIFO channel to its associated controller C_i. The controllers are themselves interconnected via a complete graph of bidirectional FIFO channels. We will refer to these connections among communicating automata as *ports*. A machine $K(A_i)$ communicates with its controller via a port P, and for all $i \neq j$, port P_j of controller C_i is connected to the port P_i of controller C_j. Note that even if machines share common port names, for every instance name k, port P_k of a controller C_i and port P_k of a controller C_j designate different physical connections. This architecture is illustrated in Figure 7.8

for three processes i, j, k. This architecture is quite flexible: All the components run asynchronously and exchange messages, without any other assumption on the way they share resources, memory or processors. The implementation described hereafter was successfully implemented using Promela code, but also with a JAVA+REST implementation on very distributed or centralized architectures mapping processes on physical machines.

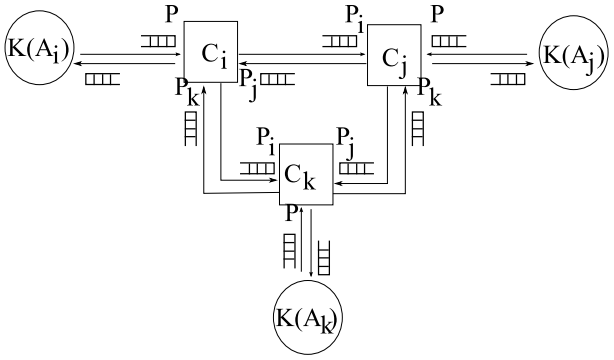


Figure 7.8: The distributed controlled architecture.

Tagging mechanism

Vector clocks are a standard mechanism to record faithfully executions of distributed systems (see for instance [48, 100]), or to enforce some ordering on communication events [124]. Usually, vector clocks count events that have occurred on each process. In the controlled synthesis architecture, each controller maintains a vector clock that counts the number of occurrences of each branch of an execution of the original HMSC it is aware of.

To allow for faithful recording of branches chosen along an execution we have to set up a total ordering on branches of HMSCs. Let H be an HMSC. We will denote by $\mathcal{B}_H$ the branches of H, and fix an arbitrary total ordering $\lhd$ on $\mathcal{B}_H$. We use this arbitrary order on branches to index integer vectors that remember the number of occurrences of branches that have occurred during an execution of an HMSC. Let us consider the example of Figure 7.4, that contains two branches $b_1 = (n_0, M_1, n_0)$ and $b_2 = (n_0, M_2, n_1)$. We can fix $b_1 \lhd b_2$, and associate to every execution a vector τ of two integers, where $\tau[b_i], i \in 1, 2$ represents the number of occurrences of branch b_i in the execution.

Definition 69 (Choice clocks) *A* choice clock *of an HMSC H is a vector of $\mathbb{N}^{\mathcal{B}_H}$. Let $\rho = n_0 \xrightarrow{M_1} n_1 \xrightarrow{M_2} n_2 \ldots \xrightarrow{M_k} n_k$ be a path of H. The* choice clocks labeling *of O_ρ is a mapping $\tau : E_{O_\rho} \longrightarrow \mathbb{N}^{\mathcal{B}_H}$ such that for every $i \in 1..k, e \in M_i$, $\tau(e)[b]$ is the number of occurrences of branch b in $M_1 \circ \cdots \circ M_i$.*

Intuitively, choice clocks count the number of occurrences of each choice in a path of H. In the rest of this section, we will show that communicating automata and their controllers can maintain *locally* a choice clock along the prefix that they are executing, and that choice clocks carry all the needed information to forbid the execution of prefixes that are not in $\mathcal{L}(H)$.

The usual terminology and definitions on vectors apply to choice clocks. A vector V_2 is an *immediate successor* of a vector V_1 of same size, denoted $V_1 \lessdot V_2$, if there is a single component b such that $V_1[b]+1 = V_2[b]$, and $V_1[b'] = V_2[b']$ for all other entries b'. We will say that vectors V_1 and V_2 are equal, denoted $V_1 = V_2$, if $V_1[b] = V_2[b]$ for every entry b. We will say that V_2 is greater than V_1, denoted $V_1 \prec V_2$, iff $V_1[b] = V_2[b]$ for some entries b, and $V_1[b] < V_2[b]$ for all others.

For a given path $\rho = n_0 \xrightarrow{M_1} n_1 \xrightarrow{M_2} n_2 \ldots \xrightarrow{M_k} n_k$, we will call the *choice events* of O_ρ the minimal events in every $M_i, i \in 1..k$. It is rather straightforward to see that when an HMSC H is local, then for every path ρ of H, the set of choice events in O_ρ is totally ordered. Note also that for a pair of events e, f in O_ρ, $\tau(e) = \tau(f)$ if and only if e, f belong to the same MSC M_i. From these facts, the following proposition is straightforward:

Proposition 11 *Let H be a local HMSC, ρ be a path of H, and τ be the choice clock labeling of O_ρ. Then, $(\tau(E_{O_\rho}), \prec)$ is a totally ordered set.*

This proposition is important: maintaining locally a consistent tagging of messages allows a controller that has two tagged messages available in two of its buffers to decide which one should be delivered first. Unlike in the solution of [56], we allow MSCs labeling H to be defined over arbitrary sets of processes. Hence, some processes may be inactive is a branch.

Definition 70 (Concerned instances) *Let $b = (c, M, n)$ be a branch of an HMSC H. We will say that instance $p \in \mathcal{P}$ is* concerned *by branch b if and only if there exists an event of M on p ($E_{Mp} \neq \emptyset$). Let $K \in \mathbb{N}^{\mathcal{B}_H}$ be a choice clock, and let $p \in \mathcal{P}$ be an instance of H. The vector of choices that concern p in K is the restriction of K to branches that concern p, and is denoted by $[K]_p$.*

In the example in Figure 7.4, the choice clock is a integer vector indexed by b_1, b_2, where $b_1 = (n_0, M_1, n_0)$ and $b_2 = (n_0, M_2, n_1)$. In this example, instances A, C are concerned by both branches (they are active in M_1 and M_2), but instance C is concerned only by b_1 and instance D is concerned only by b_2.

For a given instance i, the controller C_i associated with the projected automaton $K(A_i)$ will receive the messages sent by $K(A_i)$ and by the other controllers. Messages exchanged between the automata and the controllers are triples (j, m, b) where $j \in \mathcal{P}$ is the destination automaton, $m \in C$ is the message name, and b the branch in which the sending event has occurred. In other words, in our controlled architecture, an automaton executes $p!C_p(q, m, b)$ instead of $p!q(m)$. The messages exchanged between controllers are tagged and represented by pairs (m, τ) where m is a message name and $\tau \in \mathbb{N}^{\mathcal{B}_H}$ a choice vector. In addition, the controller C_i maintains several local variables:

- $\tau_i \in \mathbb{N}^{\mathcal{B}_H}$, its *locally known choices* vector. It is initialized to the null vector, and updated upon consumption of incoming messages.
- $numEvt$, which counts the remaining number of communication events of the instance i to be treated in the current branch that is being processed.

- *Rec* is a sequence of reception events. *numEvt* and *Rec* are initialized with constant values (that depend on the chosen branch) when dealing with the first event of a branch on process i.

- *currentb*, which memorizes the branch of H that is currently executed by process i.

Let us denote by $\pi_i(M)$ the sequence of events obtained by projection of M on instance $i \in \mathcal{P}$, and by $\pi_{i,?}(M)$ the restriction of this sequence to receptions. For a sequence of events w, we will denote by $tail(w)$ the sequence of events obtained by removing the first event from w, that is if $w = a.v$, then $tail(w) = v$. The generic algorithm for a controller C_i is composed of two rules, which are always active (see Algorithm 2). Rule 1 applies to communications from $K(A_i)$ to C_i. First case corresponds to minimal events controlled by the projected automaton $K(A_i)$. When dealing with the first event of the MSC (branch b) to be processed, the only role of the controller is to compute the tag (increment of the corresponding component of τ_i) and to initialize the variables *numEvt* and *Rec*. Consuming a message from A_i is always allowed, as it may not cause wrong behaviors. The currently processed branch is stored in variable *currentb*. The other case deals with communications from $K(A_i)$ that are not choices of $K(A_i)$. These events are generated in correct order by construction of the projection.

The second rule applies for **every port** $P_j, j \neq i$, and aims at controlling the order of the different receptions of messages arriving in the buffers between controller i and all other controllers. Note that these messages arrive in a distinct buffer for each neighbor controller. As we have seen for HMSCs that are not reconstructible, consuming messages in an unspecified order produces unspecified behaviors. Hence, a message will be consumed only when its tag shows that this is the next message to be received. Tags verification is the main objective of the controller. There are three cases:

- The first case occurs when a branch of H has already been started, that is a controller C_i has received (i.e. consumed) a message indicating the choice performed by the deciding instance of this branch, and a valid message arrives. In this situation, all the components concerning $K(A_i)$ of the current tag τ_i and of the tag τ labeling the incoming message must be equal, and this incoming message must be the next expected message (i.e. the next reception in *Rec*) in the currently executed branch. Then the message can be consumed by C_i and forwarded to $K(A_i)$. The fact that there is only one FIFO channel between the controller C_i and the projected automaton $K(A_i)$ ensures the correct order of receptions on this automaton.

- The second case is when the incoming message is the first communication signaling a new choice. The controller then checks if the received message defines the next branch of H that must be executed by $K(A_i)$. This is done by verifying if the received tag is the next tag to be treated (considering only the components that concern $K(A_i)$), that is $[\tau_i]_i \lessdot [\tau]_i$. In that case, the current tag can be updated. The current branch is retrieved by considering the component that differs between $[\tau]_i$ and $[\tau_i]_i$. Then the remaining number of events that should be executed within this branch (the number of events on

the instance i in the MSC of the current branch, minored by one) is set, as well as the expected sequence of receptions, before transmission of the message to $K(A_i)$.

- The third case applies when none of the above situations hold, that is the incoming message on port P_j can not yet be consumed, either because it is not the next reception expected (another reception on another port should occur before this one) or the incoming message signals that a new branch has been started, but more events from formerly chosen branches must occur before consuming it. In such case, the controller does nothing, and waits for other messages on other ports.

The algorithm 2 executed by every controller is presented next page.

Now that we have defined controlled automata and their controllers, we can define formally how they compose. Recalls that $K(A_i)$ is a finite state machine with the same states as A_i, but in which each transition $(q, i!j(m), q')$ is replaced by a transition $(q, i!C_i(j, m, b), q')$ (where b denotes the names of the branch currently executed by A_i, and each transition $(q, i?j(m), q')$ is replaced by a transition $(q, i?C_i(j, m), q')$. Each controller C_i is not a communicating automata, but yet it is a machine that sends and receives messages. The composition $K(A_i) \mid C_i$ of a machine with its controller is a pair of communicating machines with a FIFO buffer from $K(A_i)$ to C_i, and another from C_i to $K(A_i)$. Then, the composition of controlled machines $\parallel_{i \in I} K(A_i)|C_i)$ is the union of all $K(A_i)|C_i$, with communication buffers from each C_i to each C_j, for $i \neq j$ in $\mathcal{P}$. Note that $K(A_i)'s$ communicate only with their controllers. This composition is illustrated in Figure 7.8, where the depicted architecture is $\big(K(A_i) \mid C_i\big) \parallel \big(K(A_j) \mid C_j\big) \parallel \big(K(A_k) \mid C_k\big)$. At this point, let us note that our controlled implementation is not a CFSM anymore. However, our controllers, which are currently defined with several lines of code, could be easily be defined as communicating finite state automata with a communication policy allowing to read messages without consuming them, and with variables. All these features exist for instance in SDL [83], in which variables are allowed, and for which the *save* construct allows to read messages without consumption. Last, note that adding controllers to our synthesis architecture does not really increase the expressive power of the machines, as CFSMs can already simulate Turing machines.

We can now show correctness of the controlled synthesis. Of course, adding controllers to the system means adding the controllers' actions to the executions. Hence, we can not require that $Pref(\mathcal{F}_H) = \mathcal{F}(\parallel_{i \in I} K(A_i)|Ci)$ anymore. An adequate notion of correctness should abstract of controllers. As communications among processes in the controlled architecture are implemented via controllers, we can easily define a renaming function that replaces a communications (q, m, b) from a process p to his controller by a communication of a message m from p to q (and similarly for receptions).

Theorem 43 *Let H be an HMSC, and let $\mathcal{A}^{cont} = \parallel_{i \in I} K(A_i)|C_i$ be its controlled synthesis. Then, $\mathcal{A}^{cont}$ simulates H (up to renaming).*

The detailed proof can be found in the appendix.

Algorithm 2 Controller C_i

RULE 1: when (j, m, b) available on port P_i
/* There is a message from $K(a_i)$ ***in the buffer from*** $K(a_i)$ ***to*** C_i ****/***
consume (j, m, b)
if $numEvt = 0$ **then**
  $\tau_i[b]$++
  $numEvt := |\Pi_i(M_b)| - 1$
  $Rec = \Pi_{i,?}(M_b)$
  send (m, τ_i) on port P_j
else
  $numEvt$ - -
  send (m, τ_i) on port P_j
end if

RULE 2: when there exists a port P_j with (m, τ) available on port P_j
/* There is a message from controller C_j ***in the buffer between*** C_j ***and*** C_i ****/***
if $([\tau_i]_i = [\tau]_i) \wedge (Rec = A_i?A_j(m).w)$ **then**
  /* continuation of an already started branch */
  consume (m, τ)
  $numEvt$ - -
  send (j, m) on port P_i
  $Rec = w$
else
  if $(numEvt = 0) \wedge ([\tau_i]_i \lessdot [\tau]_i)$ **then**
    /* A new branch b ***was started, and this is the next */***
    /* branch that A_i ***should execute (***i ***is concerned by*** b***)*/***
    consume (m, τ)
    $\tau_i := \tau$
    $currentb := b$ s.t. $[\tau][b] - [\tau_i][b] \neq 0$
    $numEvt := |\Pi_i(M_{currentb})| - 1$
    $Rec := tail(\Pi_{i,?}(M_{currentb}))$
    send (j, m) on port P_i
  end if
  /* The last situation is when the message can not be consumed because it does not have the right sequence number */
end if

5 Realizability of Dynamic MSC Grammars

We have introduced a dynamic model of MSCs in chapter 6, and a natural question is whether this model can be implemented. Dynamicity raises several particular issues. First, the implementation model should be able to create new threads, which is not the case of standard CFSMs. Second, process identities are not known a priori, and defined at runtime. As a process can only send a message to another thread if it knows it identity (or equivalently its address), the implementation model should take into account the knowledge each thread has about its neighbors. Furthermore,

some scenarios that seem implementable may not be realizable as one thread has no way to know the exact identity of a neighbor to whom it is supposed to send a message.

In this section, we first define an implementation model for Dynamic Message sequence Charts, and then show that the addressing problems in a MSC grammar specification can be detected.

5.1 Dynamic Communicating Automata: an implementation model for MSC grammars

Dynamic communicating automata (DCA), as introduced in this section, extend classical communicating finite-state machines [32]. They allow for the dynamic creation of processes, and *asynchronous* FIFO communication between them. Each thread p holds a set of process variables, that denotes the set of processes that p remembers at a given time. Their values represent process identities and allow a thread to communicate with them. This model is close to the threading mechanism in programming languages such as JAVA and Erlang, but also borrows elements of the routing mechanisms in protocols implemented over partially connected mesh topologies. Threads will be represented by dynamically created copies of the same automaton. At creation time, the creating thread will pass known process variables to the created thread. A thread can communicate with another one if both threads know each other, i.e., they have kept their identities in memory.

Definition 71 *A* dynamic communicating automaton *(or simply DCA) is a tuple* $\mathcal{A} = (X, Msg, Q, \Delta, \iota, F)$ *where*

- X *is a set of* process variables*,*
- Msg *is a set of* messages*,*
- Q *is a set of* states*,* $\iota \in Q$ *is the* initial state*,*
- $F \subseteq Q$ *is the set of* final states*, and*
- $\Delta \subseteq Q \times Act_{\mathcal{A}} \times Q$ *is the set of* transitions*.*

Here, $Act_{\mathcal{A}}$ *is a set of* actions *of the form* $x \leftarrow \mathsf{spawn}(s, \eta)$ *(*spawn action*),* $x\,!\,(m, \eta)$ *(*send action*),* $x\,?\,(m, Y)$ *(*receive action*), and* $\mathsf{rn}(\sigma)$ *(*variable renaming*) where* $x \in X$*,* $s \in Q$*,* $\eta : (X \uplus \{\mathsf{self}\})^X$*,* $\sigma : X \mathrm{to} X$*,* $Y \subseteq X$*, and* $m \in Msg$*.*
We say that $\mathcal{A}$ *is* finite *if* X*,* Msg*, and* Q *are finite.*

Figure 7.9 is an example of DCA, with sets of process variables $X = \{x_1, x_2, x_3\}$ (every process will have its own copy of X), messages $Msg = \{m\}$, states $Q = \{s_0, \ldots, s_6\}$ where s_0 is the initial state, final states $F = \{s_3, s_4, s_6\}$, and transitions, which are labeled with actions. Each process associates with every variable in X the identity of an existing process. We will denote by $proc(p)[x]$ the value (process identity) that process p associates with variable $x \in X$.

Let us detail on an example how DCA work. At the beginning, a single process exists, say 1. Moreover, all process variables are bound to 1, i.e., $(x_1, x_2, x_3) =$

$(1,1,1)$. When process 1 moves from the initial state to s_1, it executes $x_1 \leftarrow \mathsf{spawn}(s_0,(1,1,1))$, which creates a new process, say 2, starting in state s_0. In the creating process, x_1 is then bound to 2. In process 2, on the other hand, (x_1,x_2,x_3) will have the value $(1,1,1)$. So far, this scenario is captured by the first three events in the MSC M of Figure 6.2. Process 2 itself might now spawn a new process 3, which, in turn, can create a process 4 in which we initially have $(x_1,x_2,x_3)=(3,3,1)$. Now assume that, instead of spawning a new process, 4 moves to s_5 executing $3\,!\,(m,(\mathsf{self},x_2,x_3))$, which sends the message (m,γ) to process 3, with $\gamma=(4,3,1)$. Recall that process 3 is in state s_1 and that $(x_1,x_2,x_3)=(4,2,1)$. Thus, 3 can execute $4\,?\,(m,\{x_1\})$, i.e. receive $(m,(4,3,1))$ and bind x_1 to 4. We then have $(x_1,x_2,x_3)=(4,2,1)$ on process 3. The DCA accepts the behavior M depicted in Figure 6.2.

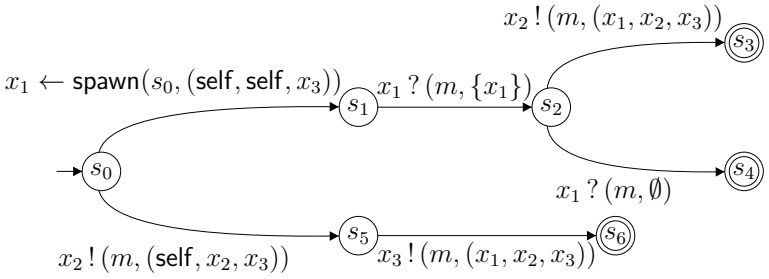


Figure 7.9: A dynamic communicating automaton

We can now formalize the semantics of DCA as a word language over Σ. This language is the set of linearizations of some set of MSCs and therefore yields a natural semantics in terms of MSCs.

Let $\mathcal{A}=(X,Msg,Q,\Delta,\iota,F)$ be some DCA.

A *configuration* of $\mathcal{A}$ is a quadruple $(\mathcal{P},state,proc,ch)$ where $\mathcal{P}\subseteq\mathbb{N}$ is a finite set of active processes (or identities), $state : \mathcal{P}toQ$ maps each active process to its current state, $proc : \mathcal{P}to\mathcal{P}^X$ contains the identities that are known to some process, and $ch : (\mathcal{P}\times\mathcal{P})to(Msg\times\mathcal{P}^X)^*$ keeps track of the channels contents. The configurations of $\mathcal{A}$ are collected in $Conf_{\mathcal{A}}$. We define a global transition relation $\Longrightarrow_{\mathcal{A}} \subseteq Conf_{\mathcal{A}}\times(\Sigma\cup\{\varepsilon\})\times Conf_{\mathcal{A}}$ as follows: For $a\in\Sigma\cup\{\varepsilon\}$, $c=(\mathcal{P},state,proc,ch)\in Conf_{\mathcal{A}}$, and $c'=(\mathcal{P}',state',proc',ch')\in Conf_{\mathcal{A}}$, we let $(c,a,c')\in\Longrightarrow_{\mathcal{A}}$ if there are $p\in\mathcal{P}$ and $\hat{p}\in\mathbb{N}$ with $p\neq\hat{p}$ (the process executing a and the communication partner or spawned process), $x\in X$, $s_0\in Q$, $\eta:(X\uplus\{\mathsf{self}\})^X$, $Y\subseteq X$, $\sigma : XtoX$, and $(s,b,s')\in\Delta$ such that $state(p)=s$, and one of the cases in Figure 7.10 holds (c and c' coincide for all values that are not specified below a line).

An *initial configuration* is of the form $(\{p\},p\mapsto\iota,proc,(p,p)\mapsto\varepsilon)\in Conf_{\mathcal{A}}$ for some $p\in\mathbb{N}$ where $proc(p)[x]=p$ for all $x\in X$. A configuration $(\mathcal{P},state,proc,ch)$ is *final* if $state(p)\in F$ for all $p\in\mathcal{P}$, and $ch(p,q)=\varepsilon$ for all $(p,q)\in\mathcal{P}\times\mathcal{P}$, i.e., each process is in a final state and all messages have been received yielding empty channels.

A *run* of DCA $\mathcal{A}$ on a word $w\in\Sigma^*$ is an alternating sequence $c_0,a_1,c_1,\ldots,a_n,c_n$ of letters $a_i\in\Sigma\cup\{\varepsilon\}$ and configurations $c_i\in Conf_{\mathcal{A}}$ such that $w=a_1.a_2\ldots a_n$, c_0 is an initial configuration and, for every $i\in\{1,\ldots,n\}$, $(c_{i-1},a_i,c_i)\in\Longrightarrow_{\mathcal{A}}$.[1] The

[1]Here and elsewhere, $u.w$ denotes the concatenation of words u and v. In particular, $a.\varepsilon=a$.

$$\textsf{spawn}\quad \frac{a = \textsf{spawn}(p,\hat{p}) \qquad b = x \leftarrow \textsf{spawn}(s_0,\eta)}{\begin{array}{ll} \mathcal{P}' = \mathcal{P} \uplus \{\hat{p}\} & ch'(q,q') = \varepsilon \\ state'(p) = s' & \text{if } \hat{p} \in \{q,q'\} \\ state'(\hat{p}) = s_0 & \end{array} \qquad \begin{array}{l} proc'(p)[x] = \hat{p} \\ proc'(\hat{p})[y] = \begin{cases} proc(p)[\eta[y]] & \text{if } \eta[y] \neq \textsf{self} \\ p & \text{if } \eta[y] = \textsf{self} \end{cases} \\ \quad \text{for all } y \in X \end{array}}$$

$$\textsf{send}\quad \frac{a = !(p,\hat{p}) \qquad b = x\,!\,(m,\eta) \qquad \hat{p} = proc(p)[x]}{state'(p) = s' \qquad ch'(p,\hat{p}) = (m,\gamma).ch(p,\hat{p})}$$
$$\text{where } \gamma \in \mathcal{P}^X \text{ with}$$
$$\gamma[y] = \begin{cases} proc(p)[\eta[y]] & \text{if } \eta[y] \neq \textsf{self} \\ p & \text{if } \eta[y] = \textsf{self} \end{cases}$$

$$\textsf{receive}\quad \frac{a = ?(\hat{p},p) \qquad b = x\,?\,(m,Y) \qquad \hat{p} = proc(p)[x]}{state'(p) = s' \qquad \begin{array}{l} \text{there is } \gamma \in \mathcal{P}^X \text{ such that} \\ \left[\begin{array}{ll} & ch(\hat{p},p) = ch'(\hat{p},p).(m,\gamma) \\ \wedge & \text{for all } y \in Y\, proc'(p)[y] = \gamma[y] \end{array}\right] \end{array}}$$

$$\textsf{renaming}\quad \frac{a = \varepsilon \qquad b = \textsf{rn}(\sigma)}{state'(p) = s' \qquad \begin{array}{l} proc'(p)[y] = proc(p)[\sigma(y)] \\ \quad \text{for all } y \in X \end{array}}$$

Figure 7.10: Global transition relation of a DCA

run is *accepting* if c_n is a final configuration. The *word language* of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words $w \in \Sigma^*$ such that there is an accepting run of $\mathcal{A}$ on w. Finally, the *(MSC) language* of $\mathcal{A}$ is $\mathcal{F}(\mathcal{A}) := \{M \in \mathbb{M} \mid Lin(poset(M)) \subseteq \mathcal{L}(\mathcal{A})\}$. Figure 7.9 shows a *finite* DCA. It accepts the MSCs that look like M in Figure 6.2.

Note that DCA actually generalize the classical setting of communicating finite-state machines with a fixed number of processes [32]. To simulate them, the starting process of a DCA can first spawn the required number of processes and then broadcast the identity of any process to any other process by sending additional messages.

5.2 Implementability of Dynamic MSC Grammars

Dynamic process creation raises new issues, both in DMGs and in DCAs. Indeed, process identities can only be known at runtime. To communicate, two threads must know each other, in order to allow buffered communications. Now, a DMG may specify that a communication occurs between two threads that can not know each other. Another problem is that supposing a limited number of available variables, an thread may have to forget identities of its neighbors. In some situations, it might be impossible with a fixed number of variables to implement all communication patterns prescribed by the Dynamic MSC Grammar.

Definition 72 *Let $L \subseteq \mathbb{M}$ be an MSC language. We call L* well-formed *if there is a DCA $\mathcal{A}$ such that $L = \mathcal{F}(\mathcal{A})$. For $B \in \mathbb{N}$, we say that L is B-*realizable *if there is a DCA $\mathcal{A} = (X, Msg, Q, \Delta, \iota, F)$ such that $L = \mathcal{F}(\mathcal{A})$ and $|X| \leq B$.*

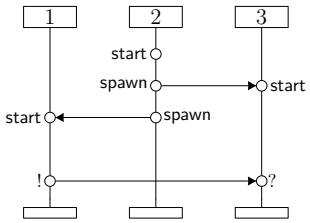


Figure 7.11: not well-formed

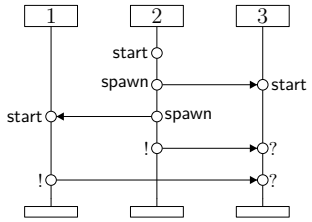


Figure 7.12: 2-well-formed

Consider the examples of Figure 7.12. The language $L = \{M\}$, where M is the MSC of Figure 6.2 is 3-well-formed. It is, however, not 2-well-formed. If we consider the language $L_2 = \{M_2\}$, where M_2 is the MSC from Figure 7.11, we have that L_2 is not well-formed. The reason is that when process 3 is created, process 1 is not know from process 2, and hence process 3 has no way to know process 1's identity. In the semantics, we forbid receptions from unknown processes. This semantics can be relaxed to allow message receptions from any process, but a similar problem arises if process 3 has to send a message to process 1. A slight change in the MSC, as shown in Figure 7.12, where a message from 2 to 3 is inserted after creation of 1 solves the problem. The reason is that this new message can be seen as carrying the identity of the newly created process.

Theorem 44 *For a DMG G, one can decide in doubly exponential time if $L(G)$ is well-formed.*

As for emptiness, the proof for well-formedness is obtained by decorating the parse trees of MSC grammars by structures that represents process identifiers migrations, and in addition the knowledge that a process may have of its neighbors.

5.3 Local Dynamic MSC Grammars

Well-formedness is close to realizability, so one may wonder why a property that is undecidable for HMSCs becomes decidable for DMGs. A first fact to notice is that DMGs start from a single process. A second remark is that decidability of well-formedness checks that all MSCs recognized by an MSC grammar G are such that threads can have enough knowledge about their neighbors to perform communications prescribed by the grammar. Within this setting, nothing is said about the number of states of a DCA needed to implement a grammar. A well-formed DMG is not necessarily implementable in terms of a finite DCA, for several reasons. First, the behavior of a single process depicted by a DMG need not be regular. Second, the MSCs in the language of a DMG all have a single starting process, with a single start event, and all subsequent events are causal consequences of this initial event. This mimics the shape of orders described by local-choice HMSCs, and the initial process can hence choose a particular derivation of the grammar and communicate its choice to all other processes at spawn time. This can be seen as an anticipation of choices, and again results in an infinite state space for the DCA implementation.

Hence, the real challenge is to propose an implementation with online choices using a DCA with finite number of states. As DMG can simulate HMSCs, all

implementation problems of HMSCs such as non-reconstructible choices may also appear in DMGs. However, we can exhibit a non-trivial class of DMGs that is close to local HMSCs.

Our class will restrict DMGs with right-linear rules. A rule $A \longrightarrow_f expr$ is *right-linear* if $expr$ is of the form $\mathcal{M}$ or $\mathcal{M}.B$. Our implementable class of DMGs inspires from *local-choice HMSCs* as introduced in [79]. Local choice HMSCs are scenario descriptions over a fixed number of processes in which every choice of the specification is taken by a root process for that choice. This root is in charge of executing the minimal event of every scenario starting at this choice, and the subsequent messages can then be tagged to inform other processes about the choice. Note that locality allows for a deadlock-free implementation if we deal with a fixed number of processes [57]. However, in our setting, deadlock-freeness is not guaranteed in general.

To adapt the notion of local-choice to DMGs, we essentially replace the notion of "process" in HMSCs by that of "process identifier". It means that the root process that choses the next rule to be applied must come with a process identifier π that is *active* in the current rule. For a right-linear rule $r = A \longrightarrow_f (M, \mu).expr$, we let $Active(r) = f(Free(M)) \cup \mathrm{dom}(\mu)$.

Definition 73 *A DMG* $(\Pi, \mathcal{N}, S, \longrightarrow)$ *is* local *if, for every rule* $r = A \longrightarrow_f expr$, *r is right-linear and* $M(expr)$ *has a unique minimal element. Moreover, if* $expr = \mathcal{M}.B$, *then there is* $\pi \in Active(r)$ *such that, for all B-rules* $B \longrightarrow_g \beta$, $M(\beta)$ *has a unique minimal element e satisfying* $g(\varphi(e)) = \pi$.

The projection of a local MSC grammar on a single process is a regular language, and when a non-terminal is rewritten, its is replaced by pMSCs that contain a single deciding thread. This is a good start to demonstrate implementability of local DMGs. We conjecture that local DMGs contains a reconstructible subclass, and can be implemented with deadlocks, using techniques close to that of [56]. We also think that the implementation mechanism using tags proposed in section 4 of this chapter can be adapted by circulating the same vectors, and maintaining some information on which process identifiers are attached to each threads. If this information is available, a thread can decide whether he is concerned by a message, and also whether it should consume it. This might be implementable if identifiers migrations are performed during messages or spawns.

6 Conclusion

In this chapter, we have considered synthesis of independent communicating machines from HMSCs. Synthesis of CFSMs by a simple projection mechanism is correct for reconstructible HMSCs. A controlled synthesis allows to build a system of communicating entities that simulate the original specification. We think that the class of local HMSCs is a good compromise between the abstraction that is required in a specification formalism, and the preciseness that is needed for a model to be implementable. Indeed, imposing local choices avoids considering in the synthesis some heavy synchronization mechanisms among instances to ensure that distant processes behave according to the same chosen scenario. The class of local HMSCs

seems expressive enough to model many interesting protocols, and furthermore, locality of HMSCs is decidable. The synthesis algorithms have been implemented in our tool SOFAT [66], to generate a formal description of the CFSM from an HMSC, Promela code, or even java code for all the instances and controllers needed in the system.

Possible extensions of the controlled synthesis techniques could consider introduction of time or data issues in the model. Inserting manipulation of local data in the internal actions of processes can be done easily by mixing the language of MSCs with a data manipulation language. The code attached to actions can then be copied as it is in the generated code, which does not really impact the synthesis process. However, if data are shared and used to guard choices in HMSCs, the projection technique does not necessarily work, and additional synchronization and consistency mechanisms are needed to ensure that the synthesized processes work with the same data values.

Time issues are also complex to handle. If we consider for instance as an input model a time-constrained MSC [8], synthesizing a correct model means synthesizing machines that meet all the time constraints expressed in the specification. This imposes in particular that controllers should also play the role of timed schedulers. In such a context, using equality or simulation as notions of correctness for synthesis seems too constraining. However, we could consider different correctness criteria, such as untimed language equivalence.

One may have noticed that the controllers used in our implementation solution only delay messages. Hence, a similar result could have been achieved by allowing tags in the canonical implementations of HMSCs, and changing the semantics of receptions. However, we think that controllers, with some additional power, might be used for different purposes, such as monitoring, or to enforce some simple safety properties.

We have introduced an implementation model for dynamic MSC grammars, namely the model of dynamic communicating automata. Several issues remain open. Even if a class of local MSC grammars that seem implementable has been defined, it remains to show that this class is indeed implementable. We think that the controlled synthesis technique used for local HMSCs could be used to implement DMGs. Another way to find implementation techniques is to find regular subclasses of MSC grammars. Regular sets of MSCs over a fixed number of processes have already been studied [74] (a set of MSCs is called regular if the associated linearization language is a regular language). We would like to define a robust notion of regularity that takes thread creation into account, and is implementable by dynamic communicating automata.

A more general issue concerns more global properties of scenarios. It seems that all implementable classes of scenarios have languages with similar shapes. The runs of local HMSCs or local DMGs can be viewed as sequences of finite lower semilattices. One may wonder if such property can not be used to characterize a large subclass of scenario languages that can be implemented without deadlocks.

Chapter 8

Application 1: Verification

Mais il faut tout de même voir qu'il y a des ordres apparents qui recouvrent, qui sont les pires désordres.
One should see that some apparent orders cover or even are the worst disorders.
[Charles Peguy, Notre jeunesse]

1 Introduction

Partial order automata are formal models used to represent distributed systems such as protocols, transaction systems, etc. Hence, it seems natural to adapt formal verification techniques to this model, i.e. given a formal model M, and a property to check φ, check whether M satisfies φ?

Before entering in technicalities, we should discuss the objectives of verification techniques for scenario models. We have seen in chapters 3 and 4 that HMSCs alone can only express finitely generated families of partial orders. Splitting messages allows more complex behaviors (see chapter 3), but embeds the expressive power of CFSMs in HMSCs, hence making all verification problems undecidable. To allow model-checking, one has to restrict to subclasses of CMSCs, and hence assume that an HMSC specifications can be incomplete subsets of behaviors of a larger specification. Using commutations among events located on one instance (see chapter 4) forces to consider **all** legal interleavings of events: to define an MSC language that contains a braid, one also have to include a finitely generated scenario in which the order of events follow exactly a sequential order imposed by paths of the support HMSC. Again, to be able to mode-check a causal MSC specification, one needs to ensure that this specification belongs to a subclass. Furthermore, composition operators, as we have shown in chapter 5, seldomly preserve syntactic subclasses of HMSCs. As highlighted in the conclusions of chapter 5, one should consider partial order automata more as an abstract representation of known behaviors of a distributed system than as a complete specification language. If partial order automata are only a partial and abstract view of the behaviors of a distributed system, what does verification of such models mean ? Why should one care for verification of logical properties at all?

In the next chapter, we will show that some model checking techniques can be useful, not necessarily as a verification techniques per se, but as a tool to answer more practical questions. For instance, diagnosis and model checking of MSO formulae

are tightly connected problems.

In this chapter, we will show that verification for partial order automata is not a straightforward enhancement of model checking techniques for regular models. This chapter is organized as follows: We first recall known undecidability results for verification of regular properties of HMSCs, and restrictions that allow for some easier forms of model checking.

In a second section we show an extension of MSO to describe properties of MSCs, that was originally proposed by Leucker et al [92]. MSO for MSCs is decidable for HMSCs, but also for Dynamic MSCs, and dynamic MSC grammars. This important result will prove very useful in the next chapter, as it allow for diagnosis techniques for scenarios.

We next show that for several attempts to relax the automaton support to generate infinite sets of MSCs leads in general to undecidability. This is the case for template MSCs, and also for a particular MSC logic that was proposed as a model for diagnosis.

2 Standard Results

Model checking for MSCs can be understood in several ways. The first question one may ask is, given a HMSC H a formula φ in modal temporal logic (CTL, LTL, $\mu - calculus$,...), does H satisfies φ ? While such questions have found an answer for models with regular sets of behaviors, for HMSCs and all their extensions, model checking of modal logics is in general undecidable [14], except for the subclass of regular HMSCs. Obviously, any extension of HMSCs suffers the same drawback. Model checking scenario properties either calls for a use of weak subclasses of the considered scenario language (usually with regular behaviors), or for weaker logics, that do not consider global states of the modeled system. Many works change the semantics of HMSCs to stay within the context of regular models. The changes in the semantics are usually to consider communications as synchronous, and to consider strong sequential composition rather that weak sequential composition, that is define:

$$Lin(M1 \circ M2) = \{w = w1.w2 \mid w1 \in Lin(M1) \wedge w2 \in Lin(M2)\}$$

Strong sequential composition assumes some kind of "'synchronization" among instances at the ens of each MSC. We consider that partial order automata are by nature asynchronous models of distributed systems. Synchronous communications and strong sequential composition make sense in a context where instances represent components of a modular system and messages intercomponents calls, and in a centralized environment. However, in our context, we are targeting systems with independent processes, where such semantics can only be seen as an underapproximation of the behaviors of a system.

A way to allow verification beyond regular HMSCs is to address simpler properties such as vacuity of intersections of HMSC languages, of inclusion. Such properties are decidable for globally cooperative HMSCs [55]. The reason why these properties become decidable is that the linearizations of these subclasses can be represented by regular sets of bounded linearizations, hence allowing most of formal verification usually feasible for regular models.

The intersection problem can be formalized as follows: Given $H1$, a specification HMSC, and $H2$, a "safety" HMSC collecting unwanted behaviors, does $\mathcal{L}(H1) \cap \mathcal{L}(H2) = \emptyset$? This problem makes sense if we consider $H1$ and $H2$ as defining collections of behaviors with the same level of abstraction. Again, this problem was proved undecidable [115]. However, for globally cooperative subclasses of HMSCs, CHMSCs, and causal HMSCs, this problem becomes decidable [55] (more precisely, one only asks that $H2$ is globally cooperative). The answer returned for an intersection problem can be seen as a safety property for $H1$, but has to be considered modulo precision and completeness of $H1$ and $H2$. However, if $\mathcal{L}(H1) \cap \mathcal{L}(H2) \neq \emptyset$, this means that the causal ordering between events represented in $H1$ not allowed, and hence that a system implementing this causal ordering is not safe. Note that the class of globally cooperative HMSCs (CHMSCs, causal HMSCs) contains non-regular specifications, and is a reasonably large subclass of the language.

As for intersection, one can wonder if a specification is "included" in another one, that is given $H1, H2$, does $\mathcal{L}(H1) \subseteq \mathcal{L}(H2)$? The problem is undecidable in general, and becomes decidable for globally cooperative subclasses of scenario languages (more precisely as soon as $H2$ is globally cooperative). Let us now discuss the practical use of this verification problem. $\mathcal{L}(H1) \subseteq \mathcal{L}(H2)$ means that $H2$ is a refinement of $H1$, as it allows more behaviors than $H1$. However, if $H1$ and $H2$ are incomplete collections of behaviors of systems up to abstraction, inclusion means that the all causal ordering on events seen in $H1$ still appear in $H2$. However, if $H1$ and $H2$ have distinct alphabets, inclusion is likely to fail, due to events of $H2$ that are not represented in $H1$. Hence $H1$ and $H2$ should have the same level of abstraction, which is not an ideal property for refinement questions. $H1$ can not be included in $H2$ if all runs of $H2$ stat with an single unique event which is not part of the alphabet of $H1$. Even worse, the projection of a globally cooperative HMSC needs not be globally cooperative behavior of the real systems, that is one can not "hide" the events of $H2$ that never appear in $H1$ and then check inclusion. Hence inclusion checking is of limited practical use.

Another possibility is then to consider "'local logics"', that address properties of MSCs in a languages withouts consideration for their global states.

3 MSO for MSCs

Model checking MSO properties for HMSCs was originally proposed by [98], to focus on structural properties of MSCs generated by HMSCs. One theorem given by Madhusudan states that, given an HMSC H and an MSO formula φ where relations depict message exchanges, execution of an internal action, or immediate successor relation on a process, one can decide if φ is satisfied by all MSCs generated by H. As MSO is closed by negation, one can also check whether there exists at least one MSC in $\mathcal{L}(H)$ satisfying φ, etc. A more specialized version of MSO, called MSO for MSCs was also used by [92]. The same paper proves that MSO over MSCs is decidable for dynamic MSCs. We give below the syntax of this logic and some explanations, and state several results.

The syntax of MSO over MSCs is given by

$$\varphi ::= \; lab_a(x) \;\mid\; (u,x) \to (v,y) \;\mid\; x \lessdot y \;\mid\; x \in X \;\mid\; u \in U \\ \mid\; \neg\varphi \;\mid\; \varphi_1 \wedge \varphi_2 \;\mid\; \exists x\ \varphi \;\mid\; \exists X\ \varphi \;\mid\; \exists u\ \varphi \;\mid\; \exists U\ \varphi$$

Intuitively, variables of the form $x, y, z....$ designate events, variables of the form $u, v, ...$ designate a process name, variables of the form $X, Y, Z, ...$ designate sets of events, and $U, V, ...$ designate sets of processes. The meaning of a formula of the from $(u,x) \to (v,y)$ is that x is a send event located on process u, and y is the corresponding receive event located on process v. The meaning of $lab_a(x)$ is that x is an event labeled by action name $a \in \Sigma$, where Σ is the usual set of labels that can appear in MSCs. The meaning of $x \lessdot y$ is that x and y are located on the same process p, and x is an immediate predecessor of y on p.

Satisfaction of an MSO formula is checked by considering an MSC $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \varphi)$ as a labeled graph which nodes are E, which edges are the relations μ and $\bigcup_{p\in\mathcal{P}} \lessdot_p$, and with labeling α. An interpretation $\mathcal{I}$ over M is a function that maps variables $x, y, ...$ to events in E, bound variables $X, Y, ...$ to subsets of events of E, variables $u, v, ...$ to processes in $\mathcal{P}$ and U, V, to subsets of $\mathcal{P}$. We will say that M satisfies φ under interpretation $\mathcal{I}$, and write $M \models_{\mathcal{I}} \varphi$ iff one can show inductively truth of φ, starting from atomic subformulae. Let $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \varphi)$ be a MSC over a set of processes $\mathcal{P}$. Then, we have:

- $M \models_{\mathcal{I}} (u,x) \to (v,y)$ if there exists a process $p = \mathcal{I}(u)$, a process $q = \mathcal{I}(v)$, an event $e = \mathcal{I}(x)$ located on p, an event $f = \mathcal{I}(y)$ located on q and $f = \mu(e)$ in MSC M,
- $M \models_{\mathcal{I}} x \lessdot y$ if there exists a process $p \in \mathcal{P}$ such that $\mathcal{I}(x) <_p \mathcal{I}(y)$,
- $M \models_{\mathcal{I}} lab_a(x)$ iff $\alpha(\mathcal{I}(x)) = a$,
- $M \models_{\mathcal{I}} x \in X$, where x is an event variable, and X an event set variable if $\mathcal{I}(x) \in \mathcal{I}(X)$,
- $M \models_{\mathcal{I}} u \in U$, where u is a process variable and U is a process set variable, if $\mathcal{I}(u) \in \mathcal{I}(U)$,
- $M \models_{\mathcal{I}} \neg\varphi$ iff it is not the case that $M \models_{\mathcal{I}} \neg\varphi$,
- $M \models_{\mathcal{I}} \varphi_1 \wedge \varphi_2$ iff $M \models_{\mathcal{I}} \neg\varphi_1$ and $M \models_{\mathcal{I}} \varphi_2$
- $M \models_{\mathcal{I}} \exists x, \varphi$ iff $\exists e \in E, M \models_{\mathcal{I}_{[x/e]}} \varphi$, where $\mathcal{I}_{[x/e]}$ is the interpretation that maps x to e, and agrees with $\mathcal{I}$ on other variables.
- $M \models_{\mathcal{I}} \exists X, \varphi$ iff $\exists E' \subseteq E, M \models_{\mathcal{I}_{[X/E']}} \varphi$, where $\mathcal{I}_{[X/E']}$ is the interpretation that maps X to E', and agrees with $\mathcal{I}$ on other variables.

An MSC M satisfies a formula φ iff one can find an interpretation $\mathcal{I}$ such that $M \models_{\mathcal{I}} \varphi$. We will also denote by $\mathcal{F}_\varphi$ the set of all MSCs that satisfy formula φ

Definition 74 *Let H be an HMSC, and φ be an MSO formula over MSCs. We will say that H satisfies φ iff for every $M \in \mathcal{L}(H)$, $M \models \varphi$.*

Theorem 45 *[98] Let H be a HMSC, and φ be an MSO formula over MSCs. Then, the problem of checking whether all MSCs in $\mathcal{F}_H$ satisfy φ is decidable.*

The MSO checking problem and its decidability has also been extended to unfoldings of HMSCs [97], and safe CHMSCs [98]. These results hold without any restriction on HMSCs, and also with sensible restrictions on CHMSCs. This results have an intuitive explanation: as soon as one does not consider global states properties, the MSCs in the families of partial orders generated by HMSCs or safe CHMSCs can be considered as graphs. Even better, HMSCs and CHMSCs can be seen as context free graph grammars (in the spirit of [70], with decidable MSO properties.

In 2002, [92] has introduced a class of *dynamic MSCs*, that can describe behaviors over unbounded sets of processes with *fork-join grammars*. Each fork can be seen as a way to generate new threads starting from the existing processes, and each join between two expressions selects which threads from each operand will still be active after join. Again, one can see dynamic MSCs as a context free graph grammar generating scenarios, and obtain the following result. We have described MSC grammars in chapter 6 to generate MSC languages over unbounded sets of processes.

Theorem 46 *[92] Let H be a dynamic MSC, and φ be an MSO formula over MSCs. Then, the problem of checking whether all MSCs in $\mathcal{F}_H$ satisfy φ is decidable.*

The proof by Leucker & al is as follows: a dynamic MSC is some kind of grammar. One can build a tree automaton that recognize parse trees of this grammar. Then, one can refine the sates of the tree automaton attached to leaves to model interpretation, and states attached to inner nodes to model sub-formulae inherited from the subtree recognized at some node. A formula holds if one can attach to the root a state which synthesized formula part implies φ. We refer interested readers to [92] fro more details. Note that proving a formula of the form $\neg\varphi$ needs first to built an automaton for φ, and then complement it, which may impose an exponential blowup in the size of the tree automaton.

We can apply a similar technique to dynamic MSC grammars. One can reuse the automaton $\mathcal{A}_G$ built in chapter 6, theorem 39 to check emptiness of a DMG G. We can similarly attach interpretations and sub-formulae to leaves and nodes of a parse tree recognized by $\mathcal{A}_G$, and obtain a tree automaton that recognizes all MSCs that satisfy φ. As one can check emptiness of a tree automaton (in linear time) [37], we have the following result:

Theorem 47 *Let G be a dynamic MSC grammar, and φ be an MSO formula over MSCs. Then, the problem of checking whether all MSCs in $\mathcal{F}_G$ satisfy φ is decidable.*

One can notice that the simple message problem of chapter 2 can be encoded as a property of the form : $\varphi_{MsgProb} ::= \exists x, y, u, v, (u, x) \longrightarrow (v, y) \wedge lab_{!m}(x) \wedge lab_{?m}(y)$. So MSO is undecidable for non-safe CHMSCs. As one can see, decidability of MSO comes from some kind of regularity in the structure defining how to compose MSCs, but also in a possibility to bound the number of receptions that need to be matched in a sub-expression recognized by a (tree) automaton. If this property is not satisfied,

then there is no bound on the number of sent and not yet received if a representative set of MSC generated by the model, and the (tree) automaton recognizing a representative set of MSCs need an infinite number of states. This restriction is easily guaranteed in HMSCs, Dynamic MSCs, and dynamic MSC grammars, as these structures compose only communication-closed patterns. For CHMSCs, a bound exists only if the considered specification is safe.

4 Partial order logics : Dropping the automaton support

As shown in the former section, decidability of MSO for MSCs in HMSCs safe CHMSCs and MSC grammars comes from regularity (regularity of an automaton support, or regularity of the parse tree language for MSC grammars) and also from existential bounds on communication channels that are guaranteed by the models. These restrictions either allow for the definition of a regular set of representatives, or for decoration of tree automata with interpretations, without consideration on whether communications are correctly mapped. A natural question is whether one can drop the automaton or grammar structure to design MSC languages. So far, we have introduced composition mechanisms that are very sequential, and describe the execution of a protocol from a start to its end, by appending communication patterns one after another. Such formalisms is not always adapted in very early phases of systems design, and one should notice that even if MSCs (or their variants) are very often used to describe and document systems, higher level constructs composing them are less used. An interesting question is whether one can drop this too sequential view of composition, and define scenario descriptions as a collection of known facts on a distributed system composing communication patterns, and if one can still get decidable properties for such models.

Dropping the regular structures to generate MSC languages, we can imagine logical formalisms to collect users knowledge about systems behaviors, under the form of "correlations" between pieces of behaviors. For instance one would like to describe situations such as "whenever a message *Acknowledge* is exchanged between p and q, then a message *Data* was send from q to p some time before", or "A *data* message must occur between two consecutive acknowledgments",.... But beyond the wish to specify with logical statement, we need tools to check if a given specification is consistent. As we have seen for MSC grammars (chapter 6), CHMSCs (chapter 3), the question of whether a specification described using a particular formalism defines a non-empty set of behaviors is a crucial sanity check. This question is undecidable for CHMSCs, but becomes decidable for safe CHMSCs, MSC grammars, and is trivial for HMSCs or causal HMSCs. For shuffles of HMSCs, this question was also undecidable (see chapter 5). Hence, emptiness is a very discriminant test for MSC formalisms, and should be guaranteed by the model. We will also see in the next chapter that satisfiability is a key ingredient for **diagnosis**. We will now review several potential logical formalisms, namely MSO for MSCs, template MSCs, and a new logic of partial order computations, and for each of them consider the requirements needed to ensure consistency checking.

Specifying with MSO for MSCs

A good candidate as logical specification formalism is of course MSO for MSCs. Cast in the MSO context, the consistency check mentioned above is in fact a satisfiability problem. Let φ be a MSO formula over MSCs. We will say that φ is *satisfiable* iff there exists an MSC M such that $M \models \varphi$ (or written differently, does $\mathcal{F}_\varphi = \emptyset$?). However, it was shown that this question is undecidable.

Theorem 48 *[50] The satisfiability problem for MSO over MSCs is undecidable.*

An easy way to prove this theorem is again to define a PCP encoding. One can indeed define formulae over three processes p, q, r, such that one formula φ_p describe sequences of $u_i's$ in a PCP, one formula φ_q describe sequences of $v_i's$, (each letter and index is encoded as a message sending in u_i and located on p, or as a reception reception and located on q for $v_i's$), and an additional process ensures equality of sequences of indexes and letters on p, q.

This means that MSO for MSCs is not a suitable specification formalism. Interestingly, if one assumes that the MSC that shall satisfy φ are concatenations of MSCs chosen from a finite set $\mathcal{M}$, then satisfiability becomes decidable. Decidability also holds if one slightly changes the problem, and gives as input of the problem a bound B and imposes that the considered MSCs satisfying φ are existentially B-bounded.

Theorem 49 *[55] Let B be an integer, φ be an MSO^{MSC} formula. Then one can decide if there exists a $\exists B$-bounded MSC M that satisfies φ*

The proof comes from the fact that one can represent the set of all existentially B-bounded MSCs as a CFSM. The results in [55] also sow that, denoting by MSC^B the set of $\exists B-$bounded MSCs, and given a CFSM $\mathcal{A}$, one can check whether $\mathcal{F}_\mathcal{A} \cap MSC^B \subseteq \mathcal{F}_\varphi$. This means that one can check if $MSC^B \subseteq \mathcal{F}_{\neg\varphi}$, which solves the satisfiability problem for $\exists B$-bounded MSCs.

Template MSCs

The first attempt in the MSC world to drop the automaton structure is called template MSCs [60]. Template MSCs define "MSCs with holes". This formalism can define templates of the form assume/guarantee, i.e. pairs of MSCs of the form (M, M'), meaning that when MSC M is observed, then MSC M' will occur in the future. In addition, template MSCs are extended with a logic that combines templates. Such kind of template specification was also proposed for live sequence charts (LSCs) [38], where in a pair (L, L') of LSCs, the pre-chart L (usually a single message) is seen as triggering LSC L'. However, LSCs have a synchronous semantics that can be described as finite state machines.

Definition 75 *A template MSC is a tuple $(P, E, \Gamma, C, \lambda, m, \leq)$, where $(P, E, C, \lambda, m, \leq)$ is a CMSC, Γ is a set of* boxes*, $\lambda E \cup \Gamma \to \Sigma$ is a mapping that associates a type to events, and a set of allowed types to boxes. The ordering relation $\leq$ is a partial order over events and boxes, and is a total order on each process.*

The intuitive meaning of template MSCs is that boxes in Γ are "holes" (unspecified parts) in a MSC. One can fill a box $\gamma \in \Gamma$ with CMSCs which events are all of type $\lambda(\gamma)$ and build a message mapping that satisfies fifo ordering on messages to obtain a complete MSC. A single template MSC T defines a potentially infinite set of MSCs denoted by $\mathcal{F}_T$.

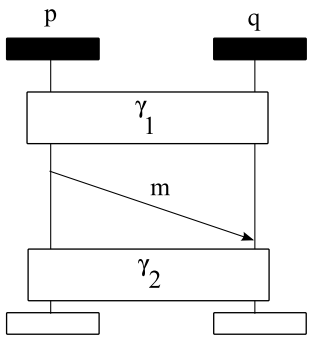


Figure 8.1: A template MSC

Figure 8.1 shows an example of template MSC with two boxes γ_1, γ_2. By simply setting $\lambda(\gamma_1) = \lambda(\gamma_2) = \bigcup_{\sigma \in \Sigma} p!q(\sigma) \cup q!p(\sigma) \cup p?q(\sigma) \cup q?p(\sigma)$, the template MSC of Figure 8.1 represents all MSCs with messages of type Σ that contain at least one occurrence of message m from p to q. One can immediately notice that template MSC languages are not necessarily existentially bounded, and hence can be difficult to analyze or implement.

Template MSCs have been extended with a simple logic called *assume-guarantee template MSC*. Expressions of this logic are of the form $T_a \rightarrow T_b$ and $T_a \rightarrow \neg T_b$. The semantics of these expressions are again MSC languages:

$$\begin{aligned} \mathcal{F}_{T_a \rightarrow T_b} &= \{M \mid \text{for every factorization M=N.N', } N \notin L(T_a) \text{ or } N' \in L(T_b)\} \\ \mathcal{F}_{T_a \rightarrow T_b} &= \{M \mid \text{for every factorization M=N.N', } N \notin L(T_a) \text{ or } N' \in L(T_b)\} \end{aligned}$$

This logic can be extended with conjunction, to obtain expressions of the form $\varphi = \bigwedge_i T_i \longrightarrow \bigvee_j \pm T_j$, where $\pm T_j$ stands for either T_j or $\neg T_j$. The meaning of this formula is that a MSC belongs to $\mathcal{F}_\varphi$ if for every T_i there exists one T_j (respectively $\neg T_j$) such that $M \in \mathcal{F}_{T_i \longrightarrow T_j}$ (respectively $M \in \mathcal{F}_{T_i \longrightarrow \neg T_j}$). Such expressions are called *template MSCs formula*. However, satisfiability of template MSCs formula is undecidable:

Theorem 50 *[60] Satisfiability of a template MSC fomula φ is undecidable. Given a bound B, one can decide if there exists a $\exists B$-bounded MSC in $\mathcal{F}_\varphi$.*

Again, the undecidability comes from an encoding of a PCP. We refer interested readers to [60] and [58] for more results on template MSCs. Undecidability of template MSC formulas show that it is hard to design specifications with this logic (at least without restriction), as one can not decide if a formula has an MSC model. Template MSCs are strictly included in FO [58], hence there are many MSO properties that do not have a counterpart in Template MSCs.

We note that the satisfiability problem of several local temporal logics in the literature are undecidable. These include m-LTL [104], a local temporal logic on Lamport diagrams. Similarly, satisfiability for TLC^-, a fragment of MSO for MSCs proposed in [121] is known to be undecidable [50].

LPOC: a Logic of Partially Ordered Computations

One may wonder if the problem with template MSCs does not come from the fact that we are considering messages, and furthermore FIFO communications, which are well known to introduce undecidability (CFSMs can encode Turing machines using their queues as a tape). We hence have considered a slightly different logic called LPOC (Logic of Partial order Computations), that aims at defining causal dependencies among local states in distributed computations [138], without mentioning explicitly communications. The main idea driving this work was to propose a language to collect knowledge of a system without the need to describe an operational model specifying runs of a system from its initial configuration. The goal of this formalism was also to define behaviors up to some abstraction: messages, changes of phases of protocols, etc are not necessarily observable, nor precisely specified. This can be particularly true in a context where a part of a system may have been designed by external developers, and in such a way that design details are not precisely documented. Hence, one can not always reason on a system in terms of messages, MSCs etc. MSO for MSCs partially answers this problem, by describing relations among events that can occur at any place during a computation, but does not emphasize this notion of causal pattern. Furthermore, we already know that satisfiability of MSO is undecidable. Template MSCs is another alternative, but emphasizes messages, and furthermore, satisfiability is also undecidable. The LPOC formalism presented hereafter describes a system in terms of causal dependencies patterns, and in terms of relations between these patterns, expressed using a temporal logic.

In this section we will drop the assumption that behaviors of distributed systems are necessarily described with MSCs. Considering this, we have proposed a logic-based models that take causal relations among identified local states of a system as building bocks. More precisely, we represent behavioral patterns of distributed systems with restricted partial orders which define cause-effect relations among local states. These partial orders are called *partially ordered computations*. We then propose a temporal logic over partially ordered computations and call it simply the *logic of partially ordered computations* (LPOC). The main feature of LPOC is to reason about evolution of patterns of causal orders. We use temporal operators similar to Computation Tree Logic [36], and use them to combine patterns, that is finite partial orders (depicting for instance short sequences of message exchanges).

We will consider that LPOC formulae are defined for a known and nonempty set $\mathcal{P}$ of process names, and $\mathcal{AP}$ a finite nonempty set of atomic propositions. We let p, q range over $\mathcal{P}$.

Definition 76 *A* partially ordered computation *(or computation for short) over* $(\mathcal{P}, \mathcal{AP})$ *is a tuple* $(S, \eta, \leq, V)$ *where:*

- S *is a finite set of* (local) states.

- $\eta : S \to \mathcal{P}$ *identifies the* location *of each state. For each* $p \in \mathcal{P}$*, we define* $S_p = \{s \in S \mid \eta(s) = p\}$.
- $\leq \subseteq S \times S$ *is a partial order, called the* causality *relation. Furthermore, for each* p, $\leq$ *restricted to* $S_p \times S_p$ *is a total order.*
- $V : S \to 2^{\mathcal{AP}}$ *is a labeling function which assigns a set of atomic propositions to each state. We call* $V(s)$ *the* valuation *of* s.

Intuitively, a computation (also called Lamport diagram in the literature [104]) represents the causal ordering among local states in a distributed execution, in which states of each process are sequentially ordered. The valuation of local state s collects the atomic propositions that hold at s. Such propositions can denote the type of the last event executed on some process, some known facts about variables assignments in the local state, ... Figure 8.2-a) shows a computation. States are designated by black dots with associated name $s_1, \ldots, s_6$. Processes P, Q, R are represented by vertical lines, and states located on a process line are ordered from top to bottom. Finally, valuations of states take value in $\{a, b, c\}$, and are represented between two brackets near the associated state. Note that we consider only *finite* computations. We will say that two computations $(S, \eta, \leq, V)$ and $(S', \eta', \leq', V')$ are *isomorphic* iff there is a bijection $f : S \to S'$ such that $\eta(s) = \eta'(f(s))$ for any $s \in S$, $s_1 \leq s_2$ iff $f(s_1) \leq' f(s_2)$ for any $s_1, s_2 \in S$, and $V(s) = V'(f(s))$ for any $s \in S$. We identify isomorphic computations and write $W \equiv W'$ if $W$and W' are isomorphic.

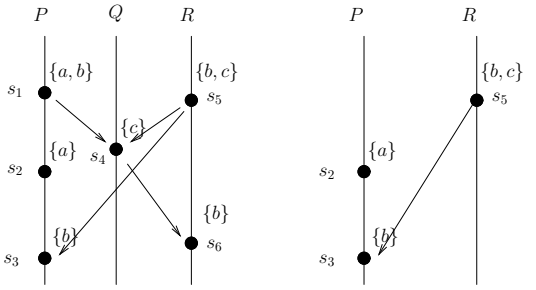


Figure 8.2: a) A computation W b) the 1-view of s_3 in W

Let $(S, \eta, \leq, V)$ be a computation, and $s, s' \in S$. As usual, we write $s < s'$ when $s \leq s'$ and $s \neq s'$. For each $p \in \mathcal{P}$, we define $\ll_p \subseteq S \times S$ as: $s \ll_p s'$ iff $s, s' \in S_p$, $s < s'$, and there does not exist $s'' \in S_p$ with $s < s'' < s'$. That is, $\ll_p$ is the "immediate" sequential ordering of states belonging to p. We let $\lessdot \subseteq S \times S$ be the least relation such that $\leq$ is the reflexive and transitive closure of $\lessdot$. For each $p, q \in \mathcal{P}$ with $p \neq q$, let us define $\ll_{pq} \subseteq S \times S$ as follows: $s \ll_{pq} s'$ iff $s \in S_p$, $s' \in S_q$, and $s \lessdot s'$. We also define $\ll = (\cup_{p \in \mathcal{P}} \ll_p) \bigcup (\cup_{p,q \in \mathcal{P}, p \neq q} \ll_{pq})$. We note that $<$ is in fact the transitive closure of $\ll$. If $s \ll s'$, we say s' is a *(causal) successor* of s, and call s a *(causal) predecessor* of s'. We emphasize that $\ll$ is not equal to $\lessdot$. Indeed, $\lessdot$ does not necessarily capture the relations defined by the local ordering on processes. Consider for instance states s_5 and s_6 in Figure 8.2-a: we have $s_5 \ll s_6$, but not $s_5 \lessdot s_6$. A state s is *minimal* if it has no predecessor, and *maximal* if it has no successor. A *causal chain* is a sequence $s_1 s_2 \ldots s_n$ of states where $s_1 \ll s_2 \ll \ldots \ll s_n$.

In the sequel, we define a temporal logic of partial-order computations (called "LPOC" for short) to reason about distributed behaviors. It has two basic features. First, at a state s of a computation, atomic formulae assert that a "pattern" occurs in a bounded past or bounded future of s. Secondly, we consider a branching time framework with CTL-like operators and reason along sequences of causally ordered states.

Definition 77 *Let $(S, \eta, \leq, V)$ be a computation, and m be a natural number. The* m-view *of $s \in S$, denoted $\downarrow_m(s)$, is the collection of states s' in S such that there exists a causal chain of length at most m starting from s' and ending at s. More precisely, $\downarrow_m(s) = \{s' \mid \exists s_0, \ldots s_n \in S, n \leq m$ and $s' = s_0 \ll s_1 \ll \ldots \ll s_n = s\}$. Similarly, the* m-frontier *of s, denoted by $\uparrow_m(s)$, is the collection of states s' in S such that there exists a causal chain of length at most m starting from s and ending at s'.*

Intuitively, the m-view of a state described the causal past of a state, and m-frontier the causal future. Figure 8.2-b) shows an example of m-view. Note that the 0-view and 0-frontier of a state s are both the singleton set $\{s\}$. Each state s has at most $|\mathcal{P}|$ successors, one belonging to each S_p. Thus, inductively, the m-view and m-frontier of s contains at most $\mathcal{N}_m = \sum_{i=0}^{m} |\mathcal{P}|^i = \frac{1-|\mathcal{P}|^{m+1}}{1-|\mathcal{P}|}$ states. In order to reason about the "pattern" of a computation, we also need a notion of *projection*.

Definition 78 *Let $W = (S, \eta, \leq, V)$ be a computation over $(\mathcal{P}, \mathcal{AP})$, and let $A \subseteq \mathcal{AP}$. The* projection *of W onto A is the computation $W' = (S', \eta', \leq', V')$ where $S' = \{s \in S \mid V(s) \cap A \neq \emptyset\}$, and η', $\leq'$, are the respective restrictions of $\eta, \leq$ to S' and $V'(s) = V(s) \cap A$ for every $s \in S'$.*

The atomic formulae of our logic will assert that the projection of the computation formed from the m-view or the m-frontier of a state is isomorphic to a given computation. We are now ready to define the logic LPOC.

Definition 79 *The set of LPOC formulae over a set of processes $\mathcal{P}$ and a set of atomic propositions $\mathcal{AP}$, is denoted by $LPOC(\mathcal{P}, \mathcal{AP})$, and is inductively defined as follows:*

- *For each $p \in \mathcal{P}$, the symbol loc_p is a formula in $LPOC(\mathcal{P}, \mathcal{AP})$.*
- *Let m be a natural number, A be a subset of $\mathcal{AP}$, and $T = (S, \eta, \leq, V)$ be a computation such that $V(s) \subseteq A$ for every $s \in S$. Then $\downarrow_{m,A}(T)$, $\uparrow_{m,A}(T)$ are formulae in $LPOC(\mathcal{P}, \mathcal{AP})$.*
- *If φ, φ' are formulae in $LPOC(\mathcal{P}, \mathcal{AP})$, then $\mathsf{EX}\varphi$, $\mathsf{EU}(\varphi, \varphi')$ are formulae in $LPOC(\mathcal{P}, \mathcal{AP})$.*
- *If $\varphi, \varphi' \in LPOC(\mathcal{P}, \mathcal{AP})$, then $\neg\varphi$ and $\varphi \vee \varphi'$ are formulae in $LPOC(\mathcal{P}, \mathcal{AP})$.*

From now on, we shall refer to formulae in $LPOC(\mathcal{P}, \mathcal{AP})$ simply as *formulae*. Their semantics is interpreted at local states of a computation. For a computation W and a given state s of W, we write $W, s \models \varphi$ when W satisfies φ, which is defined inductively as follows:

- $W, s \models loc_p$ iff the location of s is p (i.e. $\eta(s) = p$),
- $W, s \models \downarrow_{m,A}(T)$ iff the projection of $\downarrow_m(s)$ onto A is isomorphic to T.
- $W, s \models \uparrow_{m,A}(T)$ iff the projection of $\uparrow_m(s)$ onto A is isomorphic to T.
- $W, s \models \mathsf{EX}\varphi$ iff there exists a state s' in W such that s' is a causal successor of s and $W, s' \models \varphi$.
- $W, s \models \mathsf{EU}(\varphi, \varphi')$ iff there exists a causal chain $s_1 s_2 \ldots s_n$ in W with $s = s_1$. Further, there exists an index i in $\{1, 2, \ldots, n\}$ with $W, s_i \models \varphi'$, and $W, s_j \models \varphi$ for every j in $\{1, 2, \ldots, i-1\}$.

The semantics for boolean combinations and negations of formulae is as usual. We assume the standard boolean operators. We define some derived temporal operators as follows: $\mathsf{EF}\varphi \equiv \mathsf{EU}(true, \varphi)$, $\mathsf{EG}\varphi \equiv \mathsf{EU}(\varphi, \varphi \wedge \neg\mathsf{EX}true)$, $\mathsf{AX}\varphi \equiv \neg\mathsf{EX}(\neg\varphi)$, $\mathsf{AF}\varphi \equiv \neg\mathsf{EG}(\neg\varphi)$, and $\mathsf{AG}\varphi \equiv \neg\mathsf{EF}\neg\varphi$. We can also assert the truth of an atomic proposition a at a state of a computation with $\varphi_a = \bigvee_{p \in \mathcal{P}} \big(loc_p \wedge \downarrow_{0,\{a\}}(T_{p,a})\big)$, where each $T_{p,a}$ is the computation containing a singleton state of location p and valuation $\{a\}$. We have chosen the existential until operator because it is essential in asserting properties such as "whenever some pattern T occurs, some other pattern T' will follow". More precisely, this demands that along every causal chain, whenever a pattern T occurs, pattern T' should occur later *and* no more pattern T can occur again before the point at which the pattern T' has occurred. This kind of properties are commonly needed in practical applications.

For a computation W and a LPOC formula φ, we say that W *satisfies* φ, written $W \models \varphi$, iff there exists some minimal state s_{min} of W such that $W, s_{min} \models \varphi$. We say that φ is *satisfiable* iff there exists a computation W such that $W \models \varphi$.

Let us define a simple example with LPOC. We define a formula meaning that whenever a connection phase described by a pattern T_{conn} occurs between two processes *Client* and *Server*, then a data transfer described by a pattern T_{data} necessarily occurs later. This formula can be expressed by $AG\varphi$, where :

$$\begin{aligned} \varphi = \quad & (loc_{Client} \wedge \uparrow_{2,A}(T_{conn})) \\ & \Rightarrow (EX(loc_{Client} \wedge EX(EU(loc_{Client}, loc_{Client} \wedge \uparrow_{2,A'}(T_{data}))))) \end{aligned} \ ,$$

$A = \{disc, noclient, client, connected\}$, and $A' = \{DataSent, DataRecv, DataAck\}$, and the patterns T_{conn} and T_{data} are described in Figure 8.3.

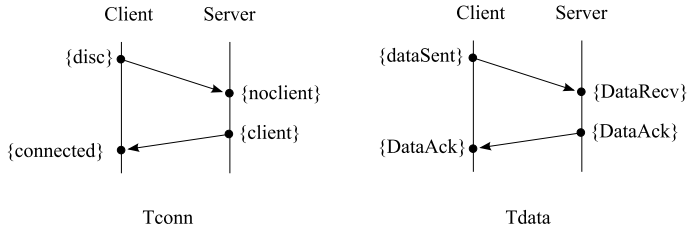


Figure 8.3: Two patterns

We can also define the notion of a computations satisfying a collection of formulae, one for each process. Formally, for a computation $W = (S, \eta, \leq, V)$ and a

$\mathcal{P}$-indexed family of formulae $\{\varphi_p\}_{p\in\mathcal{P}}$, we say W satisfies $\{\varphi_p\}_{p\in\mathcal{P}}$, written $W \models \{\varphi_p\}_{p\in\mathcal{P}}$ by abuse of notation, iff the following condition holds: for each $p \in \mathcal{P}$, $S_p \neq \emptyset$ and $W, s_p \models \varphi_p$ where s_p is the minimum state in S_p (i.e. $s_p \leq s$ for every $s \in S_p$). Note that $W \models \{\varphi_p\}_{p\in\mathcal{P}}$ iff $W \models \bigwedge_{p\in\mathcal{P}} \mathsf{EU}(\neg loc_p, loc_p \wedge \varphi_p)$.

The LPOC logic presented above satisfies most of the requirements we have for representation of partial knowledge of a system. However, one can easily show that this logic is too expressive.

Theorem 51 *[138] Satisfiability of an LPOC formula is undecidable.*

The proof uses the same PCP encoding used to prove undecidability of satisfiability for MSO over MSCs. Letting $\varphi_p, \varphi_q, \varphi_r$ be LPOC formulae encoding properties of a distributed system with processes $\mathcal{P} = \{p, q, r\}$. φ_p can be used to encode the behavior of one process that choses indexes and sequences of messages in a PCP, φ_q the behavior of another one, and φ_r to ensure that the two processes choose the same indexes and the same sequences of messages. Then, deciding if there exists a run of the system satisfying φ_p, φ_q,and φ_r is undecidable. We refer interested readers to the appendix for proofs of this theorem. This is not really a surprise, as satisfiability TLC^-, a logic that describes properties of a single causal chain in a MSC (that is LPOC formulae that use only 0-views) is already undecidable.

As one can see, LPOC suffers the same drawback as MSO for MSCs, and template MSCs. There are two usual ways to overcome this undecidability problem. The first one is to consider a decidable fragment of the logic. Note however, that encoding a PCP becomes possible as soon as there is a way to describe sequences of properties located on a given process, and to define a mapping of states on different processes that respects the ordering. This is why in most cases very small fragment of partial order logics become undecidable when no restriction is imposed on the kind of model considered. Then, the question that naturally arises is whether we can identify a subclass of computations for which LPOC diagnosis is tractable. Again, one can rely on bounds to ensure decidability. However, bounds for MSCs-based languages are defined in terms of channel contents. Such definition of bounded runs does not directly apply to distributed behaviors in which the notion of message is not necessarily specified. However, there is another way to restrict the set of computations that we consider: for a given K, we limit explanations to so-called K-influencing distributed behaviors in which each process causally influences every other process in a bounded manner. Within this setting satisfiability of an LPOC fomlula is decidable (for the given K).

Definition 80 *Let $W = (S, \eta, \leq, V)$ be a computation, and $p, q \in \mathcal{P}$ with $p \neq q$. The* causal degree *of p towards q in W is the maximum integer $n \in \mathbb{N}$ for which there exist s_1, s_2, ..., s_n in S_p, and s'_1, s'_2, ..., s'_n in S_q such that:*

(i) $s_1 < s_2 < \ldots < s_n$ *and* $s'_1 < s_2 < \ldots < s'_n$.

(ii) for $i = 1, 2, \ldots, n$, $s_i \ll s'_i$, *that is, s_i is a predecessor of s'_i.*

(iii) $s'_1 \not< s_n$.

*For $K \in \mathbb{N}$, W is K-*influencing *iff for any pair of processes p, q in $\mathcal{P}$ with $p \neq q$, the causal degree of p towards q is at most K.*

Intuitively, the causal degree of p towards q is the maximal number of events that precede some event on q that p can execute without having to wait for q. The general shape of $K-$influencing computations is illustrated in Figure 8.4. We now state the main result of this section.

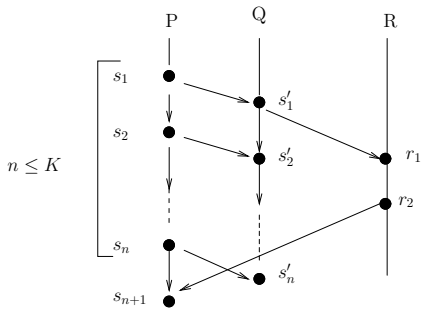


Figure 8.4: $K-$influencing computations

Theorem 52 *Given a $\mathcal{P}$-indexed family $\{\Phi_p\}_{p\in\mathcal{P}}$ of LPOC formulae, and an integer $K \in \mathbb{N}$, one can effectively determine whether there exists a $K-$influencing computation W which satisfies $\{\Phi_p\}_{p\in\mathcal{P}}$.*

The main idea of the proof is that one can build an automaton that recognizes linearizations of all K-influencing computations (over a set of processes $\mathcal{P}$ and atomic propositions $\mathcal{AP}$). Then, one can build a two-way alternating automaton Alt_φ recognizing $K-$influencing computations that satisfy φ. This automaton can then be translated to a finite state machine Aut_φ.

It may seem surprising that one needs to restrict computations to regular models to gain satisfiability, while satisfiability of MSO for MSCs is decidable as soon as we restrict to **existentially bounded MSCs**. The reason for this is that a single linearization of an MSC uniquely determines this MSC. Then restricting to existentially B-bounded MSCs allows to represent each MSC by a B-bounded representative linearization, and the set of existentially B-bounded MSCs can be described as a regular set of representatives. In partially ordered computations, a linearization may correspond to several computations, and hence we may need to consider an arbitrary set of linearizations to define uniquely concurrency and causality in computations, even if the computations have properties similar to existential K-boundedness.

Even with the restriction to K-influencing computations, satisfiability and in general verification of LPOC is very expensive (several exponential in the size of the formula and exponential in the size of the observed behavior). This high complexity could mean that most of problems (diagnosis, verification) addressed with LPOC are intractable. Note however that this complexity is in the worst cases. We strongly believe that if the m-views (resp. frontiers) in fromulae is bounded by some small integer, the complexity is more tractable. We could also consider a fragment of the logic. Note however that most of the modalities chosen for LPOC seem important.

The simple example of Figure 8.3 shows that the Until operator is essential to express properties of the form "when T1 occurs, T2 will occur later". One may also try to restrict the use of negation, that is LPOC formulae would only be conjunctions of positive assertions on the occurrence of patterns.

5 Comparison of different models

As LPOC uses the existential until operator, for a given K, LPOC restricted to K-influencing computations is not definable in the first order logic over the Mazurkiewicz traces encoding K-influencing computations. An interesting work would be to look for a fragment that is expressively complete for the first order logic over the traces encoding K-influencing computations. We furthermore think that one can define an MSO logic of partially ordered computations (using suitable relations of successor on the same process and on distinct process) and define LPOC as a fragment of this logic.

In [121], D.Peled shows that model checking TLC$^-$ formulae on High-level Message Sequence charts (HMSCs) is decidable. TLC$^-$ is a subset of TLC that only contains next and until temporal operators, and describes the shape of causal chains in all the partial orders generated by a HMSC. It is also a subset of MSO [98]. TLC^- is clearly less expressive than LPOC, but satisfiability of TLC^- formulae is already undecidable [50].

The Propositional Dynamic Logic (PDL) for message passing systems proposed by [27, 28], extends dynamic LTL for traces [75]. PDL is a proper fragment of MSO, and model checking PDL properties over HMSCs is PSPACE complete. [104] proposes a local logic LD0 and several extensions over computations, with future and past modalities, and show that in the general case, satisfiability of PDL formulae is undecidable. $LD0$ can be seen as an extension of PDL that allows to navigate causal relation backwards. PDL and LD0 are subsets of MSO. They become decidable when considering models of bounded size, or when computations can be organized as successive layers of finite message exchanges. LD0 only describes chains of causally related events occurring in the future or in the past of a local state, while the template matching in LPOC allows to describe a complete partial order in a bounded future or past of a local state. For instance, the fact that at some state, the causal future (the m-view) of a local state forms a lattice can not be expressed in LD0 (nor in PDL). LPOC is then more discriminating than LD0, and if we restrict our models to Message Sequence Charts (a partial order where locality of events and messages are explicitly represented), it is also more expressive and discriminating than TLC^- and PDL (as these logics consider properties of a single causal path). On the other hand, $LD0$ allows definition of formulae that describe properties of local states occurring arbitrarily far in the past of a local state. So, LPOC and LD0 are incomparable.

Note also that for TLC^-, PDL, or $LD0$, partial orders are seen as models of formulae, but not as elements of the logic itself. The closest approach mixing logic and partial orders is clearly Template Message Sequence Charts [60]. Note however that the models of template MSC formulas are MSCs. As shown previously, the logic is very expressive, but satisfiability is undecidable when no bound is assumed on the set of MSCs considered. Models for template MSC formulae are MSCs, while

models for LPOC formulae are arbitrary computations. Even if we only consider LPOC formulae over MSCs, LPOC and template MSCs remain incomparable. On one hand, holes in template MSCs are not necessarily descriptions of what happens in the future or in the past of an event. By filling hole, one may add concurrent events, i.e. it is possible to say with template MSCs that whenever an action a occurs on process p, a concurrent action b occurs on process q. Clearly, this kind of formula can not be expressed with LPOC. On the other hand, some LPOC formulae that use the until operator do not find their equivalent in template MSC.

Note also that the works in [121], [27, 28] and [60] rely on the existentially bounded nature of models to ensure decidability of model checking (which existence is guaranteed by HMSCs). In general, existential bound suffice This is not sufficient in our case to obtain decidability for LPOC, as the PCP encoding used to prove undecidability is indeed existentially bounded. The K-influencing restriction is then closer to the universal bound on MSCs (the contents of communication channels in all linearizations of MSCs is bounded by some integer b) needed to model check HMSCs with global logics [14]. It might be interesting to see whether the layered computation restriction of [104], that imposes computations can be represented as finite layers of local states located at equal distance from an set of initial states, is sufficient to make diagnosis with LPOC formulae decidable.

6 Conclusion

In this chapter, we have recalled that most global logics were undecidable for HMSCs and their variants. Indeed, undecidability arises as soon as one considers regular model checking, except if the model checking problem is applied for a subset of the scenario language with regular set of linearizations.

Considering local makes most of model-checking problem decidable: for instance verification of properties of MSO^{MSC} is decidable for HMSCs, dynamic MSCs, safe CHMSCs, and MSC grammars. Though local logics are usually considered as less powerful than global logics, an furthermore are often expensive, this is certainly the price to pay to equip scenarios with verification tools. Furthermore, it seems logical to model check local properties of specification that do not rely on global states to model a distributed system.

To go further, one could hope to use logical formalisms as specification language, or as a way to capture users knowledge and reason formally on this knowledge base. However, satisfiability of local logics is rapidly undecidable. As one can not decide in general if a logical specification describes at least a run, logic-based specification is not a viable technique. Solutions exist when additional assumptions (such as existential B-boundedness for a fixed B) are imposed to the runs of the considered system. Considering other restrictions, including for models that are not MSCs is a challenging issue.

Even though modeling with MSO or more generally with local logics is not feasible in full generality, decidability of verification for models that enjoy regular composition mechanisms is a good news, and will be used in the next chapter to show decidability of diagnosis.

Chapter 9

Application 2 : Diagnosis

L'ordre est le plaisir de la raison: mais le désordre est le délice de l'imagination.
Order is reason's pleasure, but disorder is imagination's delight
[Paul Claudel, Le Soulier de satin]

1 Introduction

This chapter is devoted to the application of scenario models to diagnosis. Diagnosis consists in building explanations from a model and an observation of a running system. Such explanations can be provided as a set of runs of the model, or as a generator for such this set. The problem has been addressed with several models, but the HMSC-based approach detailed hereafter showed to be efficient and scalable. In particular, it provided a solution to the non-observable cycles problem that raises termination problems for diagnosis techniques based on unfoldings.

This chapter is organized as follows: we first recall the main objectives of diagnosis, and then focus on a HMSC-based solution, that was originally proposed in [77]. We then show that in general diagnosis can be brought back to model-checking of an MSO formula built out of an observation, and that hence Dynamic Message Sequence Charts or Dynamic MSC grammars can be used for diagnosis. The work presented in this chapter summarizes a joint works with Blaise Genest, Thomas Gazagnaire, Hervé Marchand, and Benedikt Bollig.

2 Diagnosis

Complexity of distributed systems calls for automated techniques to help designers and supervisors in their tasks. Before correcting a system's software, or taking a decision (for instance a reconfiguration of a network), stakeholders need to obtain information on what occurred in a network before entering a faulty configuration, on the current state of the system, etc. The role of diagnosis is to provide a feedback to supervisors of a system (this can be online, to obtain some information on the current status of a running system, or offline, to know why a fault occurred and then correct the incriminated part of the system).

Usually, diagnosis relies on observation of the system (for instance some information stored in log files during execution), and on some a priori knowledge of the

behaviors of a system. However, observations can only be partial: distributed systems are now so complex that monitoring every event in a running system is not realistic. In telecommunication networks, for example, the size of complete logs recorded at runtime grows fast, and can rapidly exceed the storage capacity of any machine, or the computing power needed to analyze them. Furthermore, observing a system may impose additional delays to execution of programs or to communications. The time penalty due to observation of a system also advocates for a partial observation. Hence, monitoring a system means choosing an appropriate subset of observable events, and equip the implemented system to record occurrences of these events. This can be achieved by inserting lines of codes (frequently called software probes) to report the execution of a particular part of a program, adding physical mechanisms that listen to communications and report a specific kind of communication between chosen processes, ... The choice of a particular set of events to be observed is clearly part of the design of a complex system.

Several techniques are frequently called "diagnosis" while addressing different goals. Any kind of technique that provides online or offline information on a system to a supervisor can be called diagnosis, but we will focus more precisely on two of them, namely fault diagnosis, and history diagnosis. *Fault diagnosis* answers the question whether a system is faulty, i.e it has reached some bad configuration or executed some events that should have not occurred. The main challenge in fault diagnosis is to decide whether for given sets of faults and observable events the system is diagnosable, i.e. the occurrence of a fault can eventually be detected after a finite number of observations [131]. Diagnosis is then performed by an observer that monitors observable actions and raises an alarm when needed.

History diagnosis reconstructs an actual set of possible executions of a system from a partial observation. The a priori knowledge on the system available for this is defined as a model of system's behaviors. The objective is then to build a set of plausible explanations (runs of the model) that comply with the observations [24]. Then, these potential explanations can be exhaustively checked to find a fault, or to provide feedback to system's supervisor. Note that fault and history diagnosis try to solve different problems: fault diagnosis tries to infer if a fault has occurred (and very often which fault), while history diagnosis may provide several explanations (faulty or not) for a given observation. Throughout this chapter, we mainly address history diagnosis. Hence, for simplicity, will only use the term "diagnosis" instead of "history diagnosis".

Roughly speaking, the history diagnosis problem can be defined as follows: given a model M of a running distributed system, an observation alphabet Σ_{Obs} and an observation O produced during a run of the system, find all executions of M which projections on Σ_{Obs} embed O. A major difficulty of diagnosis is to find plausible occurrences of non-observed events. These events are explicitly represented in the model of the system, but do not appear in the collected observation O.

The major objective of the work presented in this chapter is to exploit the non-interleaved representation of scenarios to avoid combinatorial explosion, and provide efficient algorithms for diagnosis. Observation of a distributed system needs not be an interleaved model, and as long as an analysis of a system does not need to study all global states, true concurrency models should provide efficient solutions. We will show later that diagnosis with HMSCs can be performed without referring to global

states of the system. We will however see that one can not avoid an exponential blowup in the size of the observation, which linearizations may have to be considered.

The authors of [24] already address history diagnosis with a partial order model (safe Petri nets). In this approach, diagnosis is performed as an incremental construction of an unfolding of the net model. The incremental aspect of this approach is clearly well adapted for online diagnosis. However, unfoldings are an explicit representation of explanations. In some cases, the unfolding depicting all possible explanations provided by a net is infinite, and the incremental construction may not terminate. Consider for instance the Petri net of Figure 9.1-a. Supposing that this simple model is a model for a real system, and that the only observable event is action b, then the explanation for an observation O containing a single occurrence of b is the infinite unfolding of Figure 9.1-b. Indeed

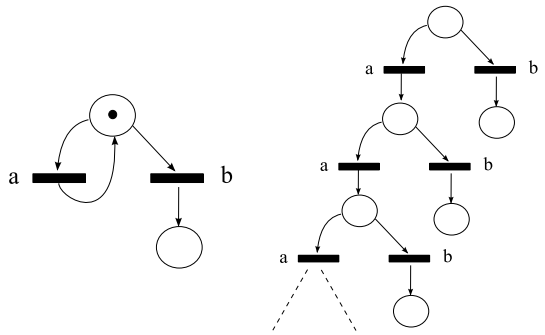


Figure 9.1: a) An example of Petri net, and b) the (infinite) unfolding explaining a single occurrence of action b

As we are considering distributed systems, it is natural to consider that observations of a monitored system are provided as a partial order. The algorithm detailed in this chapter takes as input an observation O given as a partial order, an HMSC model H of the possible behaviors of the system, and the knowledge of the type of events that have been recorded in O to build a product between O and H that generates all possible explanations.

We do not impose restrictions on the observation architecture: observed events occurrence may be collected in a centralized way, or separately by distributed observers. However, we will consider that for a given process, all observed events are totally ordered, that is, probes assign at least a sequence number to each event observed on their monitored process. In addition, probes can implement vectorial clocks à la Fidge& Mattern [47, 101], which allows to recover faithfully causalities among events occurrences. Even when no vectorial clock is implemented by probes, some causal dependencies can be inferred from packets numbering (sending necessarily precedes reception of a packet with identical number), etc... Hence an observation O may contain some particular ordering between events that is not only induced by sequence number on each process. This additional information can be used to refine the set of explanations provided by the model. Indeed, if an event e happens before an event e' in the observation, then in any possible explanation provided by the model, e must be causally related to e'. We also assume that the observation mechanisms inserted in the distributed system are lossless. That is, if an observable event does not appear in the observation, then we have the information that it did

not occur.

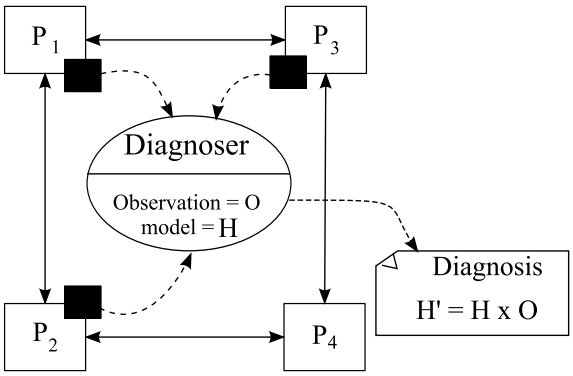


Figure 9.2: Scenario-based diagnosis framework

Figure 9.2 shows an usual diagnosis architecture. The monitored system is composed of 4 processes P_1, P_2, P_3 and P_4, represented by white squares. Communications between these processes are symbolized by arrows between processes. Some sites in the system are equipped with sensors or software probes, represented by dark squares in the figure, that detect the occurrence of some events (a message is sent or received, a timeout has occurred, a program has reached a specified point in its control flow, ...). These events are sent to a centralized mechanism, the *diagnoser*. The communications to the diagnoser can use the communication means of the monitored system, or another network dedicated to this task. We only suppose that no observed event is lost, and that all messages that are sent from a process to the diagnoser respect a FIFO ordering (this can be easily achieved by tagging the information sent with the clock of the monitored process). Only a subset of everything that occurs on a process is monitored. The diagnoser uses a model (in our case a HMSC H) that describes all possible behaviors of the system (or at least a reasonably large subset of them), and builds an observation from the set of all events that it receives. The role of the diagnoser is to output a new model (the *diagnosis*) that defines the set of all executions of the model that are consistent with the observation (the *explanations*). In our case, the output of the diagnoser is a new HMSC that describes all possible explanations of an observation.

As we will show in the next sections an advantage of using HMSCs is that we can finitely represent the set of runs of a distributed system that explains a particular observation O. As demonstrated by the example of Figure 9.2, this is not true for any kind of model. With HMSCs H as model of a system, the diagnosis for an observation O is not a set of explanations, but rather a *finite generator* of all executions of H for which the projection on observed events is compatible with O. More precisely, we show that the set of explanations can be described by another HMSC. This gives the basis of a diagnosis algorithm.

A second interesting result is that a global explanation of an observation involving several processes can be reconstructed from a set of local diagnosis performed for the same observation projected on subsets of processes. This allows for an easy partition of the diagnosis problem into smaller tasks, and hence for a distribution of a global diagnosis task to several diagnosers running in parallel. Within this setting, each diagnoser computes separately the set of executions that can explain

what it has observed. At the end of all local diagnosis, a last step combines the local explanations to produce a global explanation.

3 Observation

Let us now define the essential notions that will be used to find explanations of an observation. An observation O performed during an execution of a system should be an abstraction of an existing execution (i.e. an abstraction of a MSC). We will suppose that on each instance of our distributed system, a subset of events is monitored: every time a monitored event e is executed, a message is sent by a local observer to the supervision mechanism. In the following, we will only suppose that observations are **lossless** (all events that are monitored are effectively reported when they occur), **faithful** (observers never send events that did not occur to the supervising architecture, and do not create false causalities), and received within a **bounded delay** t_{obs}. The set of types of monitored events is defined as an observation alphabet Σ_{obs}. The observations can contain additional ordering information (built from local observations and additional information such as packet numbers, vectorial clocks,...), and are thus considered as labeled partial orders. We consider systems composed of a set $\mathcal{P}$ of communicating processes. Events are not observed on all instances, hence we define a set $\mathcal{P}_{obs} \subseteq \mathcal{P}$ on which events are monitored (i.e. $\mathcal{P}_{obs} = \varphi(\alpha^{-1}(\Sigma_{obs}))$). We will also consider that for each observed process, the observation is a sequence, and that reporting events to the supervision mechanisms does not change the observed ordering, that is, the communication between local observers and the supervision architecture is FIFO. Formally, observations can be defined as labeled partial orders as follows.

Definition 81 *An* observation *is a tuple* $O = (E_O, \leq_O, \alpha_O, \mu_O, \varphi_0)$*, where* $E_O, \leq_O, \alpha_O, \varphi_O$ *have the usual meaning in MSCs and* μ_O *is a partial application that pairs events of* E_O*. Observations have to define a total order* $\leq_{O_p}$ *on each process, as for MSCs, and satisfy the inclusion* $(\mu_O \cup \bigcup_{p\in\mathcal{P}} <_{O_p})^* \subseteq \leq_O$*.*

From this definition, observations are a relaxed version of MSCs with less constraints on ordering: they are not necessarily communication-closed, as some emissions of $E_{O_!}$ may not have an image through μ_O, and some receptions may not be the image of an emission. This is justified by the fact that we do not want to enforce that both the sending and reception of messages are observed. Furthermore, it is not required that the union of local ordering and message mapping forms a transitive reduction of $\leq_O$ as in MSCs. Indeed, all events are observed locally, and nothing guarantees that message sendings are mapped to the corresponding reception, nor that both ends of a message are observed. However, vectorial stamping, packet numbering or similar information exchanged among processes can help building a causal order that is richer than a simple collection of sequences of observed events on each process. Observations can be composed like MSCs using the $\circ$ operator. In the sequel, we will adopt the following graphical convention for observations. Processes will be represented as in MSCs, but without the black rectangle ending the process line. Events will be represented as boxes labeled by the event type, and the covering of the ordering relation will be depicted as arrows between causally related events.

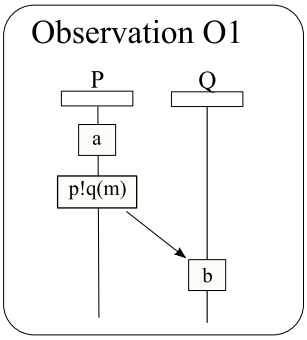


Figure 9.3: An example observation

Figure 9.3 shows an example of observation. Processes P and Q are monitored. In this observation, two events have occurred on P: an atomic action a and the sending of a message m to Q. A single event has occurred on Q, an atomic action b. Note that the sending of message m precedes action b, but that the corresponding reception is not observed. However, any MSC explaining this observation should contain this reception, which necessarily occurs before b in order to explain why $P!Q(m)$ and b are ordered.

For an arbitrary partial order $O = (E_O, \leq_O, \alpha_O, \mu_O)$, we will denote by $<_O$ the covering of relation $\leq_O$, i.e $x <_O y$ iff $\nexists z \in E_O \setminus \{x, y\}, x \leq_O z \leq_O y$. For a given event $e \in E_O$, we will denote by $\downarrow(e)$ the set of all causal predecessors of e. We will also denote by $max_{\leq}(p)$ the maximal event located on process p. Furthermore, slightly abusing the notation, for a set of events E, we will denote by $O \setminus E$ the restriction of O to $E_O \setminus E$, and write $e \in O$ instead of $e \in E_O$. Finally, we say that a set of event $E \subseteq E_O$ is a downward closed subset of of E_O if for all $a \leq_O b$ with $b \in E$, then $a \in E$.

Now that we have defined the observations that are produced by the probes and collected by our diagnosis architecture, let us show how MSCs and HMSCs can be used to explain observations. Intuitively, an MSC M is an explanation of some observation O if M and O agree on the sequences of observed events, and on their respective causal ordering. This is captured formally using notions of projection, prefixes, and suborders. As the definition of projection used in this chapter slightly differs from the definition of chapter 5, we propose a new definition below. It shall be clear that projection in current chapter refers to this new definition.

Definition 82 *Let $M = (E, (<_p)_{p\in\mathcal{P}}, \alpha, \mu, \varphi)$ be a MSC over a set of processes $\mathcal{P}$ and a set of actions Σ. Let $\Sigma_{obs} \subseteq \Sigma$ be an observation alphabet. The* projection *of M on Σ_{obs} is an observation denoted by $\Pi_{\Sigma_{obs}}(M) = (E_O, \leq_O, \alpha_O, \mu_O, \varphi_O)$ such that $E_O = E \cap \alpha^{-1}(\Sigma_{obs})$, $\leq_O = \leq \cap (E_O \times E_O)$, and α_O (resp. μ_O, φ_O) is the restriction of α (resp. μ, phi) to E_O.*

Note that an MSC is also an observation (but the converse is not true). However, what a monitoring system observes from these executions are just projections. Indeed, it is not possible in practice to instrument a system in such a way that any instruction or event occurring on every process is recorded. This also holds for the causal relationships between observed events. Hence, the observed order among

observed events might be smaller than the actual causal ordering of the projected execution. This is captured by the notion of *sub-order* defined below. Clearly, this means that observations are sub-orders of projections of executions on observed events. Furthermore, as the systems described are supposed to be networks of machines communicating asynchronously, the observation mechanisms can also report observed events asynchronously. Some events that have been observed on each site, and sent to the supervision mechanism may not have been received when a diagnosis task is launched. Hence, we have to take into account that the observation collected so far might be extended with events that will arrive later at the diagnosis mechanism. This is captured by the notion of *prefix*.

Definition 83 *Let $O = (E_O, \leq_O, \alpha_O, \mu_O, \varphi_O)$ be an observation. A* prefix *of O is an observation $O' = (E'_O, \leq'_O, \alpha'_O, \mu'_O, \varphi'_O)$ such that $E'_O \subseteq E_O$ is a downward closed subset of E_O and $\leq'_O, \alpha'_O, \mu'_O, \varphi'_O$ are restrictions of $\leq_O, \alpha_O, \mu_O, \varphi_O$ to E'_O. A* sub-order*of O is an observation $O' = (E_O, \leq'_O, \alpha_O, \mu'_O, \varphi'_O)$ such that $\leq'_O \subseteq \leq_O$ and $\mu'_O \subseteq \mu_O$.*

We will say that an MSC M and an observation O are consistent when the observation is something that might have been collected during the execution of M. This can be defined by a *matching relation* between O and M:

Definition 84 *Let $O = (E_O, \leq_O, \alpha_O, \mu_O, \varphi_0)$ be an observation, and $M = (E, \leq, \alpha, \mu, \varphi)$ be a MSC over a set of processes $\mathcal{P}$ and a set of actions Σ. O* matches *M with respect to an observation alphabet Σ_{obs} (denoted by $O \triangleright_{\Sigma_{obs}} M$) if and only if O is a* **prefix of a sub-order** *of $\Pi_{\Sigma_{obs}}(M)$.*

Whenever $O \triangleright_{\Sigma_{obs}} M$, we will say that M is an **explanation** of O (w.r.t observation alphabet Σ_{obs}). Let us detail this definition. We require O to be a sub-order of a prefix of the projection of M. The prefix requirement imposes that when an event is observed in O, all the observable preceding events on the same process have also been observed. This is a straightforward consequence of the assumption that observed events are not lost. Note however that M can still contain observable events that have *not yet* been observed. The sub-order requirement imposes that any causal ordering found in O is actually an ordering described in M (but the converse needs not hold). Note that this matching definition is close to the definition of matching proposed by [112, 115] for verification purposes. The definition of [112, 115] uses a notion of matching relation between two MSCs M and N, that is define a morphism from E_M to E_N. We can define an equivalent notion in our setting, that is build a function that sends events of an observation O onto events of an MSC M. Indeed, the diagnosis from HMSCs will mainly consist in building such matchings.

Proposition 12 *$O \triangleright_{\Sigma_{obs}} M$ if and only if there exists a matching function $h_{O,M} : E_O \longrightarrow E_M$ that sends events of E_O onto events of M such that:*

- *$h_{O,M}$ respects the labeling ($\alpha(h_{O,M}(e)) = \alpha_O(e)$), the causal ordering of O ($e \leq_O f \implies h_{O,M}(e) \leq_M h_{O,M}(f)$), and the locality of O ($\varphi(h_{O,M}(e)) = \varphi_O(e)$).*
- *for every pair of events $e \leq_M f$ in M located* **on the same process**, *and such such that $\alpha(e) \in \Sigma_{obs}$ and $\alpha(f) \in \Sigma_{obs}$, $h_{O,M}(f)$ defined implies that $h_{O,M}(e)$ is also defined.*

Furthermore, when $h_{O,M}$ exists, this function is unique.

Proposition 13 *Let O be an observation and M be a MSC. Then, checking whether $O \triangleright_{\Sigma_{obs}} M$ can be done in $O(|M| + | \leq_O |.| \leq_M |)$.*

Let us illustrate matching on the examples of Figure 9.4, where $\Sigma_{obs} = \{a, b\}$, O_1, O_2, O_3, O_4 are observations, M_1, M_2, M_3, M_4 are MSCs, and the matching relation h_{O_i,M_i} that sends an observation onto an execution is represented by dotted arrows when it exists.

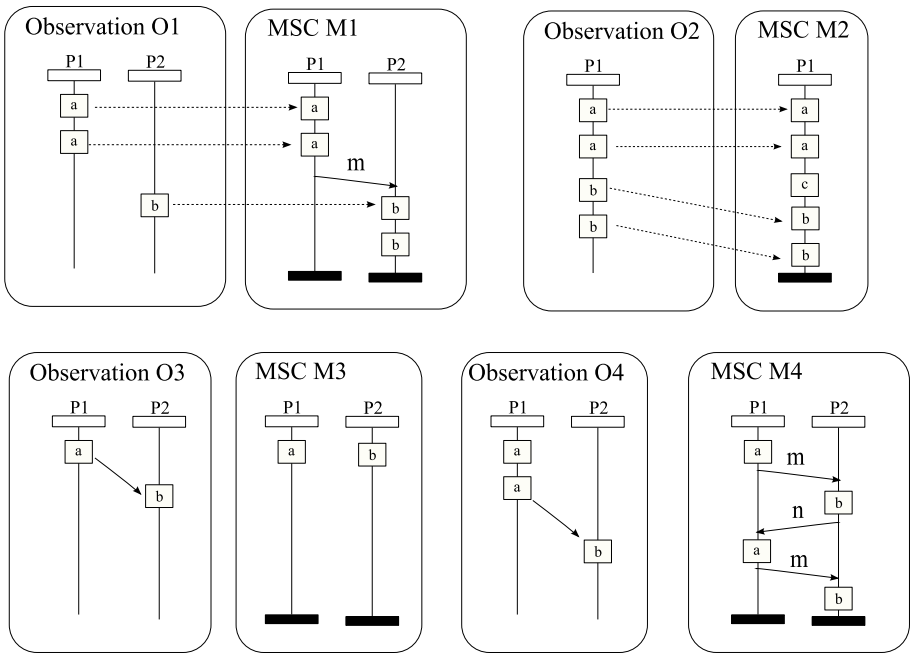


Figure 9.4: Two matching examples w.r.t $\{a, b\}$ and two counter examples

- Let us consider O_1 and M_1: there is an injective mapping from the observation to a prefix of the explanation. a's and b are concurrent in the observation, but the order O_1 can clearly be injected in M_1, hence $O_1 \triangleright_{\Sigma_{obs}} M_1$.
- For the pair O_2, M_2, there is also an injective mapping that maps O_2 to a prefix of the projection of M_2 onto Σ_{obs}. c does not have to be matched, as it is not an observed event.
- For the pair O_3, M_3, a and b are unordered in the explanation M_3 and hence the observation O_3 can not be injected in M_3.
- For the pair O_4, M_4, there is no injective mapping satisfying the three conditions of the morphism defined in proposition 12. Indeed, an occurrence of b should have been observed between two a's. Hence, M_4 is not an explanation of O_4.

From these examples, one may notice that the observation of some events provides information on whether a peculiar execution M is an explanation of what has been observed, but also that the causal ordering of some events in the observation or their absence can also be used to rule out some possible explanations (this is the case for the pairs (O_3, M_3) and (O_4, M_4)).

4 Diagnosis with HMSCs

We can now extend the notion of explanation of an observation O to sets of explanations for O provided by a HMSC.

Definition 85 *Let O be a partial order over Σ_{obs} and H be an HMSC. The set of* explanations provided by H *for an observation O is the set of paths $\mathcal{P}_{O,H} \subseteq \mathcal{P}_H$ such that $\forall \rho \in \mathcal{P}_{O,H}$, $O \triangleright_{\Sigma_{obs}} M_\rho$.*

Notice that the set of explanations provided by H is not always finite. Its linearization language is not necessarily regular, but we will prove that it can be described by an HMSC in theorem 53. The main objective of our diagnosis approach is to extract from an HMSC H a generator for the set of explanations $\mathcal{P}_{O,H}$ of an observation O. These explanations are path of H that generate MSCs that embed O. We will show that we can define a generator for all MSCs that embed O as a product $\mathcal{A}_{O,H}$ between the HMSC and the observation, with synchronization on monitored events. The states of this product are of the form (n, E, γ), where n is a node of H, E a subsets of events of O found so far in a MSC generated by a path that ends in n, and γ some information needed to ensure compatibility of MSCs generated along paths and of the observation. Accepting states of the product are obviously of the form (n, E_O, γ), that show that a complete explanation for O was found. Transitions of the product are of the form $(n, E, \gamma) \xrightarrow{M} (n', E', \gamma')$, such that (n, M, n') is necessarily a transition of H, so we can define $\mathbb{L}_{H,\mathcal{A}_{O,H}}$ as the projection of accepting paths of the product $\mathcal{A}_{O,H}$ on its first component (nodes). We refer interested readers to the appendix for a detailed construction of this product. We can now state the main result of this section:

Theorem 53 *Let H be a HMSC, and O be an observation. Then, one can compute a HMSC $\mathcal{A}_{O,H}$ such that for every path $\rho \in \mathcal{P}_H$, Then $O \triangleright_{\Sigma_{obs}} M_\rho$ iff $\rho \in \mathbb{L}_{H,\mathcal{A}_{O,H}}$. Moreover, $\mathcal{A}_{O,H}$ is of size $O(|H| \times |O|^{|\mathcal{P}|\times|\mathcal{P}_{Obs}|})$.*

More intuitively, this theorem says that the set of explanations for O provided by H is exactly $\mathbb{L}_{H,\mathcal{A}_{O,H}}$, that is $\mathbb{L}_{H,\mathcal{A}_{O,H}} = \mathcal{P}_{O,H}$. Hence, for every accepting path ρ of $\mathcal{A}_{O,H}$, we have $O \triangleright M_\rho$. Note also that paths in $\mathbb{L}_{H,\mathcal{A}_{O,H}}$ contain *all paths* embedding O, and not only the *minimal paths* embedding O. This property is important for compositionality issues.

4.1 Offline Existence

The diagnosis problem can be simplified to answer a simpler question : is there an explanation for an observation O in H? In the sequel, we will refer to this question as the *existence problem*, which can be formalized as follows: given an HMSC H, an observation alphabet Σ_{Obs} and an observation O, $\exists ?\rho \in \mathbb{L}_{H,\mathcal{A}_{O,H}}, O \triangleright M_\rho$.

Theorem 54 *Let H be a HMSC, Σ_{obs} be an observation alphabet, and O be an observation. Deciding whether there exists an explanation for O in H w.r.t. Σ_{obs} is an NP-complete problem.*

Theorem 53 shows that the complexity of diagnosis increases with the size of the observation, and Theorem 54 shows that even a simpler problem such as existence of an explanation can rapidly become difficult to solve. To handle this complexity, we can reduce the size of the observed alphabet, which will hopefully produce smaller observations, or try to split a problem into smaller ones and then combine the results. The following proposition shows that limiting the observation capacities of the system does not produce wrong negatives for the existence problem.

Proposition 14 *[77] Let H be a HMSC, Σ_{obs} be an observation alphabet, and O be an observation. Let $\Sigma'_{obs} \subseteq \Sigma_{obs}$. Then if $\Pi_{\Sigma'_{obs}}(O)$ has no explanation from H w.r.t. Σ'_{obs}, then O has no explanation w.r.t. Σ_{obs} from H.*

This property possibly reduces the time needed to give a negative answer to the existence problem and will be useful later on in the next section to divide the diagnosis problem into smaller sub-problems, and hence reduce the time needed to build a complete diagnosis.

4.2 Splitting the diagnosis problem

The diagnosis framework proposed in section 2, figure 9.2 is centralized (all observations are sent to a central diagnoser that computes the generator for the set of explanations) and offline (the production of a diagnosis is performed once for all from a fixed observation). The main objective of the approach is to perform all calculi on partial order models, and avoid the state space explosion due to an interleaved search in the execution model. Indeed, the interleaved behaviors of the model are never studied, as the construction never considers linearization of MSCs labeling the model H (we refer to the appendix for details on the construction of the diagnosis). However, in worst cases, one may have to consider all linearizations of an observation. Furthermore, the centralized diagnosis amounts to build an automaton of exponential size in the number of considered processes.

A solution to resolve this problem is to split the diagnosis computation into several simpler subproblems, distribute them to a pool of machines, and then combine the results returned by each machine. In this section, we will show a distribution schema that does not change the result of centralized diagnosis nor its worst case complexity, but can allow for a faster detection of non-existence when no explanation of an observation exists.

The main idea is to separate the diagnosis into smaller problems using projections of the observation on subset of processes. From proposition 14, we know that it is sufficient to find no diagnosis for an observation O projected on an alphabet $\Sigma' \subseteq \Sigma_{obs}$ to be sure that no diagnosis exists for O. In particular, this applies to the case when Σ is the restriction of Σ'_{obs} to events that are observable on a chosen subset of processes. The hard technical point is then to combine local diagnosis, and show that their combination produces the same result as the centralized version.

Let $p, q \in \mathcal{P}$ be a pair of instances and $O = (O, \leq_O, \alpha_O, \mu_O)$ be an observation. The local diagnosis for instances p, q is the automaton $\mathcal{A}_{p,q} = \mathcal{A}_{\pi_{\Sigma'}(O),H}$ with the observation alphabet $\Sigma' = \Sigma_{obs} \cap (\Sigma_p \cup \Sigma_q)$. Since an explanation of an observation for some alphabet Σ is still an explanation for any alphabet $\Sigma' \subseteq \Sigma$, we have that

$\mathbb{L}_{H,\mathcal{A}_{O,H}} \subseteq \mathbb{L}_{H,\mathcal{A}_{p,q}}$. Hence, a finer diagnosis can be obtained from successive compositions of local diagnosis. This composition $\otimes$ is simply an intersection, defined as a synchronous product of two diagnosis automata.

Definition 86 *Let $\mathcal{A}_{O,H} = (Q, \delta, \mathcal{M}, q_0, F)$ and $\mathcal{A}'_{O',H} = (Q', \delta', \mathcal{M}, q'_0, F')$ be two diagnosis automata. The* synchronous product *of $\mathcal{A}_{O,H}$ and $\mathcal{A}'_{O',H}$ is denoted by $\mathcal{A}_{O,H} \otimes \mathcal{A}'_{O',H}$, and is the automaton $\mathcal{A}_{O,H} \otimes \mathcal{A}'_{O',H} = (Q \times Q', \delta'', \mathcal{M}, (q_0, q'_0), F \times F')$, where $\big((v, w), M, (v', w')\big)$ is a transition of δ'' iff $(v, M, v') \in \delta$ and $(w, M, w') \in \delta'$, and the two transitions refer to the same transition of H.*

Slightly abusing the notations, we can define $\mathbb{L}_{H,\mathcal{A}_{O,H} \otimes \mathcal{A}'_{O',H}}$ an the set of path of H followed jointly by the two local diagnosis. The next proposition shows that when a run belongs to every $\mathcal{A}_{p,q}$, for pairs of processes in $\mathcal{P}_{obs}$ then it is an explanation of O:

Theorem 55 *For every HMSC H and observation O, we have $\mathbb{L}_{H,\mathcal{A}_{O,H}} = \mathbb{L}_{H,\mathcal{A}_{\otimes}}$, where $\mathcal{A}_{\otimes} = \bigotimes_{p \neq q \in \mathcal{P}_{Obs}} \mathcal{A}_{p,q}$.*

From Theorem 53, we know that the size of $\mathcal{A}_{p,q}$ is in $O(|O|^{2|\mathcal{P}|}.|H|)$. Let us detail how this can impact the diagnosis and existence problems. An immediate idea stemming from theorem 55 is to split computation of $\mathcal{A}_{O,H}$ from O and H into $|\mathcal{P}_{Obs}|^2$ sub-problems, that is compute $\mathcal{A}_{p,q}$ from $\Pi_{\Sigma_{obs} \cap (\Sigma_p \cup \Sigma_q)}(O)$ for each pair of processes $p \neq q \in \mathcal{P}_{Obs}$, and then compute the product of these local diagnosis. To obtain the final diagnosis, we then have to compute a product of $|\mathcal{P}_{Obs}|^2$ automata if none of the local results is empty. Note that the size of a local automaton is not necessarily smaller than the size the original diagnosis automaton computed in the centralized version. The size of the whole product is exactly the same as in the original version. However, the time needed to complete diagnosis is enhanced if some local problems can be computed in parallel, or in any case when one of the local diagnosis returns an empty diagnosis. Indeed, if $\mathbb{L}_{H,\mathcal{A}_{p,q}} = \emptyset$ for some pair $p, q \in \mathcal{P}^2_{Obs}$, then $\mathbb{L}_{H,\mathcal{A}_{O,H}} = \emptyset$, and there is no need to compute other local diagnosis. This allows for a decentralized version of our initial diagnosis architecture, depicted in Figure 9.5.

In this new architecture, there is still a central diagnoser that collects all the observation. Its role is to compute a projection $\Pi_{\Sigma_{obs} \cap (\Sigma_p \cup \Sigma_q)}(O)$, and send it to local diagnosers. Each local diagnoser owns a copy of H, and upon reception of an observation $\Pi_{\Sigma_{obs} \cap (\Sigma_p \cup \Sigma_q)}(O)$, computes a diagnosis $\mathcal{A}_{p,q}$ and returns it to the central diagnoser. The central diagnoser computes incrementally the product $\mathcal{A}_{\otimes}$ of the $|\mathcal{P}_{Obs}|^2$ after each reception of a local result, and returns the final result when all local diagnosis have been completed. It can also return the empty diagnosis as soon as one local diagnoser returns an empty diagnosis. Note that there is no need for $|\mathcal{P}_{Obs}|^2$ local diagnosers, as each of them can compute more that one local diagnosis.

We have shown in theorem 55 that the diagnosis problem can be brought back to a product of smaller local diagnosis problems. A question that immediately arises is whether the existence problem can also be decentralized, and even more interesting, be seen as the conjunction of boolean answers to local existence problems. For the existence problem, we know that as soon as a local diagnosis is empty, H provides

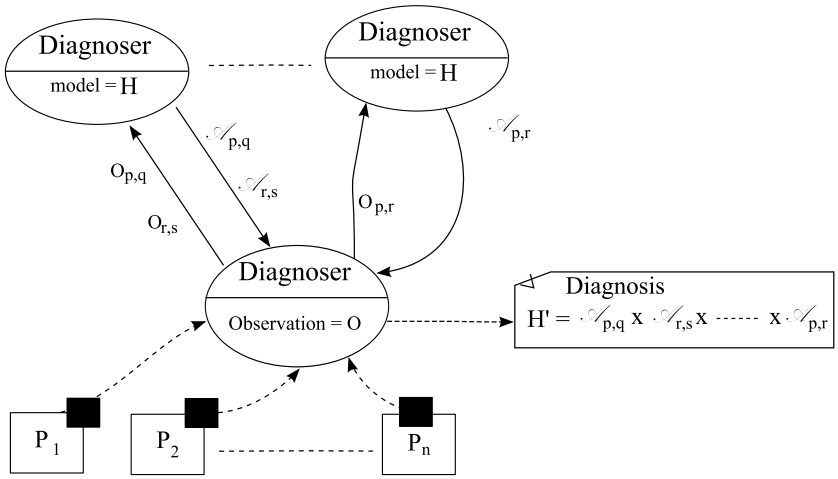


Figure 9.5: Decentralized Diagnosis/Existence problem

no explanation for O. However, a product of *all* local results *have to be computed* to make sure that a global explanation exists for O in H. As for diagnosis, the product can be built incrementally from local diagnosis computed concurrently, and a negative answer can be returned as soon as a local diagnoser returns an empty diagnosis. Note however that there are cases where all diagnosis compute a diagnosis automaton with non-empty language (i.e. $\mathbb{L}_{H,\mathcal{A}_{p,q}} \neq \emptyset$ for every pair $p,q \in \mathcal{P}^2_{Obs}$) but where the global diagnosis is nevertheless an automaton with empty language (i.e. $\mathbb{L}_{H,\mathcal{A}_{O,H}} = \bigcap_{p,q \in \mathcal{P}^2_{Obs}} \mathbb{L}_{H,\mathcal{A}_{p,q}} = \emptyset$) . Consider for instance the HMSC H and the observation O of figure 9.6. For each pair of processes p,q in $\{P1,P2,P3\} \times \{P1,P2,P3\}$, it is possible to find an explanation for $\Pi_{\Sigma_{obs} \cap (\Sigma_p \cup \Sigma_q)}(O)$. However, H does not contain execution that exhibits at the same time events a,b,c.

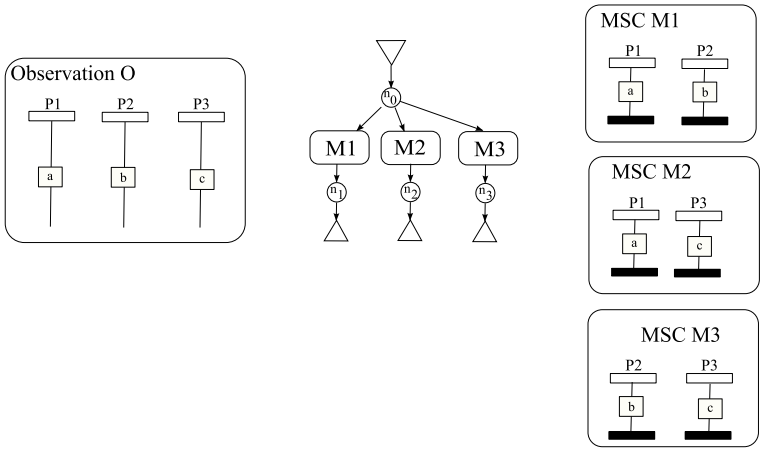


Figure 9.6: An example that shows that computing a product is still needed for decentralized existence

Another solution to decentralized diagnosis is to consider process by process diagnosis, that is compute the automaton $\mathcal{A}_p = \mathcal{A}_{\pi_{\Sigma_{obs} \cap \Sigma_p}(O),H}$ that provides all explanations of H for $\pi_{\Sigma_{obs} \cap \Sigma_p}(O)$ for each $p \in \mathcal{P}_{obs}$. This is particularly interesting,

as the projection of $\mathcal{L}(H)$ on a single process is a regular language, which can greatly simplify the construction of $\mathcal{A}_{\pi_{\Sigma_{obs}\cap\Sigma_p}(O),H}$. Computing this set of automata gives a less precise solution than with pairs of processes, because ordering between events of O located on distinct processes cannot be used to discriminate some paths of H. Indeed, in general $\mathbb{L}_{\mathcal{A}_{p,q}} \subseteq \mathbb{L}_{\mathcal{A}_p} \otimes \mathbb{L}_{\mathcal{A}_q}$, but equality does not hold. However, the initial step computing $\mathcal{A}_p$ is performed with complexity in $O(|O| \times |H|)$. Notice that $\mathcal{A}_{p,q}$ has to be computed only for those $p \neq q \in \mathcal{P}_{obs}$ for which there exist two events $e, f \in E_O$ respectively located on p and q and such that $e \leq_O f$. If no such ordering from p to q exists in O, then $\mathcal{A}_{p,q} = \mathcal{A}_p \otimes \mathcal{A}_q$. This leaves room for improving efficiency of decentralized diagnosis. Such situation can occur if the observation mechanisms only record one sequence of events per observed process, that is the observation can be described as a set of strings.

4.3 Online Diagnosis

The previous section shows how to address diagnosis from HMSCs starting from an observation that is considered as definitive. Such a setting applies post-mortem that is when a system has crashed, and one wants to know which run of the system has been played before crashing, to find explanations for the failure. However, one is frequently interested in online diagnosis that is maintaining incrementally a diagnosis of the running system. Such diagnosis can be used to take decisions (for instance deciding to launch some curative actions in a system when the occurrence of a fault seems unavoidable).

A naive approach is to store in memory the observation collected, and rebuild a complete diagnosis when a new event is collected by the diagnosis system. However, this operation is obviously too costly, as it consists in recomputing a new diagnosis from scratch when a new event arrives. In [52], we show an algorithm that updates a diagnosis when a new event is observed, and removes from memory the part of the observation that becomes useless (i.e. that have already found an explanation, and can not be used anymore to discard future potential embeddings), and the part of the diagnosis that can not be refuted even if new events and causalities are observed. The information removed from memory (in the observation and in the diagnosis) can be stored for later use to build a complete diagnosis. The algorithm can be adapted for online existence checking: in this case, the information removed from memory need not be stored, and hence the algorithm maintains only the information needed to guarantee soundness of the existence check (i.e. the information needed to guarantee that at least one path of the HMSC model explains current observation).

Even if the online setting seems appealing, the worst case complexity to increment a diagnosis is redhibitory. We show in [52] that incrementing a diagnosis of size K can be done in $O(K \times |\mathcal{P}| \times |H| \times 2^{|\mathcal{P}|})$. Hence, computing online a diagnosis for an observation O can be done in

$$O\left(\sum_{i\in 1..|O|} |H| \times (i-1)^{|\mathcal{P}|\times|\mathcal{P}_{Obs}|} + d^{h(i-1)} \times |H| \times |\mathcal{P}| \times 2^{|\mathcal{P}|}\right)$$

where $h(i)$ is the maximal height of the diagnosis automaton built at step i, and d is the outgoing degree of HMSC H.

In fact, incremental online diagnosis can be efficient only if one can ensure that the part of diagnosis that is kept in memory remains bounded. Such a situation is not even guaranteed for regular HMSCs. Overall, the expected gain for online diagnosis was not so appealing, and the online algorithm was not implemented.

5 Diagnosis as a verification problem

Intuitively, an observation describes everything that is "recorded" during the execution of some prefix of a computation. A computation embeds an observation if and only if one can find a prefix of the computation which projection is a partial order that contains O. We can immediately notice that all runs of a model should embed the empty observation. So, if a model has at least one run, it provides an explanation for the empty observation. As a consequence, the diagnosis and existence problems are undecidable for all formalisms for which the emptiness problem (given a model M does $\mathcal{L}(M) = \emptyset$?) is undecidable. This gives the following result:

Theorem 56 *Existence is an undecidable problems for systems described with LPOC, MSO^{MSC}, (non-safe) CHMSCs, CFSMs, or mixed products of HMSCs. Furthermore, for such systems, there is no effective procedure to build a generator of all explanations of an observation O under the form of a HMSC, DMG, or safe CHMSC.*

A second remark is that the embedding relation can be easily translated to an MSO formula. Let us consider an observation $O = (E_O, \leq_O, \alpha_O, \mu_O, \varphi_0)$. Then we can compute an MSO^{MSC} formula φ_O such that a MSC M embed O if and only if $M \models^{\varphi}_{O}$.

- for every event $e \in E_O$ create a variable x_e, and denote the set of all event variables by X_E.
- for every event $e \in E$, let $lab_e ::= lab_{\alpha(e)}(x)$
- for every pair of events $(e, f) \in \mu_O$, let $\varphi_{msg(e,f)} ::= \exists u, v, (u, x_e) \rightarrow (v, x_f)$
- for every pair of events $(e, f) \in \leq_O$, let $\varphi_{e,f} ::= x_e \leq x_f$. One can encode $\leq$ relation as an MSO^{MSC} formula :

  $$x \leq y ::= \exists X, [x \in X \wedge y \in X \wedge \forall z(z \in X \wedge z \neq y \Rightarrow \exists z'(z' \in X \wedge z \lessdot z' \vee \varphi_{msg(z,z')}))]$$

  Similar properties denoting closure of a relation can be found in [29, 55].

At this point, we have enough ingredients to describe all events, labels, messages and ordering of an observation with an MSO^{MSC} formula. However, we have to enforce that a pattern described starts at the first observable events on each process, and that an interpretation association variables of the MSO formula with events of an MSC does not miss any observable action. For this, we assume a macro $SameProcess(x, y)$ that holds if events denoted by variables x and y are located on the same process. Such macro can be easily implemented using properties of labeling, if we assume that the observation alphabet is disjoint for each pair of processes. Otherwise, we can rely on the property that two events located on the

same process are linked by a causal chain that uses only the direct successor relation $\lessdot$ on the same process. We are now ready to define the embedding, using the following fomulae:

- for every pair of events e, f in direct causal successor relation on the same process in O, impose $SameProcess(x_e, x_f)$. We denote by φ_{same} the conjunction of all formulae of this kind.
- $\varphi_{em1} ::= \begin{array}{l} \nexists x_e, x_f \in X_E, y \notin X_E, \\ lab_{\Sigma_{obs}}(y) \wedge SameProcess(x_e, x_f) \\ \wedge SameProcess(x_e, y) \wedge x_e \leq y \leq x_f \end{array}$, where $lab_{\Sigma_{obs}}(y)$ denotes the fact that event denoted by y is an observable action (this can be encoded as disjunction of atomic label formulae).
- $\varphi_{em2} ::= \forall x_e \in X_E, \nexists y, y \notin X_E \wedge lab_{\Sigma_{obs}}(y) \wedge SameProcess(y, x_e) \wedge y \leq x_e$

We then let $\varphi_O ::= \varphi_{same} \wedge \varphi_{em1} \wedge \varphi_{em2} \wedge \bigwedge_{e \in E} lab_e \wedge \bigwedge_{e \leq_O f} \varphi_{e,f} \wedge \bigwedge_{(e,f) \in \mu_O} \varphi_{msg(e,f)}$

Intuitively, M satisfies φ_O means that the pattern O can be observed (embedded) on a subset of processes of M. We implicitly assume that interpretations map distinct event variables to distinct events, which can be either enforced by the diagnosis algorithm, or explicitly specified by additional formulae of the form $x_e \neq x_f$. Note also that so far, processes identities are not fixed in φ_O. One can hence refine φ_O to fix processes identities through labeling, or by adding new atomic formulae of the form $loc_p(x)$ that indicate that the event represented by x is located on process p. However, process identities are considered as important for models such as HMSC, CHMSCs, causal HMSCs, that is for models defining behaviors over a finite and a priori known set of processes. As soon as processes identities are not addressed nor considered as important, there is no need to refine φ_O, one can adapt the diagnosis problem to formalisms such as Dynamic MSC grammars.

Definition 87 *Given a Dynamic MSC grammar G and an observation O over a set of processes $\mathcal{P} \subseteq \mathbb{N}$, the diagnosis of O from G is the set of MSCs $Diag(G, O) = \{M \in \mathcal{F}_G \mid \exists \sigma : \mathbb{N} \to \mathbb{N}, \sigma(M) \triangleright O\}$. The diagnosis problem for dynamic MSC grammars consists in building a recognizer for $Diag(G, O)$, and the existence problem consists if deciding whether $Diag(G, O) = \emptyset$*

Note at this point that we know that in general $Diag(G, O)$ is infinite, as there can be infinitely many ways to name processes during dynamic creation. Furthermore, if M embeds O, any larger MSC $M \circ M'$ also embeds O. Obviously, we have $Diag(G, O) = \{M \mid M \models \varphi_O\}$. As we already know that MSO^{MSC} is decidable for dynamic MSC grammars, we easily obtain the following result:

Theorem 57 *Let G be a DMG, and O be an observation. Then one can effectively build a tree automaton $\mathcal{A}_{G,O}$ that recognize all parse trees of MSCs in $Diag(G, O)$. Furthermore, one can effectively check if $Diag(G, O) = \emptyset$.*

The principle to build $\mathcal{A}_{G,O}$ is to start from the automaton $\mathcal{A}_G$ that recognizes legal parse trees of G, as defined in Theorem 39 of Chapter 6. This tree automaton can then be reused to build a recognizer for parse trees of MSCs that satisfy φ_O,

following the construction of [92] (adapted to DMGs). As for HMSCs, this gives a way to produce a generator rather than a set of MSCs.

So far, we have left as an open issue the question of computing a Dynamic MSC grammar from the tree automaton recognizing the parse trees for $Diag(G, O)$. Note that for a context-free language given by a context free grammar G, one can always compute a tree automaton that recognizes the parse trees of words in $\mathcal{L}(G)$ [37] (theorem 2.4.3). This is also true for the sequences of all possible concatenations of partial MSCs generated by a Dynamic MSC grammar. However, it is not true in general that all tree automata are recognizers for parse trees of some context free languages (the language recognized by a tree automata need to be a *local tree language* for the converse property to hold).

However, we conjecture that the set of trees recognized by $\mathcal{A}_{G,O}$ remains a set of derivation trees of some context free grammar. If this property holds, computing a new grammar from $\mathcal{A}_{G,O}$ can be done at the cost of changing non-terminals, that is defining a grammar which non-terminals are named after states of $\mathcal{A}_{G,O}$, and which rules are derived from the transitions of $\mathcal{A}_{G,O}$. Indeed, a rule of a tree automaton $\mathcal{A}_{G,O}$ is of the form $q \leftarrow A(q_1, \ldots q_n)$, where A is some non-terminal of G or its axiom. We hence need to compute rules and axioms for every kind of state. However, embedding of φ_O in the original tree automaton $\mathcal{A}_G$ may need complementation steps, and decorating a derivation tree with sub-formulae of φ_O. It means generating new sates that do not exist in $\mathcal{A}_G$. Hence, states $\mathcal{A}_{G,O}$ are not necessarily referring only to non-terminals of G. Note also that each complementation may result in an exponential blowup of the number of states used to model check a sub-formula of φ_O. We conjecture that the obtained tree language remains local, which would allow for the design of a (large) DMG, with one derivation rule per transition of the automaton $\mathcal{A}_{G,O}$, that generates sequences of partial MSCs which concatenation embeds O.

A first remark is that the number of states of $\mathcal{A}_{G,O}$ grows at least exponentially at each negation in the formula φ_O. Hence, the obtained grammar is so large that it can not provide useful synthetic information to designer, and hence presents few interest. Furthermore, negations seem unavoidable, as every $x \leq y$ contains a universal quantifier $\forall z, (...)$. A second remark is that computing such a grammar G^O has less interest than for HMSCs. Indeed, consider a DMG G, and two observations O_1, O_2 of the same run of a system produced by distinct sets of probes located at distant sites of a network. One can compute $\mathcal{A}_{G,O_1}$ and $\mathcal{A}_{G,O_2}$, and get back to equivalent grammars, say G_1 and G_2. However, G_1 and G_2 are defined over distinct non-terminals, and one can not decide in general whether there exists an MSC recognized by G_1 and G_2. To decide if some run of G explains both O_1 and O_2, we can however check $\varphi_{O_1} \wedge \varphi_{O_2}$ with respect to $\mathcal{A}_G$. But existence can not be done a priori in a modular way. Similarly, as modular existence looks undecidable, one can not compute a diagnosis from G_1 and G_2, that is a tree automaton that recognizes derivations of G producing a MSC recognized by both G_1 and G_2. Indeed, checking emptiness of this diagnosis tree-automaton would resume to a modular existence.

Hence, diagnosis from DMG is decidable, but with a high complexity. Contrarily to HMSCs, one can not rely on a projection of an observation O to master complexity. Note however that the MSO machinery used to prove decidability of diagnosis is not necessarily needed. Indeed, considering MSO in its full generality, the tree

automaton that is computed for diagnosis may have states referring to subsets of states of $\mathcal{A}_G$ and subsets of sub-formulae of φ_O. Hence, the number of states grows very fast, and takes an exponential factor at each complementation. However, the way an interpretation is mapped onto partial MSCs when interpreting event and process variables need not be chosen randomly, and has can take into account the ordering imposed by processes. Similarly, states may not need to recall all formulae proved true in a subtree, and it might be sufficient to recall the first and last event for each process of an observation that are embedded in a subtree recognized at some node, and may be the causal relations among them. Also, note that if O is simply a set of sequences of events, sub-formulae of the form $x \leq y$ may not appear in φ_O, and hence one may avoid costly complementations. Hence, we strongly believe that diagnosis from DMGs can be performed much more efficiently than within a purely MSO context. The decidability results proved in this chapter shall hence be considered as a first step, and we can now look at efficient algorithms and precise bounds for diagnosis from DMGs.

6 Conclusion

We have seen in this chapter that High-level Message sequence charts provide a nice algebraic framework for diagnosis. From a specification HMSC H and an observation O, we can build an HMSC H_O which is a generator for all explanations of O provided by H. Similarly, one can check for the existence of an explanation in H for an input observation. Diagnosis composes well: the new HMSC H_O can then be used to find explanations for another observation coming from different source than O, and so on. Similarly, one can perform diagnosis for pairs of processes, and assemble the binary results to build the diagnosis HMSC H_O. A similar technique can be used for existence, and the algorithm can stop as soon as one binary diagnosis for a pair of processes p, q fails to provide an answer to $\pi_{p,q}(O)$. This property can be used to distributed and speed up the diagnosis and existence checking problems.

Overall, HMSCs provide a nice framework for diagnosis:

- it is endogenous : a diagnosis for an observation O from an HMSC H is an HMSC
- it has nice properties w.r.t. projection: for $\Sigma_1, \Sigma_2 \subseteq \Sigma_{Obs}$ letting $i \in 1, 2$ we have
$$\mathcal{L}(H_{\pi_{\Sigma_i(O)}}) \supseteq \mathcal{L}(H_{\pi_{\Sigma_1(O)}}) \otimes \mathcal{L}(H_{\pi_{\Sigma_2(O)}}) \supseteq \mathcal{L}(H_O)$$

The complexity of diagnosis from HMSCs is exponential in the number of processes, and grows with the size of the observation, which is in some sense unavoidable. Note however that observation's width is bounded by the number of observed processes. In the worst case, that is when processes in a HMSC are independent, nodes of a diagnosis HMSC may have to remember up to $|O|$ events, leading to the $|O|^{|\mathcal{P}|}$ factor in the complexity. However, if processes never communicate, events located on distinct processes are never causally ordered, and diagnosis can be performed process wise, leading to a better complexity. Similarly, processes in a HMSC may produce observable events at similar rates. In this case, the number of nodes built in a diagnosis is not as large as indicated in theorem 53. However, giving a

finer complexity measure is difficult, and have to take into account the structures of the HMSC and of the observation. In practice, the HMSC diagnosis algorithm was implemented in our tool SOFAT [66], and showed good performance, even for large HMSCs and large observations. The experiments conducted used HMSCs with up to 10 processes, and observations with thousands of events. There is however still a gap to fill between this algorithm and an potential industrial application, as logs generated by real distributed systems (networks management systems, etc.) contain gigabytes of information.

The techniques used for HMSCs have been adapted for causal HMSCs [51], and work for many other formalisms of partial order automata. We have also shown that diagnosis from Dynamic MSC grammars is decidable. However, the decidability results use a translation of the observation to an MSO^{MSC} property. We strongly believe that this framework can be enhanced, and that the usual complexity blowup that stems from MSO checking can be avoided. Yet, one can not hope to obtain better complexity than for HMSCs, as any HMSC can be simulated by a DMG. However, diagnosis from DMGs does not appear to compose well

History diagnosis as introduced in this chapter can be adapted to provide solution for fault diagnosis. Within a HMSC, one can consider some actions or some behaviors as faulty events or faulty scenarios. A fault diagnosis can use the explanation generators to check if a faulty event or a faulty scenario has occurred in some run allowed by the computed diagnosis. This can be done by a simple inspection of transitions of the diagnosis for HMSCs or causal HMSCs, or simply by adding a subformula asking for the presence of faulty event/scenario in the context of Dynamic MSC grammars.

An interesting question is whether one can extend diagnosis to formalisms that do not use an automaton-like support to compose basic diagrams. We have seen that satisfiability of a set of formulae in a logic-based scenario formalisms is rapidly undecidable (see chapter 8), except if one imposes restrictions on the considered runs (universal bounds for LPOC, existential bounds for template MSCs or MSO^{MSC}). Satisfiability and existence problems are close: satisfiability is existence of an explanation for the empty observation. So, existence is in general undecidable for logic-based scenario formalisms. As for diagnosis, one can not expect to have an effective generator for the set of MSCs explaining an observation (otherwise existence would be decidable). However, checking that a MSC satisfies an MSO^{MSC} formula (and similarly for all other logics described in chapter 8) is decidable. So, the diagnosis question can be slightly adapted: the model can be seen as a description of all the knowledge collected about the implementation. For a given MSC M and an observation O, one can to decide if M is a plausible explanation, that is if it is a model of the logical description that embed O. Within this setting, the key question is how to generate candidate scenarios in such a way that interesting set of explanations can be built.

Several other question remain open. A first question is whether one can synthesize monitors for the occurrence of faults from a scenario specification. Indeed, rather than explaining how a fault occurred in a system, it seems useful to equip a system with mechanisms that raise alarms when some undesired behavior occurs. However, such monitoring frameworks will face two major difficulties : first, if we keep as observation architecture the asynchronous observation framework described

in this chapter (see Figure 9.2), nothing guarantees that a monitor can always work with finite memory, even if the observed system is described by a regular HMSC. Indeed, if observable events from distinct processes are reported with different delays to a monitor, the memory needed to faithfully check that a faulty run is compatible with the events observed so far can grow unboundedly. This comes from the fact that any MSC met along a run which observable events have not yet been observed have to be remembered. The last difficulty is to check whether a monitor can always detect a fault in a finite amount of time or after a finite number of observed events have been sent to the supervision mechanisms. When this property holds, we say that the system is diagnosable). Within the context of finite automata, diagnosability has been considered [131], and amounts to checking whether there exists an observation and two infinite runs ρ_1 and ρ_2 of the automaton explaining the observation such that one of them contains a fault and the other one does not. Brought back in the context of HMSCs, this resembles an intersection or a confluence problem, so diagnosability is likely to be undecidable in general.

Another challenge to make scenario-based diagnosis more usable is to be able to work with abstraction of large logs. As already mentioned, many distributed systems that are monitored (network equipments for instance) produce huge logs (gigabytes of stored information). Within this stored information, many things are not interesting, and can be filtered. Similarly, some recurrent sequences of events can be aggregated to form macro-events. Aggregated logs depict causal relations, but the resulting object is not necessarily a partial order. A difficult challenge is to address diagnosis and all related formal activities within this context.

Chapter 10

Application 3 : security

Pour se servir de sa raison, on a besoin de sécurité et de quiétude.
To make use of one's reason one truly needed both security and quiet.
[Patrick Süskind, Le parfum]

1 Introduction

In this chapter, we show an a priori unexpected application of scenario models to security. More precisely, scenarios have shown to be of practical use to detect information flows, or abnormal behaviors (that are sometimes symptoms of an intrusion in a system). Unlike model checking, information flow detection or intrusion detection may accept some imprecision, abstraction, and incompleteness of models. Indeed, security leaks can appear due to specification errors, particular implementation choices, or even be a side effect of a particular system, software, or hardware containing a security breach. Hence, it is commonly admitted that an analysis too can not discover all flaws, and that a certain amount of uncertainty is accepted, provided the technique used is clearly described as optimistic, and may miss security flaws, or pessimistic, and may raise some wrong alarms (one also talks about wrong positives). This possibility to emit a verdict up to some imprecision clearly suits with the characteristics of scenarios, when they are interpreted as incomplete and abstract specifications of a system.

Let us consider the following problem. The behaviors of a real running system R, with set of processes $\mathcal{P}$ is modeled by a specification S. *Information flow* problems ask, for a pair of processes $p, q \in \mathcal{P}$ whether some classified information can be leaked from p to q. Within this setting, p is seen as an user of a system with access to classified information that q should not access. The *anomaly detection* problem consists in observing R as in a diagnosis framework, compare observations collected at runtime, and raise an alarm when no run of the model S can explain an observation O. Intuitively, the model of the system describes normal behaviors, and if there is no explanation for some observation in these normal behaviors, then something wrong (may be an attack) is occurring in the system. The techniques used to recognize an attack range from Bayesian networks, statistical methods, fuzzy inference systems, to data mining techniques.

As highlighted in this document, scenario specification usually represent a subset of possible behaviors of a system. Hence, all results proved with scenarios are up to

incompleteness of the model. However, this should not be seen as a limitation in a security context. For information flows, it is usually admitted that no technique can encompass all kinds of flows, that may appear as a side effect of shared resources, through observation of the discrete or timed behavior of a system. The popular typing techniques(see for example [137] use coarse grain typing rules for programs to detect information flows (a well-typed program is flow-free), and can hence raise wrong alarms. Similarly, many anomaly detection tools raise alarms when an observed behavior resembles an attack schema, and can also raise wrong alarms. This however should not be considered as a weakness of formal security tool: once a suspect behavior is detected by a formal tool, a human operator should ensure that the problem raised by the tool reflects a realistic situation. Even though, formal tools for security are one additional element to increase security.

Scenario-based information flow and anomaly detection techniques presented hereafter have the same limitation, but also have many advantages. First, scenarios emphasize causal dependencies. We will show in the first section that this is an important property to address detection of hidden communication channels. Second, they allow working with a non-interleaved model describing infinite state behaviors. Wrong anomalies can hence not be raised due to arbitrary limitation imposed by a finite state machine (for instance a bounded buffer contents). The non-interleaved nature of scenarios can also help reducing the time needed to search for explanations and emit a verdict.

The works presented in the next section are joint work with Aldric Degorre, Thomas Gazagnaire, and Hervé Marchand.

2 Information flows

Information flows is a problem that raised in the 70's [21, 22], that is in the middle of the cold war. The major fear at this period was that some classified information stored in military information system may leak. Several formal models and analyzes have been proposed to detect some information flows. We can cite, for instance, the Bell& La Padulla model [21, 22], the shared matrices [87]. These early techniques represent resources, their security level, and access rights of users (mainly read or write) on resources, and detect if an authorized user can provide classified information to unauthorized ones using legal accesses of the system to resources.

Nowadays, the information leak problem is not a military problem, but is more a personal issue. The very actual question is how to protect individual data ? Nowadays, formal analyzes use typing techniques and the notion of non-interferences to address the question of information leak. Typing techniques [137] are defined as a type system that accepts programs written in a pseudo language, with while, if, control structure, variables, and operations on these variables. Variables are classified according to a security level. High-level variables should be kept confidential, and low-level variables are considered as public. A typing system types correctly a program P if there is no execution of P that can leak information on the value of confidential variables to public ones.

In the next paragraph, we rapidly define non-interference, in order to highlight the difference between interference-based techniques and the covert channel detection algorithms proposed in [67]. Readers interested by formal techniques for secu-

rity, non-interference, and typing may consult [129] for a survey.

2.1 Non-interference, the traditional approach

Non-interference originally introduced in [62]. Informally, the definition given by Goguen & Messeguer was:

" An user p interferes with a user q in a system S if what p does in S can affect what q can observe or do."

Let us formalize this intuitive notion. Consider a system S given as a formal specification. S can be seen as a Labeled Transition System $S = (Q, q_0, \Sigma, \delta, \varphi)$, where Q is a (not necessarily finite) set of states, q_0 is an initial state, Σ is a set of actions of the system, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, and $\varphi : \Sigma \to \mathcal{P}$ localizes each action on a particular process. The *projection* of S on events of process $p \in \mathcal{P}$ is $\pi_p(S) = (Q, \Sigma, \delta', \varphi)$ where δ' replaces occurrences of actions in $\varphi^{-1}(\mathcal{P} \setminus \{p\})$ by an unobservable action ε, and leaves other transitions unchanged. In short, $\pi_p(S)$ hides all actions that are not observed/executed by p.

The restriction of S to transitions that are not performed by process $p \in \mathcal{P}$ is $S_{\setminus\{p\}} = (Q, \Sigma, \delta', \varphi)$ where $\delta' = \{(q, \sigma, q') \mid \varphi(\sigma) \neq p\}$ forbids transitions labeled by an action of p. Intuitively, $\pi_p(S)$ is what process p can observe from a system, and $S_{\setminus\{p\}}$ is what the system S can do when p does not perform any action.

Let $\bowtie$ be an equivalence relation over labeled transition systems. Such equivalence can be trace equivalence, bisimulation, etc,... We will say that p interferes with q in system S for a chosen relation $\bowtie$ iff:

$$\pi_q(S) \bowtie \pi_q(S_{\setminus\{p\}})$$

Intuitively, this formulation means that q can differentiate a system in which p performs no action from the original system in which p behaves normally. One can notice that there exists a notions of interference for each equivalence of LTS. In some sense, choosing a particular equivalence $\bowtie$ characterize the observation power of process p. We refer interested readers to [49] for a classification of interferences, and to [129] for a survey on information flow properties.

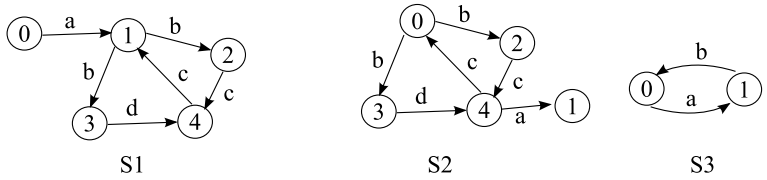


Figure 10.1: Examples of systems

It is frequently written that interference can be used to characterize covert channels. Let us recall that a covert channel appears when a pair of user can have unauthorized information exchanges through legal use of the system. According to [110], a covert channel is "the capacity for a process p to transfer a message of arbitrary size to another process q in a bounded amount of time". This definition really differs from that of interference. First, it supposes a deliberate will to send information, and second, in also supposes that the information sent can be of arbitrary size. An interference from p to q is detected as soon as a process q can detect

that process p has executed one action. That is, if p performs one action and them remains inactive, the system is still interferent. Another drawback of interference is that this notion does not distinguish deliberate actions from p performed to establish a covert flow to q, an action of p which consequences can be observed by q non-intentionally, and coincidence between behaviors of p and q that are not even causally related.

Let us consider the examples of Figure 10.1. The system starts in state 0, action a is performed by process p, and actions b, c, d are performed by process q. Let us chose as equivalence the trace equivalence : S_i is equivalent to S_j iff $\mathcal{L}(S_i) = \mathcal{L}(S_j)$. According to the definition of non-interference, p interferes with q in system $S1$, q interferes with p in $S2$, and $S3$ has an interference in both directions. However, should these systems be considered as containing a covert channel. Obviously, in $S1$ process q can simply learn if action a has occurred. In $S2$, process q can learn if at least one action in $\underline{,}c, d\}$ has occurred. These are not covert channels. The third system $S3$ *may* provide some covert communications if the time elapsed between occurrences of actions can be observed. However, no notion of time appears in the model nor in the chosen equivalence.

Millen has proposed a framework [107] in which covert flows of information are defined using mutual information. Considering two random variables X and Y with values respectively in $\mathcal{X}$ and $\mathcal{Y}$, and a joint probability distribution, $p(X, Y)$ (the probability to see a pair of values for X, Y from $\mathcal{X} \times \mathcal{Y}$. The mutual information between X and Y is defined as:

$$I(X, Y) = H(X) + H(Y) - H(X, Y)$$

where

$$H(X) = \underset{x_i n \mathcal{X}}{\Sigma} p(x) log(p(x))$$

$H(X)$ is known as the entropy of X, and represents the level of uncertainty of variable X. Millen's definition applies to finite sate machines, and states that for sequences of actions W_k of length at most k allowed by a system, if $I(\pi_p(W_k), \pi_q(W_k)) > 0$, then a covert channel exists.

Though mutual information may reveal whether some actions of a process can be observed by another (and provide a quantitative measure on this kind of leak), this definition does not yet suits the definition of covert channel. Consider, for instance the example of Figure 10.2, where a, b are actions of process p, c, d are actions of process q, and x, y actions of a third process r. Clearly, from what q observes, he can infer immediately the sequence of actions played by p. The mutual information between sequences of actions of p and q is obviously not null. However, actions of q are not consequences of actions of p, but rather consequences of a choice of another process r; Hence this situation can not be considered as a covert channel from p to q, because *there is no causal relation from events of p to events of q*.

2.2 The covert channel game

Considering the limitations of non-interference frameworks, we have proposed a covert channel detection framework based on local HMSCs [67]. The proposed definition says that a covert channels from p to q exists in an HMSC H if there is a

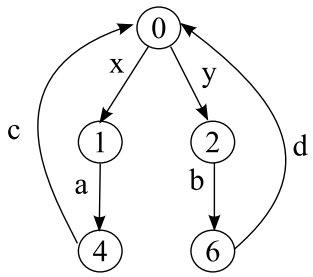


Figure 10.2: A finite state system that contains interferences, but not a covert channel

strategy for process p to perform an arbitrary number of binary decisions (that is choose one branch of a choice among two possible branches) such that the causal consequences of choosing one branch or the other induces different observable consequences on process q.

Let us start with the introductory example of Figure 10.3 This HMSC is a simple communication protocol to transfer data from a process *Sender* to a process *Receiver*. Transfers use short or long data packets, that are chosen according to the network's congestion. Once a session is opened, a user can send short data packets, which are forwarded as data packets to the receiver, or long data packets, which are split into two kinds of packets: $DataInc$, meaning incomplete $Data$ packets, followed by $Data$ Packets.

Now, by choosing long or short data packets regardless of the network's status, process *Sender* can encode 0 and 1, and hence add information over a legal data flow. The receiving process can then decode a message by observing the respective order between $Data$ and $DataInc$ packets. Note that information transfer is only possible if processes *Sender* and *Receiver* have agreed on a protocol for sending information. Note also that as a message can be of arbitrary length, one needs to be able to perform an arbitrary number of decisions for encoding it. Note also that being able to perform two decisions is not sufficient to transmit information. The choices must have different observable consequences for the receiver. Suppose that $DataInc$ packet are replaced by $Data$ packets. Then, upon reception of 3 data packets, it would become impossible for a receiver to be sure whether message "0.1" or message "1.0" was sent. Hence, despite the two possible decisions, the covert protocol cannot be used to transfer reliably covert data. Of course, the example of Figure 10.3 is rather simple, and a use of this channel can be easily discovered, as the sending process does not react properly to network congestion, as required by the protocol.

We can define informally a covert channel from p to q in a HMSC H as a way to encode covert information with decisions of p , and decoded it on process q. In fact, the receiving process q must observe what happens in the system, and deduce the choices performed by the sending process p. We will discuss several ways to decode information at the end of this section. A simple strategy to encode data is to perform choices that ensure that the protocol will eventually get back to the same decision point. This was proposed as a first covert channel identification procedure in [78]. The main idea behind this definition of covert channels is that corrupted users can

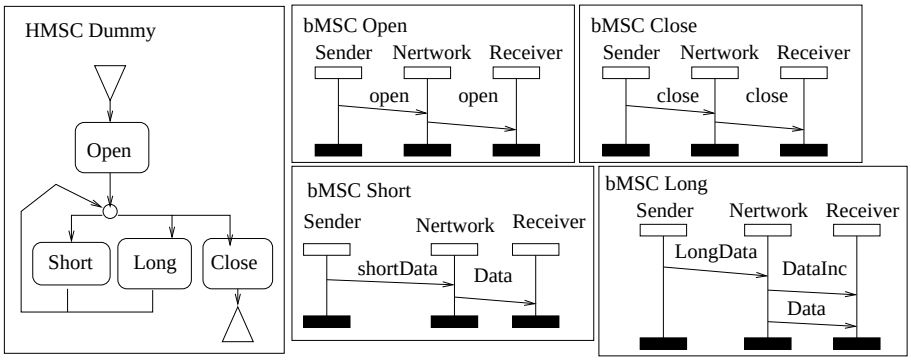


Figure 10.3: The DummyIP protocol

exploit iterations in protocol's behavior to get back to states from which a decision can be taken by the sender in the covert channel, and which causal consequences can be observed by the receiver.

However, with some knowledge of a system, a pair of attackers may have more elaborated strategies to transfer covert data, for which monitoring becomes difficult. These strategies consist in moving the systems towards multiple decision points from which information transmission is always possible. In the rest of this section, we will formalize this definition of covert flows in HMSCs. Consider the example of Figure 10.4, supposing that processes *Sender* and *Receiver* are dishonest users trying to transfer covert data. Let us decide that encoding 0 at choice node n_1 can be performed by choosing scenario *Data*, and 1 by scenario *Wait*. Both corrupted processes can agree to get back systematically to decision node n_1 by executing scenario *Restart* from node n_2, after choosing scenario *Wait*, hence allowing another bit transmission. However, at node n_2, executing *Restart* or *Resume* is another encoding possibility. This gives the possibility to enrich the set of behaviors used to pass covert information, and hence make detection harder. Note that a covert channel can be implemented if the communicating parties agree on a set of choice nodes that will be used to encode information, and on which behavior must be executed to encode a bit in each decision node. In fact, these encoding strategies can be considered as a game between a pair Sender/Receiver, and the rest of the protocol. The attackers win if they can transmit any message of arbitrary length, and the protocol wins if it can prevent messages from being passed.

A cover channel can be seen as a strategy of the sender process to pass infinitely often through choice nodes that can be used to encode information. For every choice node n controlled by process S, the process can deliberately chose to avoid a subset of branches of the $HMSC$ starting from n. This can be encoded as a memoryless strategy $\sigma : N \to 2^T$ (where T denotes the set of transitions).

Given a HMSC H and a strategy σ, one can compute a subgraph of H denoted by H_σ, that describes the possible behaviors of the system when process S follows strategy σ.

Definition 88 *Let H be a local HMSC, $p, q \in \mathcal{P}$ be a pair of processes, and σ be a strategy of S for p. Let n be a node of H controlled by process p. Then, node n is an* encoding node *for processes p, q w.r.t strategy σ if it for every pair of paths ρ_1, ρ_2 of H_σ starting with distinct transitions $t_1, t_2 \in \sigma(n)$, $\pi_q(\rho_1) \neq \pi_q(\rho_2)$.*

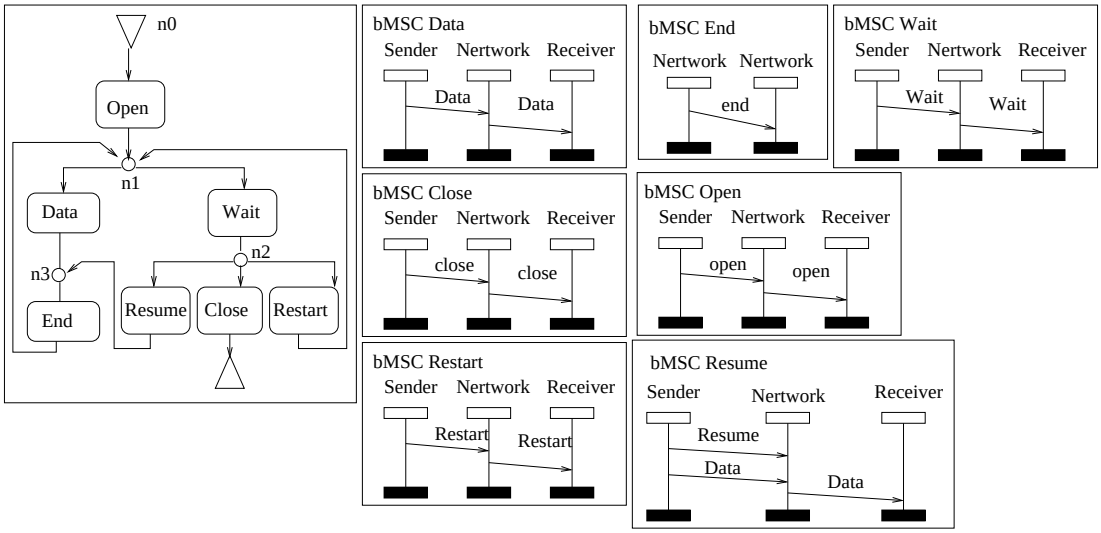


Figure 10.4: Protocol containing a covert channel involving two decision points

Definition 89 *There exists a* potential covert channel *in node n from process p to process q iff starting from node n, p has a strategy σ that allows to pass infinitely often through encoding nodes of p, q in H_σ.*

Theorem 58 *One can effectively decide the existence of a potential covert channel.*

The decision procedure can be brought back to some kind of Muller game. The full algorithm to compute the strategy is not presented here, but can be found in [67]. The main difficulty of the algorithm is that defining a strategy decides at the same time if some nodes will be reachable, and if some other nodes are encoding nodes.

At this stage, we have only considered encoding of information. The process p, chosen as a sender in a covert communication protocol takes decisions that might be observed by the receiver q of the cover flow. Taking decisions means choosing a set of path $\{\rho_1, ...\rho_n\}$, as even if reaching encoding states in a finite number of decisions after a decision of p is guaranteed, some decisions in H_σ can be taken by other processes. Now, the question is whether, for a fixed ρ, path of H_σ, process q is able to retrieve from $\pi_q(\rho)$ the sequence of decisions taken by p. There is no unique solution to guarantee it. In [67], we show that one can build a transducer T that takes as input accepts as inputs the possible observations of q in H_σ and outputs sequences of decisions of p. If T is functional, that is is outputs a single sequence of choices per input word, then the potential covert channel is effective. Another possibility is to use information theory. Indeed, if one can show that the capacity of an information channel that takes as input actions of p and outputs observations of q is not null, then the potential channel discovered in H is effective. An advantage of using information theory is that efficient techniques to build encoders and decoders for a channel while maximizing the capacity of the transmission are known. This setting was used, but in the context of finite automata in [72].

Coming back to the differences between interference problems and covert channel detection, let us consider the example

The work on covert information flows highlighted interesting features that were not considered in standard covert flow detection techniques proposed for finite state

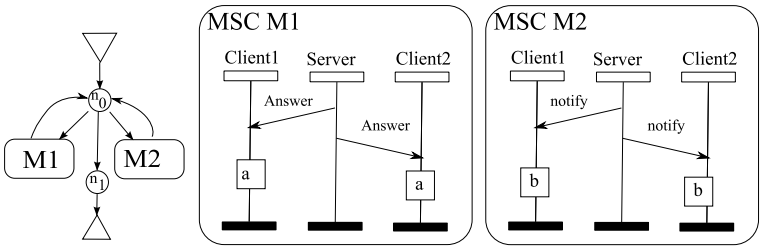


Figure 10.5: A HMSC that does not contain a covert channel from *Client*1 to *Client*2.

systems. First of all, local HMSCs allow for specifications with infinite state space, and yet, covert flow detection remains decidable. Second, this work showed the importance of causality in the definition of covert information flows. This study showed that scenarios can be of practical interest for security applications, and also demonstrated that causality is an important aspect for security. An important remark is that the covert channel definition proposed in this section uses *local* HMSCs. As demonstrated in chapter 7, local HMSCs can be implemented. It then means that a covert channel appearing in a local HMSC specification is likely to appear in a correct implementation of this HMSC.

3 Anomaly detection with diagnosis techniques

The diagnosis framework shown in previous sections is originally designed to help debugging a distributed application when a fault has occurred. In this section, we address another possible application of diagnosis for security. In a diagnosis framework, the model represents the expected behavior of the system. We have already mentioned that the diagnosis obtained from an observation may produce an empty set of explanations. In a debugging context, this can be bothering, and means that our model is not complete enough to provide explanations for a given observation. If on the contrary, we consider that our model is a complete representation of the normal use of a system, then, finding no explanation for a given observation means that the currently observed execution **is not** a normal behavior, and that our system may have security problems: it is attacked by an intruder, a process is corrupted, ... We will show in this section how diagnosis can be used in the context of *anomaly detection*.

Since the 80's several intrusion detection systems (IDS) have been proposed. These systems recognize attacks by monitoring message exchanges in a system, and comparing them with signatures (abstract representation) of attacks. Huge databases of attacks have been collected [42, 118].

However, IDS are trained from datasets, and hence can not discover novel attacks. For this reason, a new complementary solution called *anomaly detection* has been proposed. Anomaly detection relies on comparison of observations with a description of a normal situation. The assumption is that when an attack occurs, it usually exploits weaknesses of a system, that are rarely used in normal conditions. This is peculiarly true for denial of service attacks, where the rate of some requests suddenly becomes unusual. Hence, a detection of some unusual behavior in a system

is assumed to be a potential attack, and raises an alarm. The main challenge here is to establish a profile of normal behaviors. This can be done in the same ways as for IDS, using statistical techniques [130], or with a specification-based approach. [90] proposes a logical framework to define normal behaviors, and [132] proposes a definition of normal behaviors using extended finite state machines. Several surveys on IDS and anomaly detection have already been published, and we refer interested readers to [17, 85, 86] for more information.

The scenario-based anomaly detection framework proposed hereafter uses partial order diagnosis techniques to detect abnormal behaviors while avoiding costly interleaved representations. The main idea behind the solution proposed in this section is to observe the running system, and to compare the observation with a set of predetermined "standard" behaviors, defined as a collection of HMSCs. When an observed run can not be described as a superposition of standard executions, then it is considered as suspect. This can be compared with diagnosis techniques, and we will show in the sequel that scenario-based anomaly detection can be brought back to the existence problem.

3.1 Monitoring architecture

The anomaly detection framework proposed hereafter relies on the diagnosis technique and on the architecture proposed in chapter 9 to compare the observations with HMSCs descriptions of normal behaviors. The framework we consider is a distributed system, composed of several sites (or processes) $\mathcal{P} = P_1, \ldots, P_n$, providing distributed applications $\mathcal{D} = A_1, \ldots, A_k$ to a set of users $\mathcal{U} = U_1, \ldots, U_q$. This system is monitored by inserting probes on each site as described in chapter 9, with the only difference that the diagnoser will compare observed executions with *several models*.

Each user can run several applications of the system, according to a predefined policy. Whenever a user $U_i, i \in 1..q$ uses an application $A_j, j \in 1..k$, we depict the normal use of application j by user i as a HMSC H_{ij}, and define an observation alphabet Σ_{ij}. Indeed, it has been observed that an user's behavior is very often the same when using an application: only a subset of functionalities is used, frequently in a certain order, etc. All these facts can be collected as a *profile*, and in our case as a profile HMSC H_{ij}. We set $H_{ij} = (N_{ij}, \longrightarrow_{ij}, \mathcal{M}_{ij}, n0_{ij}, F_{ij})$, where all MSCs in $\mathcal{M}_{ij}$, are defined over a subset of $\mathcal{P}_{ij} = U_i \cup \mathcal{P}$. More intuitively, the interactions depicted do not involve other users of the system. We also require that $\Sigma_{i,j} \cap \Sigma_{k,l} = \emptyset$ for every $(i,j) \neq (k,l)$. This may seem restrictive, but if we consider that all communications and events during the use of an application are tagged with a unique identifier (for instance the pair (i,j) of user identity and application), this property is immediately met. Note also that we could use more general HMSCs involving several users and several applications without changing the techniques described hereafter.

The system is instrumented to detect the occurrence of some events with signature in $\bigcup_{i \in 1..q, j \in 1..k} \Sigma_{ij}$ and to send them to a centralized diagnoser that logs all events. This is the usual diagnosis architecture defined in chapter 9. Here again, the observation mechanisms can provide some causal ordering among events, and events located on a given process are totally ordered. The collected observation is compared with the models of legal use of all applications by the diagnoser. This

monitoring architecture is depicted in Figure 10.6.

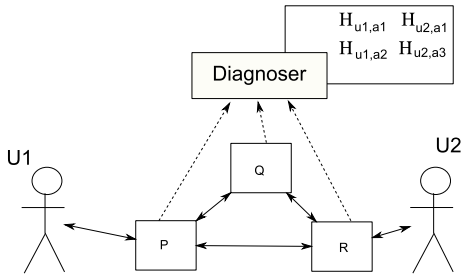


Figure 10.6: Architecture of the anomaly detection framework

3.2 Anomaly detection with Diagnosis techniques

The main difference with the framework of chapter 9 is that observations collected to detect anomalies mix uses of several applications by several users. We hence need to recover the order associated to each pair $(user, application)$ that is contained in an observation, but forget the causal ordering that is due to messages exchanges from other users and applications. This is however not captured by the definition of projection, and we need to define a new restriction operation to separate the observation of different users and applications:

Definition 90 *Let $O = (E_O, \leq_O, \alpha_O, \mu_O)$ be an observation defined over a set of processes $\mathcal{P}$ and over an alphabet $\Sigma = \Sigma_{ij} \cup \Sigma'$. The* restriction *of O to Σ_{ij} is an observation $R_{\Sigma_{ij}}(O) = (E_{O'}, \leq_{O'}, \alpha_{O'}, \mu_{O'})$ such that $E_{O'} = E_O \cap \alpha^{-1}(\Sigma_{ij})$, $\mu_{O'} = \mu_O \cap E^2_{O'}$ and $\leq_{O'} = \left(\{(e, e') \in E^2_{O'} \cap \leq_O |\ \varphi(e) = \varphi(e')\} \cup (E^2_{O'} \cap <_O)\right)^*$*

The restriction preserves the ordering on processes, but only the covering or the ordering relation for events with label in $\Sigma_{i,j}$, to avoid the causal dependencies inherited from other applications or users. Indeed, we may find a sequence $e \leq_O f \leq_O g \leq_O h$ such that e, h have labels in $\Sigma_{i,j}$, f, g have labels in another alphabet $\Sigma_{i',j'}$. As the ordering in O is closed transitively, we have $e \leq_O h$. However, we can not use this information confidently, as this ordering may occur even if e does not precede h in any use of application j by user i according to H_{ij}. Let us now show how the existence algorithm can be used for anomaly detection. Considering that a HMSC H model represents *all* legal behaviors, a negative answer to the existence problem for an observation O can be interpreted as the fact that O is an observation of an *illegal behavior*. The detection can be performed either:

- offline, that is after recording an execution, the anomaly detection algorithm is run to discover whether this execution contains an attack
- online, that is a monitoring systems analyzes current execution and raises a warning as soon as an anomaly is detected.

Based on the observation architecture described in section 4, we can define an offline detection framework for unusual behaviors. First of all, we can notice that if

we use a HMSC H_{ij} to describe the behaviors attached to user u_i and to the system when running application j, we need to allow this user to run this application several times. Furthermore, an attack is not necessarily contained in a single use of an application. We then have to compare several successive (mis-)use of an application with normal use. This can be defined by computing a cyclic version of H_{ij} denoted by H^*_{ij}, that is simply the iteration of H_{ij}, as defined in chapter 5.

We will consider that there is an anomaly when an observed behavior can not be explained as a mix of all legal behaviors defined by HMSCs in $\{H^*_{ij}\}_{i\in 1..q, j\in 1..k}$. This is justified for anomaly detection, as we can consider that an attack exploits unknown (and then unused) weaknesses of a system, and that an attacker does not necessarily have enough knowledge of the system and of users profiles to generate an attack that resembles a legal use of the system.

Definition 91 *Let $U_1, \ldots U_q$ be a set of users of a system composed of processes $P_1, \ldots, P_n$ and applications $A_1, \ldots A_k$. Let O be the observed behavior of the system. Then U_i has an* unusual behavior *in O when using application A_j if $R_{\Sigma_{ij}}(O)$ has no explanation in H^*_{ij}. An observed behavior contains an* anomaly *if and only if at least one user $U_i, i \in 1..q$ has an unusual behavior when using an application $A_j, j \in 1..k$.*

From this definition, we can immediately derive an anomaly detection algorithm. From an observation O fro every pair of users and applications i, j run the existence algorithm of chapter 9 using $R_{\Sigma_{ij}}(O)$ and H^*_{ij}. If the existence algorithm returns a negative answer, then there is an illegal use of application j by user i. Using the results of theorem 53, we immediately obtain the following complexity:

Theorem 59 *Let $\{H_{ij}\}_{i\in 1..k, j\in 1..q}$ be a set of normal behaviors of a system composed of q users, k applications, and n processes. Then, offline anomaly detection in an observation O can be performed in $O(k.q.h.|O|^{(n.p_{obs})})$, where h is the size of the largest HMSC in all H_{ij}'s and p_{obs} is the maximal number of process observed in all H_{ij}'s.*

We have assumed for convenience and efficiency that all observation alphabets were disjoint. Within this setting, the conclusion of the analysis is obvious: when all restrictions $R_{\Sigma_{ij}}(O)$ have their explanation in H^*_{ij}, no alarm is raised, and when a single projection have no explanation, an alarm must be raised. Hence, if we consider that observations are faithful, that is all observed events really occurred, no observation is lost by the supervision architecture, and the order among them is contained in the order of the execution, then we can not have wrong positives: when an alarm is raised, the actual execution that has produced the observation is not a mix of MSCs provided by the models. Note however that wrong negatives can still occur. This is not surprising, as observations only record a subset of all events that have occurred during an execution, and similarly only a subset of causal ordering that occurred among the observed events. Hence, an observation might be compatible with some MSC generated by each model, but not the actual execution that lead to this observation. Hence, the causal ordering among events in an execution might contradict any explanation provided by the models (it may contain different events, and different causal ordering among observed actions), but the recorded information might still allow for explanations.

The disjoint alphabets assumption can be relaxed, but forces considering all possible assignment of each event of O to a pair i, j of user and application. This means that in the worst case (when all Σ_{ij} are equal), we have to apply diagnosis techniques to up to $(q.k)^{|O|}$ different interpretations of the observation. The exponential blowup is not the only problem with overlapping observation alphabets. If none of the possible assignments show unusual behaviors, then the observation corresponds to an interleaving of projections of normal executions. If some (but not all) assignments exhibit an anomaly, then two possible verdicts can be returned by the algorithm: one can consider that as there is a possibility of abnormal behavior an alarm should be raised, or conversely, as at least one assignment of events provides an explanation for the observation, the observed execution was normal.

Offline detection can be used when something went wrong in a system, to make sure that the reason for a failure, for data corruption, or something bad that occurred is not due to an attack. However, detection mechanisms find their full interest when they can be used online to monitor ongoing executions. Online detection mechanisms must raise alarms when an anomaly is detected. Then a supervisor, that might be an automatic process or a human operator has to analyze the threat and react accordingly. The decision that follows an alarm depends on the analysis performed by other detection mechanisms, on the severity of the supposed attack, but also on the security level that one wants to provide for a system, and may range from closing a session, banishing an user or an IP address from the system, to switching off the whole system.

Similarly to offline anomaly detection, online anomaly detection is an adaptation of the online existence problem. The main difficulty here is to maintain several copies of the online existence algorithm (one per H^*_{ij}), and to feed these monitors with the correct observed events. In a setting where all observation alphabets are disjoint, this is not a problem. In case these alphabets are not disjoint, the solution consists in assigning an observed event to a pair (user,application) and create a copy of all diagnosers for every possible assignment. However, the cost of this solution is rapidly prohibitive.

4 Conclusion

This chapter shows that scenario models can be of practical interest for security application. Traditional approaches usually ignore causality aspects, which is (in our opinion) an important aspect to characterize fr instance covert information flows. Similarly, when security properties of distributed systems are defined in terms of pattern matching techniques (as for anomaly detection), one should try to use non-interleaved representations with decidable properties, and HMSCs are good candidates. We shall be more careful with properties defined in terms of equivalences of models, such as non-interference. In the equivalence concerns one single process, then non-interference may be decidable for all scenarios models which projections on processes are regular. However, this is not the case for causal HMSCs, or MSC grammars.

It is not yet clear how results on security aspects of this chapter can be applied to extensions of HMSCs. Anomaly detection clearly applies to any extension for which MSO^{MSC} is decidable. Information flow techniques could be improved by adding

quantitative evaluation of leakage. Indeed, knowing that a covert flow exists is not sufficient. If the detected flow has a very low capacity, then no action should be taken to correct it. On the contrary, if a covert flow is important, something should be done to close it. Possible solution may come by considering quantitative games, where moves may provide rewards in terms of improvement of a channel capacity.

Anomaly detection techniques find their full interest when they can be used online. However, online diagnosis techniques with scenarios are memory consuming. So a natural question is how to improve the anomaly detection framework proposed in this chapter to bound the amount of memory used by the detection algorithm, may be at the cost of some imprecision.

Chapter 11

Conclusion and future work

Sauf pour les dictateurs et les imbéciles, l'ordre n'est pas une fin en soi.
Except for dictators and fools, order can not be an end in itself.
[Michel Audiard]

1 Some side material

It is always difficult, when writing a summary of 10 years of research to choose elements of interest in the available material. Most of my research conducted this last decade was around scenarios, and appears in this document. However, some side works on security, web-services, or robustness of models influenced the research conducted on scenarios, and helped us draw some conclusions on the efficiency and effectiveness of scenario based approaches, and on future research directions. In this last chapter of the document, we first mention works that seem important around security, time, robustness, and Web Services. We then synthesize some lessons learned from the research on scenarios. We then conclude with some research directions.

1.1 Security

A line of research has been devoted to finding hidden information flows from formal models of protocols. The work in [78] was the first attempt to find covert channels in HMSC descriptions, and was later improved in [67], where we showed that covert channel detection could be brought back to the existence of a winning strategy in a game, with an encoding/decoding scheme derived from this strategy. The lessons learned from this work were that causality is an important aspect to consider in information flow to differentiate between leakage and covert channels. This experience was published in [73]. However, the work in [67] was preliminary, and led us to consider information theory as a more general tool to discover covert flows. Our primary intent was to use scenario descriptions and quantitative games. The main idea was to see rewards as gains in terms of mutual information between a sending process and a receiver in a covert channel. However, this idea was not tractable for scenario models such as HMSCs, because the mutual information gained at each MSC concatenation depends on the whole past. We hence considered weaker models, namely finite state machines with distribution of actions on processes [72]. Even in this

context, the channel models to consider are complex, and covert channels capacity can only be approximated.

One lesson learned from our work on security, and more particularly on covert channels is that causality is an important property for security aspects. We are also convinced that information theory is the right tool to characterize the severity of leaks (and not only in scenario models), but that one can only expect approximations of leaks capacities for realistic models.

1.2 Robustness, time and realism of models

Another line of research have addressed robustness issues for models. Question frequently asked when a new model is proposed are: "how realistic is your model ? ", "do you have real case studies or industrial applications?". Very often, in asynchronous and distributed systems, models that are expressive enough to implement are undecidable, and models for which some decision procedures exist are not expressive enough to program a real system. We are convinced that models need not be real applications, they can be abstractions of real systems, or collect the incomplete knowledge of some experts, requirements, etc. However, a reasonable question to ask is whether a model reflects a plausible (sub)set of behaviors of a real system. This was for instance one of the reasons to choose as prefix-closed semantics for HMSCS and CFSMs in our work on synthesis (presented in chapter 7).

There are several ways to ensure that a model is plausible. Realizability can be seen as an extreme case of plausibility for a model. Another simple sanity check is to verify that a specification describes at least one behavior (i.e. for a given model M, check that $\mathcal{L}(M) \neq \emptyset$). As we have seen for scenarios, this simple problem is not always decidable.

Recently, a particular attention has been paid to timed robustness. A typical example of non-robust timed models is specifications containing Zeno behaviors, that is in which an unbounded number of actions can occur in one time unit. Obviously, a specification that contains Zeno behaviors should be considered as ill-formed. Similarly, timed models often have an idealized visions of time: clocks progress at a single rate, never drift, measurement of time is exact, actions are fired instantaneously as soon as some guard becomes true, etc .... In a real distributed application, each machine has its own clock, clocks may have different rates. So, exact measurement of time and instantaneous decisions are not possible.

Timed robustness is tightly related to this discrepancy between the idealized vision of time and a more realistic interpretation of timed mechanisms (called imperfect time assumption). Since the seminal work by Puri [123], a lot of attention has been devoted to analysis of timed automata under imperfect time assumption. Roughly speaking, timed robustness issues compare the semantics of a timed model with idealized interpretation of time, and with imperfect time. Robustness of a model can be defined in several ways: One may require that the set of reachable states of a model under perfect and imperfect time assumptions are the same. One can also require to preserve the untimed language of the model of some safety properties under imperfect time assumption, etc. In several works, we have considered extensions of true concurrency models with time and their semantics under realistic time assumption and under architectural constraints.

In [4,5], we have considered time-constrained MCS. This model decorates HMSCs with time constraints, that can state that the difference between the occurrence dates of two events lays within a given interval. Adding time constraints to HMSCs changes the semantics of the model, and some accepting paths generate MSCs that are inconsistent with respect to the timing constraints defined in the specification. In general, consistency of time-constrained MSC specifications (i.e. the question of whether the specification generates at least one consistent MSC) is undecidable. As very often with scenarios, there is a sub-class of regular TC-MSCs for which consistency is decidable [9]. In [5], we have shown that with the simple and verifiable assumption that no MSC appearing in a HMSC can be forced to take more that K time units to complete, consistency is decidable. Coupled with the assumption that the system can not be forced to execute more that Z events in one time unit, runs of a time-constrained HMSC can be represented by a regular set of representative timed behaviors [4]. We think that these works open the way for more elaborated verification techniques on time constrained MSCs..

We also have considered robustness issues for timed Petri nets (TPN) [105], and their semantics under guard enlargement [7]. We have identified several subclasses of TPNs for which robustness issues are decidable. So far, all decidable results on state robustness or language robustness rely on existence of a bound on markings of the considered net. When such bound exists (which is undecidable for TPNs in general), then the considered net can be translated into an equivalent timed automaton for which several decidability results were already proved [30]. Another study with time Petri nets considers constraints due to distribution and architectures, and resources limitations [6]. These constraints are modeled by a TPN which can prevent an action from being fired (due to time or resources constraints). This allows for instance to model time-sharing architectures. Given a specification net and architectural constraints, the question is then if the behavior of the specification is preserved under the given architectural constraints. Untimed language preservation is shown to be PSPACE complete, and timed language inclusion is also decidable (and strongly believed to be PSPACE complete too).

1.3 Specification and verification of Web Services

The last aspect of our research that does not appear explicitly in this document concerns Web Services. The main challenges to address in Web Services are composition, contract elaboration, and of course, verification of formal properties of a system. Since 2006, we have conducted a line of research on a declarative model for applications distributed over the web. This model, originally proposed by Serge Abiteboul [1, 2] is called *active documents*, and consists of guarded rules that either rewrite locally structured documents, or call services from other sites. In its full generality, this model is very powerful, but tuning the power of guards or rules, one can verify some simple reachability properties, or ensure that two modules compose well [99].

Session management is another important notion to address in Web Services. In a transactional system (web store), a client can interact with a commercial site using the Http protocol. However, several clients can access the site concurrently, and a client can access several sites at the same time. So, distinct transactions have

to be uniquely identified in a system. Http was not originally designed to handle this notion of transaction, and nowadays, many sites work by storing cookies on clients personal computers to remember a unique identifier for a transaction. The drawback of such unique identifier for sessions is that all models incorporating session identifiers management mechanisms need more or less to store counters, which rapidly leads to undecidability. Languages such as BPEL address the notion of session through correlations, that are filters to redirect a message to a particular process. However, BPEL is now a programming language, and is already too powerful to allow for automated reasoning. We have proposed an intermediate model [41] called *session systems*, which central paradigm is sessions, and that allows to design transactional systems over a finite set of agents, running an arbitrary number of sessions. The main idea is that agents accept to play a role within a session, that becomes private to all participants that have joined it. Sessions are described as regular models, and can be for instance finite automata of acyclic HMSC. In a configuration of a session system, an arbitrary number of sessions can coexist. Session systems can be implemented using shared memories, and furthermore coverability of some configuration is a decidable property of the model. Though coverability may seem a weak property, it is sufficient to model conflicts of interest, and several other interesting properties in a Web Services world.

2 Lessons learned

Before drawing some sketches for future work, let us summarize the lessons learned from 10 years of study of partial order automata. First of all, most of the models studied in this thesis (HMSCs, CHMSCs causal HMSCs, dynamic HMSCs) share common features beyond the fact that they compose partial orders. They all comport decidable syntactic subclasses of regular and globally-cooperative models, for which interesting properties are decidable. Let us also note that scenario models outside the globally cooperative classes are very often specifications that can be considered as ill-formed or too abstract.

We have seen that diagnosis is decidable for HMSCs and dynamic MSCs, but not for CHMSCs in general. We conjecture that diagnosis is decidable for all scenario models that compose communication closed patterns, and for which the projection on each process exhibits some regularity. We have show that scenarios can be practically applied, for diagnosis, but also for security issues. Indeed, the explicit representation of causality in scenarios seems to ideally characterize intentional information leaks, where notions such as interference can not distinguish between causal relationship between events and coincidence.

Overall, scenarios seem well adapted to applications where only a partial knowledge of the system is required. The more satisfactory results are those on diagnosis, which exploit the compactness of non-interleaved representation.

Should we extend existing scenario models ?

Before answering this question, let us summarize all the features that appear in the already considered scenario models: scenarios use asynchronous FIFO communications (by default), can allow sliding windows modeling (with CHMSC ans causal

MSCs), thread creation (with Dynamic MSCs,and MSC grammars). All these models have been studied with respect to theoretical concerns: expressiveness, verification,.... We must also consider them with respect to their potential interest as a modeling tool for designers of real-life systems. Many of the scenario models listed above reach the limits of human comprehension: a causal HMSC remains intuitive if commutations are limited to a single MSC, or to its immediate neighbors. When commutations allow for the definition of non-regular local process behaviors, we think that most designers would simply not understand the meaning of their specification. Similar concerns hold for dynamic MSCs. Hence, one still may imagine fancy ways of composing partial orders, but extensions of composition have a limit in terms of human understanding and practical use.

In our opinion, reasonable and needed extension of HMSCs and their variant should deal with time and probabilities. Indeed, many problems that are considered as solved in the untimed setting raise new issues in the timed case. Let us consider for instance time constrained HMSCs. This variant of HMSCs defines timed executions, that associates dates to events in the generated scenarios. These dates must satisfy some constraints (time intervals between events). In this simple and intuitive model, globally cooperative HMSC may not have a regular set of representant of timed behaviors. The reason is that time constraint disallow some executions that would have been legal in the timed case. The second direction for further studies is probabilities. The study of probabilities in HMSCs is still in its infancy, but is really needed. Among the expected information that one can expect from a systems's designer, one expects him to be able do describe which scenario is more probable than the other. Similarly, in HMSCs, messages sent are considered as eventually received, which is not true in a real life situation: messages get lost with some probability. Last, we have seen in chapter 9 that the size of a diagnosis grows with that of the observation. A good way to deal with this growth is to restrict diagnosis to the most probable explanations.

How to design large scenario models ?

The main idea behind all the works of chapter 5 was to design a complex model as a composition of simple and small HMSCs. However, scenarios do not compose well. First of all, none of the studied operations preserve the desirable properties of HMSCs and their extensions (global cooperation, locality, ...), accepts if the considered scenarios are restricted to regular HMSCs. This is clearly not sufficient. Then, one can not decide in general if two views of a system described by two simple scenario specifications comport at least one consistent run. This is clearly a drawback that hinders all compositional approaches to scenario modeling.

As we know that no human being will be able to design large models, and that composition of HMSC has many drawbacks, another research direction is *model learning*. From a set of collected traces of a running system, it seems interesting to build scenario models that embed all collected behaviors. Of course, the learned model would certainly be very complex. However, it is not necessarily meant to be read by human users, and could be used as a tool for diagnosis or security analysis.

Should we implement scenario models ?

In chapter 7, we have shown a way to implement any local choice HMSC, and this topic has been widely studied in the past. However, one has to remember that HMSCs are an abstraction of a distributed system's behavior. Implementing an HMSC means providing a code skeleton, that will be filled afterwards, introducing new features and wasting the verification performed on the original specification. Admittedly, HMSCs represent the initial stage of a more complex system's design. However, behaviors described in the HMSC should be executed by the model, up to some abstraction. This means in particular that diagnosis performed with an abstract model on a real running application derived from it should provide useful hints to debug code. Furthermore, if model and implementation have derived too far away from one another, the failure of the existence problem is a good indication for it. So, we believe that automatic generation of code from scenarios can be very useful is this techniques starts from specifications that are precise enough specifications.

Specifying with logics and orders ?

Chapter 8 of this document has shown that satisfiability of a set of formulae described using a partial order logic is rapidly undecidable, except with strong restrictions on the shape of models for the formula. Such restriction usually bringing back the decision procedure to regular models that either represent the whole set of possible models of the formula (for instance the K-influencing restriction for LPOC allows for the specification of a regular language of computations) or a set of representative runs (setting an existential bound for MSCs reduces the search fr a model to a regular set of representative linearizations).

However, without restriction, emptiness of the designed MSC language is undecidable for logic-based scenario specifications. As for compositional specification with scenarios, undecidability of emptiness tolls the bell for a fully general use of such declarative formalisms for scenarios. Considering restrictions to the models may help providing solutions. For instance, requiring that the models of a formula are existentially bounded MSCs, means is some sense restricting to specifications of systems that implemented in terms of existentially bounded CFSMs. Such limitation does not look too severe.

3 Future work

To end this document, we would like to draw some future research directions. The general motivation for our future work is to stay connected to real-life problems, and provide useful formal tools to reason on models of distributed systems. Indeed, we can continue increasing the expressive power of scenario while exhibiting features that are not needed, nor represent any real situation in a distributed system. Similarly, event when decidable classes of a scenario variant are found, there are decision procedures with such a high complexity that they can not be used in practice. A good test for practicality of a model is whether the model can be implemented on a network of machines. We do not feel that a model should always be implementable: this often means that no automated verification technique applies, and we also want

to use verification tools. However, we feel that models should always represent some real running system up to abstraction or incompleteness.

Of course, this remark does not apply specifically to scenarios, and we plan to consider other models, still with this realisms requirement in mind. Current distributed systems include huge networks, web-service platforms, but also organizations workflows. Partial order automata were originally created and used in the area of protocols modeling, but we think that they have reached some maturity, and also some limits in this domain. Changing the application domain for partial order automata also means changing needs, semantics rules, possible restrictions, etc to the model.

3.1 Scenario-based formalisms with practical restrictions and semantics variants

While we have considered variants of scenarios with many features (time, dynamic process creation, etc.), a few work has been devoted to searching sound restrictions to the behaviors depicted by a partial order model. A first step in this direction is of course the assumption of an existential bound for runs of a model. Used for verification purposes, this property ensures that one can represent an infinite set of behaviors by a set (not necessarily finite nor regular) of bounded representative linearizations. Existential bound also guarantees that an implementation can not be forced to store an arbitrary number of messages in its communication buffers, which seems a safe restriction. Furthermore, it allows for decidability of more properties. Considering existentially bounded MSCs as models of MSO^{MSC} formulae allows for decidability of satisfiability.

In a timed context, we have proposed close restrictions saying that a basic scenario can not be forced to take an arbitrarily long time to be completely executed. This looks realistic and plausible if one considers a basic scenario as some phase of a protocol. Hence, loosing a little generality is scenarios semantics, one can gain decidability. Hence, it seems interesting to continue considering this tradeof between expressiveness and decidability. However, one should at the same time ensure that the proposed restrictions can be accepted in a targeted applications domain. For instance, requiring HMSCs to be regular in the context of asynchronous protocols modeling is too restrictive. The subclasses of HMSCs are now well known, so we think that very few sensible subclasses can be proposed. However, in the context of Time Constrained MSCs, verification is still in its infancy, and a lot of work remains to be done.

Another way to relate scenario models to real world situations is to adapt their semantics. Indeed, in a Web Services context, for instance, noting guarantees that communications between machines are FIFO anymore. Processes can hence be seen as communicating with bags rather than with queues. Slightly changing the semantics of scenarios to take into account the specificities of an application domain may result in changes in the decidability and complexity of many problems.

3.2 Robustness

Work on robustness also answers the realism demand. Indeed, even if a model is considered as a partial, abstract, and incomplete specification of the behaviors of a system, as soon as some architectural information is available, one should check that the requirements are consistent with the architecture. Following our former results on Time Petri Nets, we would like to consider timed robustness for other models. As the preliminary results show, robustness issues are rapidly undecidable for models with infinite numbers of configurations. However, robustness can be seen as some kind of sanity check, ran before starting any implementation. Such check can be performed up to a sound abstraction. Furthermore, one can also rely on semi-decision algorithms to answer a robustness question, as demonstrated in [7].

Interesting lines of research are time robustness for Time Constrained scenarios. Indeed, robustness checking on scenarios can be very useful to test design hypotheses at very early stages of a design. So, the question to address is what happens to scenarios if one considers that time measurement by processes is imperfect, that some processes share physical machines via time-sharing mechanisms, that each process has it own independent clock, etc ? Indeed, using even incomplete information on a system's resources and architectural constraints may rapidly show that some requirements can not be implemented, or that the specification becomes inconsistent.

Leaving the scenario world, robustness issues should also be considered for variants of Petri nets that allow to associate time to tokens (in Time Petri nets, time is attached to transition, and measures time since enabling). Of course, most issues should be undecidable, but we believe that this feature is essential to model transactions in distributed systems, where one can not afford to forget time progress when disabling a transition. We should hence consider tools to check robustness for such models, probably at the cost of some abstraction or approximation.

3.3 Web services

Web services is an interesting domain to consider for formal modeling of distributed systems, for several reasons. First, they are distributed systems, and hence interesting objects to consider. But of course, this is not sufficient to build a research project on this topic. Another remark is that contrarily to low-level protocols based on TCP-IP, for instance, web-services architectures seldomly assume that communications are FIFO. Indeed, clients requests are not necessarily served in their order of arrival, and communication overs the World Wide Web do not necessarily guarantee FIFO links between two sites. This has one important consequence: the FIFO assumption, which is often source for undecidability, is dropped. Hence models for Web-services are close to Petri nets variants, and might be somehow more amenable to verification.

Note however that even without FIFO communications, Web-services comport difficult points to address. Web-services own and share data, and very often, the dynamics of a system depends on this data. We note that data was not seriously addressed within the MSC community, but also that introducing data is often at the cost of new undecidable results. In the future, we would like to consider Web services, seen as *communicating rewriting rules for distributed documents*. Every specification is implementable (it is simply a set of local rules applied locally to data,

generating communications). So, implementability is guaranteed, and the interesting questions are rather **compatibility** between services (does a service fulfills the needs of a infrastructure for some functionality), and **Quality of Service** (does a service returns an answer of the expected quality, for instance within the required delay and with a value ranging in a predetermined domain?).

Compatibility of services can be presented as follows: one wants to use a web service W that provides some function in an environment E that expects a service to be provided. The questions that arise are: does W accepts all inputs that may be sent by E ? Does all answers returned by W are expected by E ? Does W terminate for any input provided by W ? The first two questions are of purely static nature, and can be seen as some query inclusion problem (see for instance [63, 106, 116, 117]), for which solutions have been provided for a long time, even for complex data description. The last question is more a termination problem. Formal verification of web-based systems is a close issue. We are convinced that complex logics (CTL, μ-calculus, etc) are not suited to the size of web-based systems. However, coverability of some undesired configuration was shown decidable for some Web Services models (see [41]), and this question is already an interesting feedback for systems designer. Techniques inherited from coverability decision techniques can be reused to prove more elaborated properties describing successive configuration in a run of the system.

Termination is an issue, but one also has to consider QoS, that is in particular if a service replies in a reasonable amount of time, if the data returned satisfies some quality criteria, etc. Such questions can not be addressed in a purely qualitative manner. If a service is answered with a decent delay for a high percentage of the calls, then it can be considered as correct, even if if some exceptional cases it may not answer. Hence, we need to consider assembling services if they agree on the exchanged data, but also on a the executed QoS.

The last interesting paradigm is that Web-based systems are **open systems**. Indeed, most of transactional systems accept request form external users, for which the arriving data, kind, and rate of requests can not be controlled. Furthermore, a web services architecture is build by assembling services, and among them, some services might be provided by external stakeholders. In such situation, only an interface to the services (input data accepted, output data returned, ...) is provided, and the specification of the whole systems is not available. Nevertheless, it is possible to reason formally on an open system with holes, by assuming properties of the services that will be called. Hence, rather that trying to solve question of the form "does service f terminates?" that are likely to be undecidable, we would like to consider proof techniques that accept and generate hypotheses. Hence, the questions to ask would be of the form "does f terminates under the assumption that g always returns integers smaller than 12?". Answers could be conditional results, i.e. contain new proof obligations "'yes, f terminates iff g terminates". The condition to satisfy can be a synthesized proof obligation, or more simply a new fact entered by an user to allow progress of the proof system. We think that such approach have huge interest if one can mix qualitative and quantitative aspects of the form "the distribution of response times for f is below distribution $F()$ iff the distribution of response times for g is below $G()$.

In an open world, specification of interfaces for services can be seen as contract. Contract allow to reason in a compositional way, assuming that a service provider

by an external stakeholder will be fulfilled. Once contracts have been established, the remaining question is how to ensure that they are fulfilled ? The answer is to monitor web-based systems, but ideally, monitors for contracts should be generated automatically.

Overall, the world of Web services seems an interesting area to provide useful applications for formal models and verification techniques, in a real distributed and asynchronous context. Preliminary results and models [41, 99] for Web services show that even in a difficult context mixing openness, data, infinite configuration space, etc. some solutions can be found to coverability or composition problems. The models that we plan to consider are mainly declarative rule based models to manipulate structured data. However, transactions on the Web can be intuitively depicted by simple scenarios. We think that variants of scenarios adapted to the Web Services world, as well as the experience gained during the last decade of research on scenarios might provide some useful help.

Bibliography

[1] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo et R. Weber : Active XML: A data-centric perspective on web services. *In BDA02*, 2002.

[2] S. Abiteboul, L. Segoufin et V. Vianu : Static analysis of Active XML systems. *In PODS08*, pages 221–230, 2008.

[3] M. Ahuja, A.D. Kshemkalyani et T. Carlson : A Basic Unit of Computation in Distributed Sytems. *In Proc. of ICDS'90*, pages 12–19, 1990.

[4] S. Akshay, Blaise Genest, Loïc Hélouët et Shaofa Yang : Regular set of representatives for time-constrained msc graphs. *Inf. Process. Lett.*, 112(14-15):592–598, 2012.

[5] S. Akshay, Blaise Genest, Loïc Hélouët et Shaofa Yang : Symbolically bounding the drift in time-constrained msc graphs. *In ICTAC*, volume 7521 de *Lecture Notes in Computer Science*, pages 1–15. Springer, 2012.

[6] S. Akshay, Loïc Hélouët, Claude Jard, Didier Lime et Olivier H. Roux : Robustness of time petri nets under architectural constraints. *In FORMATS*, volume 7595 de *Lecture Notes in Computer Science*, pages 11–26. Springer, 2012.

[7] S. Akshay, Loïc Hélouët, Claude Jard et Pierre-Alain Reynier : Robustness of time petri nets under guard enlargement. *In RP*, volume 7550 de *Lecture Notes in Computer Science*, pages 92–106. Springer, 2012.

[8] S. Akshay, M. Mukund et N.K. Kumar : Checking coverage for infinite collections of timed scenarios. *In CONCUR'07*, pages 181–196, 2007.

[9] S. Akshay, Madhavan Mukund et K. Narayan Kumar : Checking coverage for infinite collections of timed scenarios. *In CONCUR*, volume 4703 de *Lecture Notes in Computer Science*, pages 181–196. Springer, 2007.

[10] R. Alur, K. Etessami et M. Yannakakis : Inference of message sequence charts. *In ICSE*, pages 304–313, 2000.

[11] R. Alur, K. Etessami et M. Yannakakis : Realizability and verification of msc graphs. *In ICALP*, pages 797–808, 2001.

[12] R. Alur, K. Etessami et M. Yannakakis : Realizability and verification of MSC graphs. *Theoretical Computer Science*, 331(1):97–114, 2005.

[13] R. Alur, G. Holzmann et D. Peled : An analyser for mesage sequence charts. *In* Tiziana Margaria et Bernhard Steffen, éditeurs : *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 de *Lecture Notes in Computer Science*, pages 35–48. Springer-Verlag, 1996.

[14] R. Alur et M. Yannakakis : Model checking of Message sequence Charts. *In Proc. of CONCUR'99*, numéro 1664 de LNCS, pages 114–129. Springer, 1999.

[15] Rajeev Alur, Gerard J. Holzmann et Doron Peled : An analyzer for message sequence charts. *Software - Concepts and Tools*, 17(2):70–77, 1996.

[16] André Arnold : *Finite transition systems.* Prentice-Hall. Prentice-Hall, 1994.

[17] S. Axelson : Intrusion detection systems: A taxomomy and surveytechnical report. Rapport technique 99-15, Dept. of Computer Engineering, Chalmers University of Technology, Sweden, 2000.

[18] P. Baker, P. Bristow, C. Jervis, D.J King et B. Mitchell : Automatic generation of conformance tests from message sequence charts. *In SAM*, pages 170–198, 2002.

[19] N. Baudru et R. Morin : Synthesis of safe message-passing systems. *In FSTTCS*, pages 277–289, 2007.

[20] Marek A. Bednarczyk, Luca Bernardinello, Benoît Caillaud, Wieslaw Pawlowski et Lucia Pomello : Modular system development with pullbacks. *In ICATPN*, volume 2679 de *Lecture Notes in Computer Science*, pages 140–160, 2003.

[21] D.E Bell et J.J La Padula : Secure computer systems: a mathematical model. MITRE technical report 2547, MITRE, may 1973. Vol II.

[22] D.E Bell et J.J La Padula : Secure computer systems: mathematical foundations. Mitre technical report 2547, MITRE, march 1973. Vol I.

[23] H. Ben-Abdallah et S. Leue : Syntactic detection of process divergence and non-local choice in message sequence charts. *In* E. Brinksma, éditeur : *Proceedings of the Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems TACAS'97*, volume 1217 de *Lecture Notes in Computer Science*, pages 259 – 274, Enschede,The Netherlands, April 1997. Springer-Verlag.

[24] A. Benveniste, E. Fabre, C. Jard et S. Haar : Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, May 2003.

[25] B. Bollig : On the Expressiveness of Asynchronous Cellular Automata. *In FCT*, pages 528–539, 2005.

[26] Benedikt Bollig et Loïc Hélouët : Realizability of dynamic msc languages. *In CSR*, pages 48–59, 2010.

[27] Benedikt Bollig, Dietrich Kuske et Ingmar Meinecke : Propositional dynamic logic for message-passing systems. *In FSTTCS*, volume 4855 de *Lecture Notes in Computer Science*, pages 303–315, 2007.

[28] Benedikt Bollig, Dietrich Kuske et Ingmar Meinecke : Propositional dynamic logic for message-passing systems. *Logical Methods in Computer Science*, 6(3), 2010.

[29] Benedikt Bollig, Martin Leucker et Philipp Lucas : Extending compositional message sequence graphs. *In LPAR*, volume 2514 de *Lecture Notes in Computer Science*, pages 68–85. Springer, 2002.

[30] Patricia Bouyer, Nicolas Markey et Ocan Sankur : Robust model-checking of timed automata via pumping in channel machines. *In Proc. of FORMATS'11*, volume 6919 de *LNCS*, pages 97–112.

[31] D. Brand et P. Zafiropulo : On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, Avril 1983.

[32] Daniel Brand et Pitro Zafiropoulo : On communicating finite state machines. Rapport technique 1053, IBM Zurich Research Lab., 1981.

[33] Benoit Caillaud, Philippe Darondeau, Loïc Hélouët et Gilles Lesventes : HMSCs en tant que spécifications partielles et leurs complétions dans les réseaux de Petri. Research Report RR-3970, INRIA, 2000.

[34] L. Castellano, G. de Michelis et Pomello,L. : Concurrency versus interleaving: An instructive example. *BEATCS: Bulletin of the European Association for Theoretical Computer Science*, 31, 1987.

[35] E. Clarke, O. Grumberg, S. Jha, Y. Lu et H. Veith : ounterexample-guided abstraction refinement. *In Proceedings of CAV'2000*, pages 154–169. LNCS 1855, 2000.

[36] E.M. Clarke et E.A. Emerson : Desing and synthesis of synchronisation skeletons using branching time temporal logic. *In Logic of Programs*, volume 131 de *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[37] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison et Marc Tommasi : Tree automata techniques and applications. Rapport technique, 2008.

[38] Werner Damm et David Harel : Lscs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[39] Philippe Darondeau, Blaise Genest et Loïc Hélouët : Products of message sequence charts. *In FoSSaCS*, pages 458–473, 2008.

[40] Philippe Darondeau, Blaise Genest et Loïc Hélouët : Products of message sequence charts. Research Report 6258, INRIA, 2008.

[41] Philippe Darondeau, Loïc Hélouët et Madhavan Mukund : Assembling sessions. *In ATVA*, volume 6996 de *Lecture Notes in Computer Science*, pages 259–274. Springer, 2011.

[42] DARPA : Intrusion detection dataset. http://www.ll.mit.edu/IST/ideval/data/data_index.html, 2000.

[43] Volker Diekert et Gregor Rozenberg, éditeurs. *The Book of Traces*. World Scientific, 1995.

[44] Christine Duboc : Mixed product and asynchronous automata. *Theor. Comput. Sci.*, 48(3):183–199, 1986.

[45] Catherine Dufourd, Alain Finkel et Ph. Schnoebelen : Reset nets between decidability and undecidability. *In ICALP*, volume 1443 de *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998.

[46] A. Engels : Message refinement: Describing multi-level protocols in MSC. *In* Y. Lahav, A. Wolisz, J. Fischer et E. Holz, éditeurs : *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC*, numéro 104 de Informatik-Berichte, pages 67–74, Berlin, Germany, juin 1998. Humboldt-Universität zu Berlin.

[47] C. Fidge : Logical time in distributed computing systems. *Computer*, 24(8): 28–33, 1991.

[48] C. Fidge : Logical time in distributed computing systems. *IEEE Computer*, 24(8):28–33, August 1991.

[49] Riccardo Focardi et Roberto Gorrieri : Classification of security properties (part i: Information flow). *In Proceedings of FOSAD 2000*, pages 331–396, 2000.

[50] Paul Gastin et Dietrich Kuske : Satisfiability and model checking for mso-definable temporal logics are in pspace. *In CONCUR*, volume 2761 de *Lecture Notes in Computer Science*, pages 218–232. Springer, 2003.

[51] Thomas Gazagnaire : *Langages de Scénarios: Utiliser des Ordres Partiels pour Modéliser, Vérifier et Superviser des Systèmes Parallèles et Répartis*. Thèse de doctorat, Université de Rennes 1, 2008.

[52] Thomas Gazagnaire, Blaise Genest, Loïc Hélouët et Hervé Marchand : Diagnosis from scenarios, and applications. Research Report RR-xxx, INRIA, 2000.

[53] Thomas Gazagnaire, Blaise Genest, Loïc Hélouët, P. S. Thiagarajan et Shaofa Yang : Causal message sequence charts. *In CONCUR'07*, volume 4703 de *Lecture Notes in Computer Science*, pages 166–180, 2007.

[54] Thomas Gazagnaire, Blaise Genest, Loïc Hélouët, P. S. Thiagarajan et Shaofa Yang : Causal message sequence charts. *Theor. Comput. Sci.*, 410(41):4094–4110, 2009.

[55] B. Genest, D. Kuske et A. Muscholl : A Kleene Theorem and Model Checking for a Class of Communicating Automata. *Information and Computation.*, 204(6):920–956, 2006.

[56] B. Genest, A. Muscholl, H. Seidl et M. Zeitoun : Infinite-state high-level mscs: Model-checking and realizability. *In ICALP*, volume 2380 de *LNCS*, pages 657–668, 2002.

[57] B. Genest, A. Muscholl, H. Seidl et M. Zeitoun : Infinite-state high-level MSCs: Model-checking and realizability. *Journal on Comp. and System Sciences*, 72(4):617–647, 2006.

[58] Blaise Genest : *L"Odyssée des Graphes de Diagrammes de Séquences (MSC-Graphes).* Thèse de doctorat, Paris 7, 2004.

[59] Blaise Genest, Loïc Hélouët et Anca Muscholl : High-level message sequence charts and projections. *In CONCUR*, volume 2761 de *Lecture Notes in Computer Science*, pages 308–322, 2003.

[60] Blaise Genest, Marius Minea, Anca Muscholl et Doron Peled : Specifying and verifying partial order properties using template mscs. *In FoSSaCS*, volume 2987 de *Lecture Notes in Computer Science*, pages 195–210, 2004.

[61] Blaise Genest, Anca Muscholl et Dietrich Kuske : A kleene theorem for a class of communicating automata with effective algorithms. *In Developments in Language Theory*, volume 3340 de *Lecture Notes in Computer Science*, pages 30–48. Springer, 2004.

[62] J. A. Goguen et J. Meseguer : Security policies and security models. *In 1982 Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.

[63] Georg Gottlob, Christoph Koch et Reinhard Pichler : Efficient algorithms for processing xpath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.

[64] S. Govindaraju et D. Dill : Counterexample-guided choice of projections in approximate symbolic model checking. *In ICCAD*, 2000.

[65] Elsa L. Gunter, Anca Muscholl et Doron Peled : Compositional message sequence charts. *In TACAS*, volume 2031 de *Lecture Notes in Computer Science*, pages 496–511, 2001.

[66] L. Hélouët, R. Abdallah et D. Bhatia : SOFAT : Scenario formal analysis toolbox. 2011.

[67] L. Hélouët, M. Zeitoun et A. Degorre : Scenarios and covert channels, another game... *In Proc. Games in Design and Verification, GDV '04, Satellite of Computer Aided Verification, CAV'04*, Electronic Notes in Theoretical Computer Science. Elsevier, 2004. To appear.

[68] Loïc HÉLOUËT : Distributed system requirement modeling with message sequence charts: the case of the rmtp2 protocol. *Information & Software Technology*, 45(11):701–714, 2003.

[69] Loïc HÉLOUËT, Thibaut HÉNIN et Christophe CHEVRIER : Automating scenario merging. *In SAM*, volume 4320 de *Lecture Notes in Computer Science*, pages 64–81, 2006.

[70] Loïc HÉLOUËT, Claude JARD et Benoît CAILLAUD : An event structure based semantics for high-level message sequence charts. *Mathematical Structures in Computer Science*, 12(4):377–402, 2002.

[71] Loïc HÉLOUËT et Pierre Le MAIGAT : Decomposition of message sequence charts. *In SAM*, pages 47–60, 2000.

[72] Loïc HÉLOUËT et Aline ROUMY : Covert channel detection using information theory. *In SecCo*, volume 51 de *EPTCS*, pages 34–51, 2010.

[73] Loïc HÉLOUËT et Aline ROUMY : On the differences between covert channels and interference. *In GIPSY'2010 : 1st Workshop on Games, Logic and Security*, 2010.

[74] Jesper G. HENRIKSEN, Madhavan MUKUND, K. Narayan KUMAR, Milind A. SOHONI et P. S. THIAGARAJAN : A theory of regular msc languages. *Inf. Comput.*, 202(1):1–38, 2005.

[75] Jesper G. HENRIKSEN et P. S. THIAGARAJAN : Dynamic linear time temporal logic. *Ann. Pure Appl. Logic*, 96(1-3):187–207, 1999.

[76] J.G. HENRIKSEN, M. MUKUND, K. NARAYAN KUMAR, M. SOHONI et P.S. THIAGARAJAN : A Theory of Regular MSC Languages. *Information and Computation*, 202(1):1–38, 2005.

[77] L. HÉLOUËT, T. GAZAGNAIRE et B. GENEST : Diagnosis from scenarios. *In Workshop on Discrete Event Systems, WODES'06*, 2006.

[78] L. HÉLOUËT, M. ZEITOUN et C. JARD : Covert channels detection in protocols using scenarios. *In Proc. of SPV'03 Security Protocols Verification*, pages 21–25, sep. 2003.

[79] Loïc HÉLOUËT et Claude JARD : Conditions for synthesis of communicating automata from hmscs. *In 5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, 2000.

[80] J.E. HOPCROFT et J.D. ULLMAN : *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[81] ITU-T : Z.120 : Message sequence charts (MSC). Rapport technique, International Telecommunication Union, 2011.

[82] ITU-TS : *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, November 1999.

[83] ITU-TS : *ITU-TS Recommendation Z.100: Specification and Description Language (SDL)*. ITU-TS, Geneva, November 2007.

[84] Donald.B Johnson : Finding all the elementary circuits of a directed graph. *SIAM Journal of Computing*, 4(1):77–84, 1975.

[85] A.K. Jones et R.S. Sielken : Computer system intrusion detection: A survey. Rapport technique, Dept. of Computer Science, University of Virginia, 1999.

[86] P. Kabiri et A.A. Ghorbani : Research on intrusion detection and response: A survey. *International Journal of Network Security*, 1(2):84–102, 2005.

[87] R.A. Kemmerer : Shared ressources matrix methodology: an approach to indentifying storage and timing channels. *ACM transactions on Computer systems*, 1(3):256–277, 1983.

[88] Jacques Klein, Benoît Caillaud et Loïc Hélouët : Merging scenarios. *Electr. Notes Theor. Comput. Sci.*, 133:193–215, 2005.

[89] D.J Kleitman et B.L Rotschild : Asymptotic enumeration of partial orders on a finite set. *Transactions of the American Mathematical Society*, (205):205–220, 1975.

[90] C. Ko, M. Ruschitzka et K.N. Levitt : Execution monitoring of security-critical programs in distributed systems: A specification-based approach. *In IEEE Symposium on Security and Privacy*, pages 175–187, 1997.

[91] D. Kuske : Regular sets of infinite message sequence charts. *Information and Computation*, 187(1):80–109, 2003.

[92] Martin Leucker, P. Madhusudan et Supratik Mukhopadhyay : Dynamic message sequence charts. *In* M. Agrawal et A. Seth, éditeurs : *Proceedings of 22nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'02)*, volume 2556 de *Lecture Notes in Computer Science*. Springer, décembre 2002.

[93] K. Lodaya et P. Weil : Series-parallel languages and the bounded-width property. *Theoretical Computer Science*, 237(2):347–380, 2000.

[94] M. Lohrey : Realizability of high-level message sequence charts: closing the gaps. *Theoretical Computer Science*, 309(1-3):529–554, 2003.

[95] Markus Lohrey : Safe realizability of high-level message sequence charts. *In CONCUR*, volume 2421 de *Lecture Notes in Computer Science*, pages 177–192. Springer, 2002.

[96] Markus Lohrey et Anca Muscholl : Bounded msc communication. *Inf. Comput.*, 189(2):160–181, 2004.

[97] P. Madhusudan : Reasoning about sequential and branching behaviours of Message Sequence Graphs. *In Proceedings of ICALP'01*, page 809. LNCS 2076, 2001.

[98] P. Madhusudan et B. Meenakshi : Beyond message sequence graphs. *In FSTTCS*, volume 2245 de *Lecture Notes in Computer Science*, pages 256–267, 2001.

[99] Benoît Masson, Loïc Hélouët et Albert Benveniste : Compatibility of data-centric web services. *In WS-FM*, volume 7176 de *Lecture Notes in Computer Science*, pages 32–47. Springer, 2011.

[100] F. Mattern : Time and global states of distributed systems. *in Proc. Int. Workshop on Parallel and Distributed Algorithms , Bonas, France , North Holland*, pages 215–226, 1988.

[101] F. Mattern : On the relativistic structure of logical time in distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

[102] S. Mauw, M. van Wijk et T. Winter : A formal semantics of synchronous interworkings. *In SDL'93 : Using Objects*, 1993.

[103] Sjouke Mauw et Michel A. Reniers : Refinement in interworkings. *In CONCUR*, volume 1119 de *Lecture Notes in Computer Science*, pages 671–686, 1996.

[104] B. Meenakshi et Ramaswamy Ramanujam : Reasoning about layered message passing systems. *Computer Languages, Systems & Structures*, 30(3-4):171–206, 2004.

[105] Philip M. Merlin : *A Study of the Recoverability of Computing Systems*. Thèse de doctorat, University of California, Irvine, CA, USA, 1974.

[106] Gerome Miklau et Dan Suciu : Containment and equivalence for a fragment of xpath. *J. ACM*, 51(1):2–45, 2004.

[107] J. Millen : Covert channel capacity. *In IEEE Symposium on Security and Privacy*, pages 60–66, 1987.

[108] T. Montgommery, B. Whetten, M. Basavaiah, S. Paul, N. Rastogi, J. Colan et T. Yeh : The RMTP2 protocol. Rapport technique, IETF, 1998.

[109] Rémi Morin : Recognizable sets of message sequence charts. *In STACS'02*, volume 2285 de *Lecture Notes in Computer Science*, pages 523–534, 2002.

[110] I. Moskowitz et M. Kang : Covert channels - here to stay ? *In Proceedings of COMPASS'94*, pages 235–243. IEE Press, 1994.

[111] M. Mukund, K.N. Kumar et M. Sohoni : Synthesizing distributed finite-state systems from mscs. *In CONCUR*, pages 521–535, 2000.

[112] Anca Muscholl : Matching specifications for message sequence charts. *In FoSSaCS*, pages 273–287, 1999.

[113] Anca Muscholl et Doron Peled : Message sequence graphs and decision problems on mazurkiewicz traces. *In MFCS*, pages 81–91, 1999.

[114] Anca Muscholl et Doron Peled : From finite state communication protocols to high-level message sequence charts. *In Proceedings of ICALP'2001*, volume 2076 de *Lecture Notes in Computer Science*, pages 720–731. Springer, 2001.

[115] Anca Muscholl, Doron Peled et Zhendong Su : Deciding properties for message sequence charts. *In FoSSaCS'98*, volume 1378 de *Lecture Notes in Computer Science*, pages 226–242. Springer, 1998.

[116] Frank Neven et Thomas Schwentick : Xpath containment in the presence of disjunction, dtds, and variables. *In ICDT*, pages 312–326, 2003.

[117] Frank Neven et Thomas Schwentick : On the complexity of xpath containment in the presence of disjunction, dtds, and variables. *CoRR*, abs/cs/0606065, 2006.

[118] University of California : Kdd cup 1999 data, 1999.

[119] OMG : *UML 2.0 : Unified Modeling Language*. Object Management Group, August 2005.

[120] Robert Paige et Robert Tarjan : Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6), 1987.

[121] D. Peled : Specification and verification of Message Sequence Charts. *In Proceedings of FORTE'00*, pages 139–154. IFIP 183, 2000.

[122] A. Pnueli : Abstraction, composition, symmetry, and a little deduction: The remedies to state explosion. *In Proceedings of CAV'2000*. LNCS 1855, 2000.

[123] Anuj Puri : Dynamical properties of timed automata. *In DEDS*, 10(1-2):87–113, 2000.

[124] M. Raynal, A. Schiper et S. Toueg : The causal ordering abstraction and a simple way to implement it. *Inf. Process. Lett.*, 39(6):343–350, 1991.

[125] Vojtech Rehák, Petr Slovák, Jan Strejcek et Loïc Hélouët : Decidable race condition and open coregions in hmsc. *ECEASST*, 29, 2010.

[126] M. Reniers : *Message Sequence Chart: Syntax and Semantics*. Thèse de doctorat, Eindhoven University of Technology, 1999.

[127] M. Reniers et S. Mauw : High-level Message Sequence Charts. *In* A. Cavalli et A. Sarma, éditeurs : *SDL97: Time for Testing - SDL, MSC and Trends*, Proc. of the 8th SDL Forum, pages 291–306, Evry, France, Septembre 1997.

[128] E. Rudolph, P. Graubmann et J. Grabowski : Message Sequence Chart: composition techniques versus OO-techniques - 'tema con variazioni'. *In* R. Bræk et A. Sarma, éditeurs : *SDL'95 with MSC in CASE*, Proceedings of the Seventh SDL Forum, pages 77–88, Oslo, 1995. Amsterdam, North-Holland.

[129] A. Sabelfeld et A.C. Myers : Language-based information-flow security. *IEEE Journal on selected areas in communications*, 21(1), Jan. 2003.

[130] O. Salem, S. Vaton et A. Gravey : An efficient online anomalies detection mechanism for high-speed networks. *In MonAM'07*, 2007.

[131] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen et D.C Teneketzis : Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, 1996.

[132] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang et S. Zhou : Specification-based anomaly detection: A new approach for detecting network intrusions. *In 9th ACM conference on Computer and communications security*, 2002.

[133] R. Tarjan : Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2), 1972.

[134] S. Uchitel, J. Kramer et J. Magee : Detecting implied scenarios in Message Sequence Chart specifications. *In ESEC / SIGSOFT FSE*, pages 74–82, 2001.

[135] Sebastian Uchitel : *Incremental Elaboration of Scenario- Based Specifications and Behaviour Models Using Implied Scenarios.* Thèse de doctorat, Imperial College of Science, Technology and Medicine, University of London, Department of Computing, 2003.

[136] Sebastián Uchitel, Jeff Kramer et Jeff Magee : Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Trans. Softw. Eng. Methodol.*, 13(1):37–85, 2004.

[137] D. Volpano et G. Smith : Eliminating covert flows with minimum typings. *In Proc. 10th IEEE Computer Security Foundations Workshop*, pages 156–168, June 1997.

[138] Shaofa Yang, Loïc Hélouët et Thomas Gazagnaire : Logic based diagnosis for distributed systems. *Perspectives in Concurrency Theory, a Festchrift for P.S. Thiagarajan*, 2009, pages = 482-505,.

Index

Chapter 12

Appendix

This chapter contains the proofs for some of the theorems that appear through the document. All proofs are not included, in particular when a theorem is a direct consequence of statements appearing before in the document, when it was thoroughly demonstrated in a paper (in which case we mention this work).

1 Proofs for chapter 2

1.1 Undecidability for $\mathcal{F}_{H_1} \cap \mathcal{F}_{H_2} = \emptyset$?

Proof for (a part of) Theorem 3

A PCP can be formulated as follows. Let $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ be two sets of finite words over an alphabet Σ with at least two letters. The question asked in a PCP is whether there exists a non empty finite sequence of indexes $i_1, \ldots, i_k$ such that $u_{i_1}.u_{i_2}\ldots.u_{i_k} = v_{i_1}.v_{i_2}\ldots.v_{i_k}$.

This problem is undecidable. We can easily reduce it to HMSC languages/families intersection. We will design two HMSCs H_1, H_2 over a set of processes $\{p, q\}$. For a given word $u = a_1.\ldots.a_k \in \sigma^*$, let us call $q(u)$ the sequence of atomic actions $q(a_1).\ldots.q(a_k)$. We design two sets of MSCS $\mathcal{M}_1 = \{U_1, U_n\}$ and $\mathcal{M}_2 = \{V_1, \ldots V_n\}$ such that each U_i (respectively V_i) contains a single atomic action $p(ind_i)$ on process p and the sequence $q(u_i)$ of atomic actions on process q. We then design $H1$ and H_2 as follows: H_1 comports three nodes n_0, n_1, with n_1 final, transitions from n_0 to n_1 labeled by each U_i, and transition from n_1 to itself labeled by each U_i. The HMSC H_2 comports one node n_0 which is final, and transition from n_0 to itself labeled by each V_i. Such construction is illustrated in Figure 12.1 for instance $u_1 = aab, u_2 = bb, u_3 = baa$ $v_1 = aa, v_2 = abb, v_3 = bba$ of the PCP. It can be seen immediately that an instance of the PCP has a solution if and only if there exists a MSC M in $\mathcal{F}_{H_1} \cap \mathcal{F}_{H_2}$. In M one can chose a sequence of letters on process p that corresponds to indexes of $u_i's$, $v_i's$. Each letter chosen on p yields a unique transition in H_1 or H_2. Furthermore, as $M \in \mathcal{F}_{H_1} \cap \mathcal{F}_{H_2}$, H_1 and H_2 agree on the unique sequence of letters that have to appear on process q in M. Hence $M \in \mathcal{F}_1 \cap \mathcal{F}_2$, and $\mathcal{F}_1 \cap \mathcal{F}_2 \neq \emptyset$. Note that H_1 is not only composed of loops, so the empty MSC is not recognized by H_1, and can not be a solution of the intersection problem.

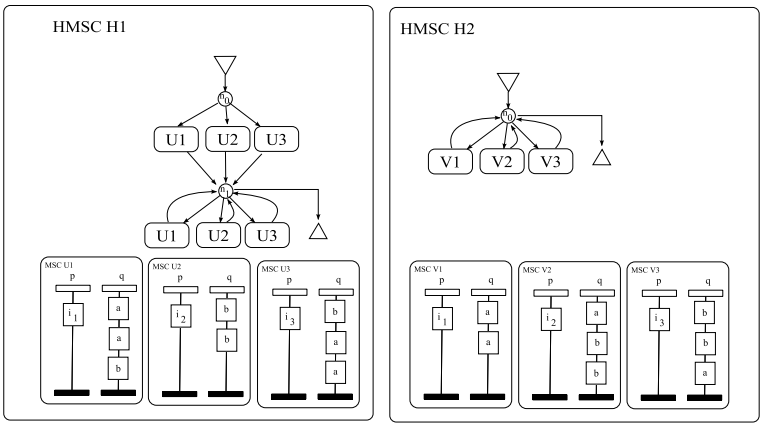


Figure 12.1: encoding a PCP with HMSCs intersection

1.2 Divergence is a co-NP Complete problem (theorem 7)

This Co-NP completeness proof for divergence is a rather standard proof that appears with some variant for classification of a HMSC in a syntactical class. We hence give details for this proof only, and will only give the small changes in the proof for similar problems.

Theorem 7 *Checking that a HMSC is not divergent is a co-NP complete problem.*

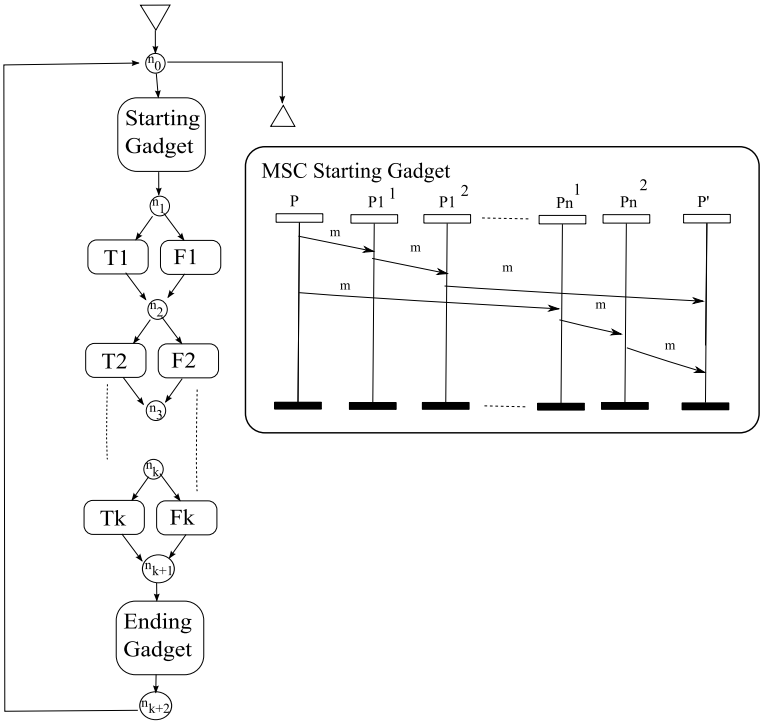


Figure 12.2: Gadget for the demonstration of co-NP completeness of non-divergence

Let us first show that finding a divergent cycle is an NP problem. For a cycle ρ of H, checking symmetry of the closure of a graph CG_{M_ρ} mainly consists in detecting connected component in a non-directed version of CG_{M_ρ} and in its directed version.

If the number of components coincide, then every message sent in is acknowledged directly or indirectly during an execution of M_ρ or at the next occurrence of M_ρ. If there are more components in the directed version of CG_M, then some messages are never acknowledged. This verification can be done in polynomial time using standard connected components algorithms [133]. For a given HMSC, computing the set of all elementary cycles and hence choosing one of them can be done in polynomial time [84]. Hence, exhibiting a counter example showing that a HMSC is divergent is in NP.

For the hardness part, we can proceed by reduction from the well-known 3CNF-SAT problem, that consists in checking satisfiability of a boolean formula in conjunctive normal form. This problem is known to be NP-Complete. The 3CNF-SAT problem is given as follows. Let φ be a formula of the form $\varphi = C_1 \wedge C_2 \wedge \ldots C_n$ with n clauses over a set of variables $V = v_1, \ldots v_k$. Each clause is a disjunction of three literals, i.e. each C_i is of the form $C_i = x_{i1} \vee x_{i2} \vee x_{i3}$, where each x_{i1}, x_{i2}, x_{i3} is either v_j or $\overline{v_j}$. The 3CNF-SAT problem consists in deciding whether there exists a valuation of variables $v_1, \ldots v_k$ such that φ holds for this valuation.

The reduction from 3CNF-SAT to a divergence problem can be done as follows. One can build in polynomial time a HMSC H_{div} with $k+3$ nodes, as illustrated in Figure 12.2. The *StartingGadget* MSC contains $2.n+2$ processes: two processes P and P' and for each clause C_i, two processes P_i^1, P_i^2. In this MSC, process P sends a message m to each process P_i^1, then P_i^1 sends a message m to process P_i^2, and last P_i^2 sends a message m to process P'. For every $i \in 1..n$, and each $j \in 1..k$ each MSC T_j contains a message from P_i^1 to P if the first literal of C_i is $\overline{x_j}$, a message from P_i^2 to P_i^1 if the second literal of C_i is $\overline{x_j}$, and finally a message from P' to P_i^2 if the last literal of C_i is $\overline{x_j}$.

For every $i \in 1..n$, and each $j \in 1..k$ each MSC F_j contains a message from P_i^1 to P if the first literal of C_i is x_j, a message from P_i^2 to P_i^1 if the second literal of C_i is x_j, and finally a message from P' to P_i^2 if the last literal of C_i is x_j.

MSCs T_j and F_j respectively correspond to choosing value *true* or $false$ in a valuation for variable x_j. Forcing a literal in a clause to be false by choosing a valuation results in a message between two processes.

The *EndingGadget* is simply the empty MSC. Formula φ is satisfiable if and only if there exists a valuation such that for every clause $C_i, i \in 1..n$, at least one of the literals is evaluated to *true*. In terms of the designed MSCs, it means that either a message from P' to P_i^2 from P_i^2 to P_i^1 or from P_i^1 to P is missing along a path from n_1 to n_{k+1}. Each of these path is a simple cycle of H, hence φ is satisfiable implies that there exists a simple loop in H which communication graph is not strongly connected. Note that H is divergent if and only if there exists a simple cycle ρ from node n_0 to node n_{k+2}, such that the communication graph of M_ρ is not strongly connected, as M_ρ is already connected, due to the *startinggadget* MSC. Hence, it means that one of the messages in the starting gadget is not acknowledged (directly or transitively). Note that as soon as there exists some $i \in 1..n$ such that a messages exists from P' to P_i^2 P_i^2 to P_i^1 and P_i^1 to P in M_ρ, then the communication graph of M_ρ is strongly connected, and clause C_i evaluates to false (and consequently the whole formula). Hence, finding a valuation for φ resumes to finding a divergent cycle in H_{div}. □

1.3 $Co - NP$ completeness of regular and globally cooperative HMSCs (Theorem9)

Unsurprisingly, verifying that a HMSC is regular or globally cooperative is a $Co-NP$ complete problem. Indeed, one can replace in the drawing of Figure 12.2 the starting gadget by a MSC containing

- only one message from P to P'. If there exists a clause in the modeled formula with its three literals false, then there exists a cycle which communication graph is not strongly connected.
- only one atomic action on instance P, and another one on P'. If there exists a clause in the modeled formula with its three literals false, then there exists a cycle which communication graph is not connected.

1.4 Local choices

Theorem 16 *Let H be an HMSC. H is* not local *iff there exists a node c and a pair of* ***acyclic*** *paths ρ, ρ' originating from c, such that M_ρ and $M_{\rho'}$ have more than one minimal instance.*

Proof: One direction is straightforward: if we can find a node c and two (acyclic) paths with more than one deciding instance, then obviously, c is not a local choice, and H is not local. Let us suppose now that for every node c, and for every pair of acyclic paths of H originating from c, we have only one deciding instance. Now, let us suppose that there exist a node c_1 and two paths ρ_1, ρ'_1 such that at least one (say ρ_1) of them is not acyclic. Then ρ_1 has a finite acyclic prefix w_1. The set of minimal instances in M_{w1} and in $M_{\rho 1}$ is the same, as $\varphi(min(M \circ M)) = \varphi(min(M))$. Hence, c, ρ_1, ρ'_1 are witnesses for the non-locality of H iff c, w_1, ρ'_1 are also such witnesses. □

Corollary 2 *Deciding if an HMSC is local-choice is in $co - NP$.*

Proof: The objective is to find a counter example, that is two paths originating from the same node with distinct deciding instances. One can choose in linear time in the size of H a node c and two finite acyclic paths ρ_1, ρ_2 of H starting from c, that is sequences of MSCs of the form $M_1 \dots M_k$. Note that to compute deciding instances in the MSC M_ρ attached to a path ρ, one does not need to compute the whole causal orderings in M_ρ. We can build incrementally the set of deciding instances of M_ρ as follows:

- If p is a process appearing in $Min(M_1 \circ M_i)$ then it is also a minimal process of $M_1 \circ M_i \circ M_{i+1}$.
- If p is a process of $Min(M_1 \circ M_i)$ that is not minimal, then it is not minimal either in $M_1 \circ M_i \circ M_{i+1}$.
- Last, if p is a process of $Min(M_{i+1})$ and does not appear in $M_1 \circ M_i$, then it is minimal in $M_1 \circ M_i \circ M_{i+1}$.

Finding minimal processes of an MSC can be done in polynomial time in the maximal number of events appearing in a MSC of H. Hence, maintaining the set of minimal

instances incrementally along a path can be performed in polynomial time in the size of H and in the size of MSC labeling H. Once the sets of minimal instances are built, one comparison suffices to decide whether two paths witness a non-local choice. □

2 Proofs for chapter 4

Proposition 3 *Let H, H' be two causal HMSCs over the* same *family of trace alphabets $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$. Consider the following six hypotheses:*

(i)	$caMSC(H) = caMSC(H')$	*(i)'*	$caMSC(H) \cap caMSC(H') \neq \emptyset$
(ii)	$Vis(H) = Vis(H')$	*(ii)'*	$Vis(H) \cap Vis(H') \neq \emptyset$
(iii)	$Lin(H) = Lin(H')$	*(iii)'*	$Lin(H) \cap Lin(H') \neq \emptyset$

Then we have:

- $(i) \Rightarrow (ii)$, $(i)' \Rightarrow (ii)'$, $(ii) \Rightarrow (iii)$ *and* $(ii)' \Rightarrow (iii)'$ *but the converses do not hold in general.*
- *If every causal MSC labeling transitions of H and H' respects $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$, then $(i) \Leftrightarrow (ii)$ and $(i)' \Leftrightarrow (ii)'$.*
- *If every causal MSC labeling transitions of H and H' is weak FIFO, then $(ii) \Leftrightarrow (iii)$ and $(ii)' \Leftrightarrow (iii)'$.*

The implications $(i) \Longrightarrow (ii)$ and $(ii) \Longrightarrow (iii)$ follow from the definitions. However, as shown in Figure 4.6, $Vis(G_1) = Vis(H_1)$ but $caMSC(G_1) \neq caMSC(H_1)$. And $Lin(G_2) = Lin(H_2)$ but $Vis(G_2) \neq Vis(H_2)$. Note that the independence relations are immaterial in these examples.

If every causal MSC labelling transitions of H and H' respects $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$, then one can define an equivalence relation $M \equiv M'$ on MSCs iff there exists a causal MSC C with $M, M' \in Vis(C)$. Then, for any causal MSC B in $caMSC(H) \bigcup caMSC(H')$, $Vis(B)$ is an equivalence class of that relation, and $(i) \Leftrightarrow (ii)$ and $(i)' \Leftrightarrow (ii)'$.

If every causal MSC labelling transitions of H and H' is weak FIFO, as remarked earlier, we know that all MSCs in $Vis(H) \cup Vis(H')$ are weak FIFO since the independence relations are irreflexive. Now, for each linearization w, one can reconstruct a unique weak FIFO MSC. Hence, if $Lin(M_1) = Lin(M_2)$ for M_1 and M_2 are in $Vis(H) \cup Vis(H')$, they are weak FIFO, and we necessarily have $M_1 = M_2$, and $(ii) \Leftrightarrow (iii)$ and $(ii)' \Leftrightarrow (iii)'$. □

2.1 Proof of Theorem 21

Theorem 21 *Let $H = (N, \longrightarrow, \mathcal{B}, n_0, F)$ be a causal HMSC. Testing whether H is regular (respectively globally-cooperative) can be done in time $\mathcal{O}((|N|^2 + |\Sigma|^2) \cdot 2^{|\mathcal{B}|})$. Furthermore these problems are co-NP complete.*

Proof: We use the ideas in the proofs of [56, 113], improving the deterministic complexity implied by the proof in [56], which was exponential in the number of transitions of the HMSC (there is at least one transition per label (else we can delete the useless labels)).

We first guess a subset $X = \{B_1, \cdots B_k\} \subseteq \mathcal{B}$ of causal MSCs and check that the communication graph of $B_1 \circledcirc \cdots \circledcirc B_k$ is not strongly connected (respectively disconnected). Using Tarjan's algorithm [133], this can be done in time linear in the number of edges of the communicating graph, that is quadratic in $|\Sigma|$. Then, we decompose the graph H_X into maximal strongly connected components in time $\mathcal{O}(|N|^2)$ using Tarjan again, where H_X is the restriction of H to transitions labeled by causal MSCs in X. Then it suffices to check in time $|N|^2$ whether one of this maximal strongly connected component uses all the labels from X. If it is the case, then we have a witness that H is not s-regular (resp. globally-cooperative). We thus obtain a co-NP algorithm. As there are $2^{|\mathcal{B}|}$ subsets X, this gives the deterministic time complexity.

The hardness part follows directly from the co-NP hardness result for HMSCs, as shown in chapter 2, theorem 9 (and from similar property for checking whether a HMSC is globally cooperative). Indeed any HMSC can be seen as a causal HMSC where the independence relation of each process is empty. □

2.2 Proof of Theorem 22 (Regular Causal HMSCs

In [91], the regularity of linearization languages of s-regular HMSC was proved by using an encoding into connected traces and building a finite state automaton which recognizes such connected traces. In our case, finding such embedding into Mazurkiewicz traces seems impossible due to the fact that causal MSCs need not be FIFO. Instead, we shall use techniques from the proof of regularity of trace closures of loop-connected automata from [43, 113].

The rest of this subsection is devoted to the proof of Theorem 22. We fix a s-regular causal HMSC H as in the theorem, and show the construction of the finite state automaton $\mathcal{A}_H$ over Σ which accepts $Lin(H)$.

First, we establish some technical results.

Lemma 1 *Let $\rho = \theta_1 \dots \theta_2 \dots \theta_{|\Sigma|}$ be a path of H, where for each $i = 1 \dots |\Sigma|$, the subpath $\theta_i = n_{i,0} \xrightarrow{B_{i,1}} n_{i,1} \dots n_{i,\ell_i-1} \xrightarrow{B_{i,\ell_i}} n_{i,0}$ is a cycle (these cycles need not be contiguous). Suppose further that the sets $\widehat{\mathcal{B}}_i = \{B_{i,1}, \dots, B_{i,\ell_i}\}$, $i = 1, \dots, |\Sigma|$, are equal. Let e be an event in $\circledcirc(\theta_1)$ and e' an event in $\circledcirc(\theta_{|\Sigma|})$. Let $\circledcirc(\rho) = (E, \lambda, \{\sqsubseteq_p\}, \mu)$. Then we have $e \leq e'$.*

Proof:

First of all notice that when (σ, σ') is an edge of CG_B, then for every causal MSC B', in the causal MSC $B \circledcirc B'$, every event e of B such that $\lambda(e) = \sigma$ precedes all events e' of B' such that $\lambda(e') = \sigma'$. Indeed, if σ and σ' belong to the same Σ_p, then (σ, σ') is an edge of CG_B if and only if $\sigma D_p \sigma'$, and we necessarily have $e \leq e'$ in $B \circledcirc B'$. Similarly, if σ and σ' label events located on different processes, then σ is of the form $p!q(m)$ and σ' of the form $q?p(m)$. Hence, there exists an event e'' of B such that $e \mu e''$ and $\lambda(e'') = \sigma'$. As the dependence relations are reflexive, we also have $e'' \leq e'$. Similarly, for two cycles θ_i,θ_{i+1} of ρ, if (σ, σ') is an edge of $CG_{B_{i,1} \circledcirc \cdots \circledcirc B_{i,l_i}}$, then all events in $\circledcirc(\theta_i)$ labelled by σ precede events of $\circledcirc(\theta_{i+1})$ labelled by σ'. As H is rigid, $CG_{B_{i,1} \circledcirc \cdots B_{i,l_i}}$ is strongly connected, and contains a path $(\sigma_1, \sigma_2) \cdots (\sigma_{k-1}, \sigma_k)$ of length at most $|\Sigma|$ from $\sigma_1 = \lambda(e)$ to $\sigma_k = \lambda(e')$, and we can find one event e_i, $i \in 1..|\Sigma|$ for each cycle such that $\lambda(e_i) = \sigma_i$ and $e = e_1 \leq e_2 \leq \cdots \leq e_{|\Sigma|} = e'$.

Let $\rho = n_0 \xrightarrow{B_1} \cdots \xrightarrow{B_\ell} n_\ell$ be a path in H, where $B_i = (E_i, \lambda_i, \{\sqsubseteq_p^i\}, \mu_i)$ for $i = 1, \ldots, \ell$. Let $\circledcirc(\rho) = (E, \lambda, \{\sqsubseteq_p\}, \mu, \leq)$. A *configuration* of ρ is a $\leq$-closed subset of E. Let C be a configuration of ρ. A *C-subpath* of ρ is a maximal subpath $\varrho = n_u \xrightarrow{B_{u+1}} \ldots \xrightarrow{B_{u'}} n_{u'}$, such that $C \cap E_i \neq \emptyset$ for each $i = u, \ldots, u'$. For such a *C-subpath* ϱ, we define its *C-residue* to be the set $(E_{u+1} \cup E_{u+2} \cup \cdots \cup E_{u'}) - C$. Figure 12.3 illustrates these notions for a path $\rho = n_0 \xrightarrow{B_1} n_1 \xrightarrow{B_2} n_2 \xrightarrow{B_3} n_3 \xrightarrow{B_4} n_4 \xrightarrow{B_5} n_5 \xrightarrow{B_6} n_6 \xrightarrow{B_7} n_7$. Each causal MSC is represented by a rectangle. Events in the configuration C are indicated by small filled circles, events not in C but the C-residues are indicated by small blank circles, and events that are not in C nor in its residues are indicated by blank squares. Note that the configuration contains only events from B_1, B_3, B_4 and B_5. The two *C-subpaths* identified on Figure 12.3 are the sequences of transitions $\rho_1 = n_0 \xrightarrow{B_1} n_1$ and $\rho_2 = n_2 \xrightarrow{B_3} n_3 \xrightarrow{B_4} n_4 \xrightarrow{B_5} n_5$ that provide the events appearing in C. One can also notice from this example that *C-subpaths* do not depend on the length of a the considered path, and that the suffix of each path that does not contain an event in C can be ignored.

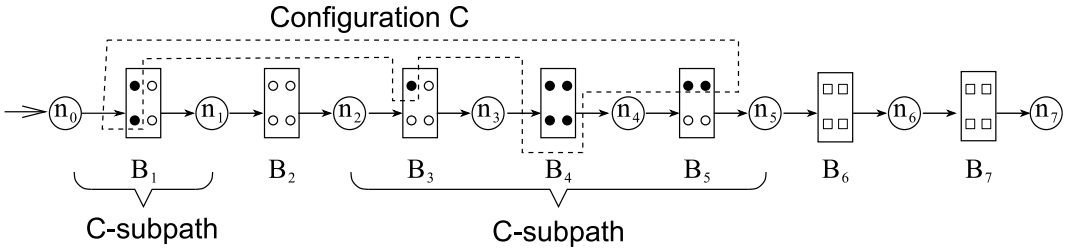


Figure 12.3: An example of path, a configuration C, and its C-subpaths.

Lemma 2 *Let ρ be a path in H and C be a configuration of ρ. Then,*

(i) The number of C-subpaths of ρ is at most $K_{subpath} = |N| \cdot |\Sigma| \cdot 2^{|\mathcal{B}|}$.

(ii) Let ϱ be a C-subpath of ρ. Then the number of events in the C-residue of ϱ is at most $K_{residue} = |N| \cdot |\Sigma| \cdot 2^{|\mathcal{B}|} \cdot \max\{|B| \mid B \in \mathcal{B}\}$.

Proof:

(i) Suppose the contrary. Let $K = |\Sigma| \cdot 2^{|\mathcal{B}|}$. We can find $K + 1$ C-subpaths whose ending nodes are equal. Let the indices of these $K + 1$ ending nodes be $i_1 < i_2 < \ldots < i_{K+1}$. For $h = 1, \ldots, K$, let θ_h be the subpath of ρ from n_{i_h} to $n_{i_{h+1}}$; and let $\widehat{\mathcal{B}}_h$ be the set of causal MSCs appearing in θ_h. Hence we can find θ_{j_1}, θ_{j_2}, ..., $\theta_{j_{|\Sigma|}}$, $j_1 < j_2 < \ldots < j_{|\Sigma|}$, such that $\widehat{\mathcal{B}}_{j_1} = \widehat{\mathcal{B}}_{j_2} = \ldots = \widehat{\mathcal{B}}_{j_{|\Sigma|}}$. Pick an event e from $\circledcirc(\theta_{j_1})$ with $e \notin C$. Such an e exists, since, for example, none of the events in the first causal MSC appearing in θ_{j_1} is in C. Pick an event e' from $\circledcirc(\theta_{j_{|\Sigma|}})$ with $e' \in C$. Applying Lemma 1 yields that $e < e'$. This leads to a contradiction, since C is $\leq$-closed.

(ii) Let $\varrho = n_i \xrightarrow{B_{i+1}} \ldots \xrightarrow{B_{i'}} n_{i'}$. Let $\widehat{E}_j = E_j - C$ for $j = i + 1, \ldots, i'$. By similar arguments as in (i), it is easy to show that among $\widehat{E}_{i+1}$, ..., $\widehat{E}_{i'}$, at most $|N| \cdot |\Sigma| \cdot 2^{|\mathcal{B}|}$ of them are nonempty. The claim then follows.

We are now ready to define the finite state automaton $\mathcal{A}_H = (S, S_\iota, \Sigma, S_{fi}, \Rightarrow)$ which accepts $Lin(H)$. As usual, S will be the set of states, $S_\iota \subseteq S$ the initial states, $\Longrightarrow \subseteq S \times \Sigma \times S$ the transition relation, and $S_{fi} \subseteq S$ the final states. Fix $K_{subpath}$, $K_{residue}$ to be the constants defined in Lemma 2. If $B = (E, \lambda, \{\sqsubseteq_p\}, \mu)$ is a causal MSC and E' a subset of E, then we define the restriction of B to E' to be the causal MSC $B' = (E', \lambda', \{\sqsubseteq'_p\}, \mu')$ as follows. As expected, λ' is the restriction of λ to E'; for each p, $\sqsubseteq'_p$ is the restriction of $\sqsubseteq_p$ to $(E' \cap E_p) \times (E' \cap E_p)$; and μ' is the restriction of μ to E'.

Intuitively, for a word σ in $\Sigma^\star$, $\mathcal{A}_H$ guesses an accepting path ρ of H and checks whether σ is in $Lin(\circledcirc(\rho))$. After reading a prefix σ' of σ, $\mathcal{A}_H$ memorizes a sequence of subpaths from which σ' was "linearized" (i.e the C-subpath of a path ρ such that C is a configuration reached after reading σ' and $\circledcirc(\rho)$ contains C). With Lemma 2, it will become clear later that at any time, we should remember at most $K_{subpath}$ such subpaths. Moreover, for each subpath, we need to know only a *bounded* amount of information, which will be stored in a data structure called "segment".

A causal MSC $B = (E, \lambda, \{\sqsubseteq_p\}, \mu)$ is of size lower than K if $|E| \leq K$. A *segment* is a tuple (n, Γ, W, n'), where $n, n' \in N$, Γ is a nonempty subset of Σ, and W is either a non-empty causal MSC of size lower than $K_{residue}$, or the special symbol $\perp$. The state set S of $\mathcal{A}_H$ is the collection of finite sequences $\theta_1\theta_2 \ldots \theta_\ell$, $0 \leq \ell \leq K_{subpath}$, where each θ_i is a segment. Intuitively, a segment (n, Γ, W, n') keeps track of a subpath ϱ of H which starts at n and ends at n'. Γ is the collection of letters of events in $\circledcirc(\varrho)$ that have been "linearized". Finally, W is the restriction of $\circledcirc(\varrho)$ to the set of events in $\circledcirc(\varrho)$ that are not yet linearized. In case all events in $\circledcirc(\varrho)$ have been linearized, we set $W = \perp$. For convenience, we extend the operator $\circledcirc$ by: $W \circledcirc \perp = \perp \circledcirc W = W$ for any causal MSC W; and $\perp \circledcirc \perp = \perp$.

We define $\mathcal{A}_H = (S, S_\iota, \Sigma, S_{fi}, \Longrightarrow)$ as follows:

- As mentioned above, S is the collection of finite sequence of at most $K_{subpath}$ segments.
- The initial state set is $S_\iota = \{\varepsilon\}$, where ε is the null sequence.
- A state is final iff it consists of a single segment $\theta = (n, \Gamma, \perp, n')$ such that $n \in N_\iota$ and $n' \in N_{fi}$ (and Γ is any nonempty subset of Σ).
- The transition relation $\Longrightarrow$ of $\mathcal{A}_H$ is the least set satisfying the following conditions.

  —**Condition** (i):

  Suppose $n \xrightarrow{B} n'$ where $B = (E, \lambda, \{\sqsubseteq_p\}, \mu, \leq)$. Let e be a minimal event in B (with respect to $\leq$) and let $a = \lambda(e)$. Let $\theta = (n, \Gamma, W, n')$ where $\Gamma = \{a\}$. Let $R = E - \{e\}$. If R is nonempty, then W is the restriction of B to R; otherwise we set $W = \perp$. Suppose $s = \theta_1 \ldots \theta_k\theta_{k+1} \ldots \theta_\ell$ is a state in S where $\theta_i = (n_i, \Gamma_i, W_i, n'_i)$ for each i. Suppose further that, e is a minimal event in $W_1 \circledcirc W_2 \circledcirc \ldots \circledcirc W_k \circledcirc W$.

  – ("create a new segment") Let $\hat{s} = \theta_1 \ldots \theta_k\theta\theta_{k+1} \ldots \theta_\ell$. If $\hat{s}$ is in S, then $s \overset{a}{\Longrightarrow} \hat{s}$. In particular, for the initial state ε, we have $\varepsilon \overset{a}{\Longrightarrow} \theta$.

– ("add to the beginning of a segment") Suppose $n' = n_{k+1}$. Let $\hat{\theta} = (n, \Gamma \cup \Gamma_{k+1}, \widehat{W}, n'_{k+1})$, where $\widehat{W} = W \circledcirc W_{k+1}$. Let $\hat{s} = \theta_1 \ldots \theta_k \hat{\theta} \theta_{k+2} \ldots \theta_\ell$. If $\hat{s}$ is in S, then $s \stackrel{a}{\Longrightarrow} \hat{s}$.

– ("append to the end of a segment") Suppose $n = n'_k$. Let $\hat{\theta} = (n_k, \Gamma_k \cup \Gamma, \widehat{W}, n')$, where $\widehat{W} = W_k \circledcirc W$. Let $\hat{s} = \theta_1 \ldots \theta_{k-1} \hat{\theta} \theta_{k+1} \ldots \theta_\ell$. If $\hat{s}$ is in S, then $s \stackrel{a}{\Longrightarrow} \hat{s}$.

– ("glue two segments") Suppose $n = n'_k$ and $n' = n_{k+1}$. Let $\hat{\theta} = (n_k, \Gamma_k \cup \Gamma \cup \Gamma_{k+1}, \widehat{W}, n'_{k+1})$, where $\widehat{W} = W_k \circledcirc W \circledcirc W_{k+1}$. Let $\hat{s}$ be $\theta_1 \ldots \theta_{k-1} \hat{\theta} \theta_{k+2} \ldots \theta_\ell$. If $\hat{s}$ is in S, then $s \stackrel{a}{\Longrightarrow} \hat{s}$.

—**Condition** (ii):

Suppose $s = \theta_1 \ldots \theta_k \theta_{k+1} \ldots \theta_\ell$ is a state in S where $\theta_i = (n_i, \Gamma_i, W_i, n'_i)$ for $i = 1, 2, \ldots, \ell$. Suppose $W_k \neq \perp$. Let $W_k = (R_k, \eta_k, \{\sqsubseteq_p^k\}, \mu_k, \leq_k)$ and e a minimal event in W_k. Suppose further that e is a minimal event in $W_1 \circledcirc W_2 \circledcirc \ldots \circledcirc W_k$.

Let $\hat{\theta} = (n_k, \Gamma_k \cup \{a\}, \widehat{W}, n'_k)$, where $\widehat{W}$ is defined as follows. Let $\widehat{R} = R_k - \{e\}$. If $\widehat{R}$ is nonempty, then $\widehat{W}$ is the restriction of W to $\widehat{R}$; otherwise, set $\widehat{W} = \perp$. Let $\hat{s} = \theta_1 \ldots \theta_{k-1} \hat{\theta} \theta_{k+1} \ldots \theta_\ell$. Then we have $s \stackrel{a}{\Longrightarrow} \hat{s}$, where $a = \eta_k(e)$. (Note that $\hat{s}$ is guaranteed to be in S.)

We have now completed the construction of $\mathcal{A}_H$. It remains to show that $\mathcal{A}_H$ recognizes $Lin(H)$.

Lemma 3 *Let $\sigma \in \Sigma^\star$. Then σ is accepted by $\mathcal{A}_H$ iff σ is in $Lin(H)$.*

Proof: Let $\sigma = a_1 a_2 \ldots a_k$. Suppose σ is in $Lin(H)$. Let $\rho = n_0 \xrightarrow{B_1} \ldots \xrightarrow{B_\ell} n_\ell$ be an accepting path in H such that σ is a linearization of $\circledcirc(\rho)$. Hence we may suppose that $\circledcirc(\rho) = (E, \lambda, \{\sqsubseteq_p\}, \mu, \leq)$ where $E = \{e_1, e_2, \ldots, e_k\}$ and $\lambda(e_i) = a_i$ for $i = 1, \ldots, k$. And $e_i \leq e_j$ implies $i \leq j$ for any i, j in $\{1, \ldots, k\}$. Consider the configurations $C_i = \{e_1, e_2, \ldots, e_i\}$ for $i = 1, \ldots, k$. For each C_i, we can associate a state s_i in $\mathcal{A}_H$ as follows. Consider a fixed C_i. Let $\rho = \ldots \varrho_1 \ldots \varrho_2 \ldots \varrho_h \ldots$ where ϱ_1, ϱ_2, …, ϱ_h are the C_i-subpaths of ρ. Then we set $s_i = \theta_1 \ldots \theta_h$ where $\theta_j = (n_j, \Gamma_j, W_j, n'_j)$ with n_j being the starting node of ϱ_j, and Γ_j the collection of all $\lambda(e)$ for all events e that are in both $\circledcirc(\varrho_j)$ and C_i. Let R_j be the C_i-residue of ϱ_j. If R_j is nonempty, W_j is the causal MSC $(R_j, \eta_j, \{\sqsubseteq_p^j\}, \mu_j, \leq_j)$ where η_j is the restriction of λ to R_j; $\sqsubseteq_p^j$ is the restriction of $\sqsubseteq_p$ to those events in R_j that belong to process p, for each p; and μ_j the restriction of μ to R_j. If R_j is empty, then set $W_j = \perp$. Finally, n'_j is the ending node of ϱ_j.

Now it is routine (though tedious) to verify that $\varepsilon \stackrel{a_1}{\Longrightarrow} s_1 \ldots s_{k-1} \stackrel{a_k}{\Longrightarrow} s_k$ is an accepting run of $\mathcal{A}_H$. Conversely, given an accepting run of $\mathcal{A}_H$ over σ, it is straightforward to build a corresponding accepting path of H.

With Lemma 3, we establish Theorem 22. As for complexity, the number of states in $\mathcal{A}_H$ is at most $(N_{seg})^{K_{subpath}}$, where N_{seg} is the maximal number of segments. Now, N_{seg} is $|N|^2 \cdot 2^{|\Sigma|} \cdot N_{res}$, where N_{res} is the possible number of residues. Recall that a residue is of size at most $K_{residue}$. According to Kleitman & Rotschild [89], the number of partial orders of size k is in $2^{f(k)}$ where $f(k) = \frac{1}{4}k^2 + \frac{3}{2}k + \mathcal{O}(\log_2(k))$. It

follows that the number of $|\Sigma|$-labeled posets of size k is in $2^{f(k)} \cdot |\Sigma|^k$. All residues of size up to k can be encoded as a labeled poset of size k with useless events, labelled by a specific label $\sharp$. Hence the number of residues N_{res} is lower than $2^{f(K_{residue})} \cdot (|\Sigma|+1)^{K_{residue}}$. Combining the above calculations then establishes the bound in theorem 22 on the number of states of $\mathcal{A}_H$.

2.3 Proof of Proposition 4

Proposition 4 *A causal MSC B can be effectively decomposed in time $O(|B|^2)$.*
Proof: Let $B = (E, \alpha, \{\sqsubseteq_p\}, \mu)$. We describe the decomposition of B, which is analogous to the technique in [71]. We consider the *directed* graph $(E, \leq \cup R)$, where R is the symmetric closure of $\mu \cup \left(\bigcup_{p\in\mathcal{P}} R'_p \cup R''_p\right)$, with

$$R'_p = \{(e,e') \in E_p \times E_p \mid e \widehat{\sqsubseteq}_p e' \text{ and } \alpha(e)\ I_p\ \alpha(e')\}\ ,$$
$$R''_p = \{(e,e') \in E_p \times E_p \mid e \not\sqsubseteq_p e' \text{ and } e' \not\sqsubseteq_p e \text{ and } \alpha(e)\ D_p\ \alpha(e')\}\ .$$

Intuitively, R'_p denote pairs of events that are ordered in B, but which labels are independent in I_p. As this ordering can not be obtained via composition, this ordering should appear in the decomposition of B, that is e and e' should belong to the same basic part. Similarly, relation R''_p contains pairs of events that are unordered in B, but which labels are dependent.

For each strongly connected component E' of $(E, \leq \cup R)$, we associate a structure $C = (E', \alpha', \{\sqsubseteq'_p\}, \mu')$, where α' is the restriction of α to E', $\sqsubseteq'_p$ is the restriction of $\sqsubseteq'_p$ to E', and μ' is the restriction of μ to E'. It is easy to see that C is a causal MSC, since each receive needs to be in the same strongly connected component than its associated send (since the relation includes the symmetric closure of μ). We first prove that C is a basic part. By contradiction, otherwise, we would have $C = B_1 \circledcirc B_2$, which by definition of $\circledcirc$ means that no edge of $\leq \cup R$ can go from one event of B_2 to one event of B_1, which contradicts the fact that E' is strongly connected.

Let $E_1, \ldots, E_n$ be the set of basic parts obtained. We order them such that there is no edge of $\leq \cup R$ from any event of E_j to some event of E_i with $i < j$ (it is always possible, else E_i, E_j would be in the same strongly connected component). It is now clear that $B = E_1 \circledcirc \cdots \circledcirc E_n$, since no event of E_j can be before an event of E_i, $i < j$ (else there would be an edge of $\leq$ from E_j to E_i). Notice that the decomposition in strongly connected components with Tarjan's Algorithm is in linear time in the number of edges, that is linear in $|B| + \sum_{p\in\mathcal{P}} |\sqsubseteq_p| \leq |B| + |B|^2$, where $|B|$ is the number of events of B. For comparison, recall that the complexity of decomposing an MSC B in atoms [71] is in $O(2|B|)$ (the immediate successor relation in MSCs is the union of the message pairing relation and the total ordering on instances, that is there are at most 2 immediate successors for a given event).□

2.4 Proof of theorem 23

Proposition 15 *Let H be a causal HMSC. Then $BP(H) = [\mathcal{L}_{Basic}(H)]$.*

Proof: First, let us take a word w in $[\mathcal{L}_{Basic}(H)]$. Thus $w = B_1 \ldots B_k \sim B_{i_1} \ldots B_{i_k}$ such that $B_{i_1} \ldots B_{i_k} \in \mathcal{L}_{Basic}(H)$. As $\mathcal{L}_{Basic}(H) \subseteq BP(H)$ and $B_1 \circledcirc \ldots \circledcirc B_k =$

$B_{i_1} \circledcirc \ldots \circledcirc B_{i_k}$ we conclude that $[\mathcal{L}_{Basic}(H)] \subseteq BP(H)$. Second, let us take a word w in $BP(H)$. Let us note $\circledcirc(w)$ its corresponding causal MSC, i.e. for $w = B_1 \ldots B_k$, $\circledcirc(w) = B_1 \circledcirc \ldots \circledcirc B_k$. Then this word is generated by an accepting path $\rho = n_0 \xrightarrow{P_1} n_1 \ldots \xrightarrow{P_l} n_l$ of H such that $\circledcirc(w) = P_1 \circledcirc \ldots \circledcirc P_l$. We know that any other decomposition of $\circledcirc(w)$ belongs to $[B_1 \ldots B_k]$, and in particular, the one we choose. Thus we obtain that $BP(H) \subseteq [\mathcal{L}_{Basic}(H)]$.□

Assuming we know how to compute the trace closure of the regular language $\mathcal{L}_{Basic}(H)$, we can obtain $BP(H)$ with the help of Proposition 15. In general, we cannot effectively compute this language. However if H is globally-cooperative, then $[\mathcal{L}_{Basic}(H)]$ is regular and a finite state automaton recognizing $[\mathcal{L}_{Basic}(H)]$ can be effectively constructed [43, 113]. Indeed, [113] shows that when a finite sate automaton $\mathcal{A}$ over an alphabet Σ is *loop conected* with respect to and independence alphabet I, that is when every loops of $\mathcal{A}$ are labeled by a word w that can not be defined as suffle of independant subwords (or equivalently, the letters of w connected by the dependency relation $D = \Sigma \times \Sigma \setminus I$ form a connected graph) , then one can compute a finite automaton $\mathcal{B}$ that recognizes $[\mathcal{A}]$. Let us consider globally-cooperative causal HMSCs as finite state automata over basic parts. When a causal HMSC H is globally cooperative, then for every loop of H labeled by a sequence of basic parts $B_1, \ldots, B_k$, we have that $(\{B_1, \ldots, B_k\}, D_{\mathcal{B}}\})$ forms a connected graph, as $B_1 \circledcirc \ldots \circledcirc B_k$ is tight. We can then apply [113] to obtain the following decidability and complexity result of theorem 23

2.5 Proof of Theorem 24

Theorem 24 *Let G, H be globally-cooperative causal HMSCs with respectively families of trace alphabets $\{(\Sigma_p, I_p)\}_{p\in\mathcal{P}}$ and $\{(\Sigma_p, J_p)\}_{p\in\mathcal{P}}$, where for each p, I_p and J_p are allowed to differ. Then determining if $Vis(G) \cap Vis(H) = \emptyset$ is undecidable.*
Proof: A PCP problem can be reduced to the emptiness of the intersection of the visual languages of two (globally-cooperative) causal HMSCs, if we *do not* assume that both causal HMSCs use the same independence relation.

Let J be a finite set and $(v_i, w_i)_{i\in J}$ be an instance of PCP, with $v_i, w_i \in \{a, b\}^* \setminus \varepsilon$ for all $i \in J$. We will use two causal HMSCs H_1 and H_2 to encode the PCP. The intuition for the reduction is that the causal HMSC H_1 generates sequences of CaMSCs of the form $(V_i W_i)^*$, where CaMSCs V_i and W_i represent respectively the words v_i and w_i. The causal HMSC H_2 generates sequences of the form $(A\bar{A} \vee B\bar{B})^*$, where the causal MSCs X and $\bar{X}$ represent an x letter in v and w respectively, with $x \in \{a, b\}$.

We have three process, P_1, P_2 and P_3. The causal MSCs A, B are made of a single message from process P_1 to process P_3, respectively labeled by a and b. The causal MSCs $\bar{A}, \bar{B}$ are made of two messages both labeled by the same letter (respectively $\bar{a}$ and $\bar{b}$). The first message is from process P_1 to P_2, and the second message is sent after the reception of the first message, from process P_2 to process P_3. For each pair (v_i, w_i) in the PCP instance, we build two causal MSCs V_i and W_i. If v_i is of the form $v_i = xyz$, the causal MSC V_i is made of the concatenation of X, Y, Z. Similarly, W_i is made of the concatenation $\bar{X}, \bar{Y}, \bar{Z}$, when $w_i = xyz$. These causal MSCs are depicted in Figure 12.4.

Let us denote by V the labels appearing in V_i's and by W the labels appearing

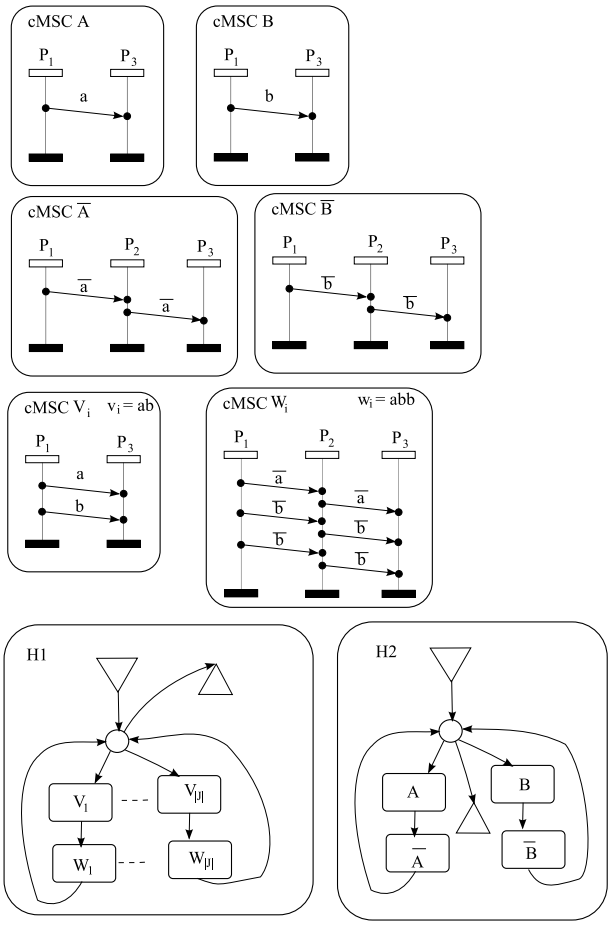


Figure 12.4: PCP encoding with two causal HMSCs

in W_i's. The independence relation I_1 for H_1 states that all events on process P_2 and P_3 commute. On process P_1, events labeled by a letter of V (namely $P_1!P_3(a)$ and $P_1!P_3(b)$ commute with events labeled by a letter of W (namely $P_1!P_2(\bar{a})$ and $P_1!P_2(\bar{b})$. There is no commutation among events labeled by a letter of V, and no commutations among events labeled by a letter of W. In the same way, the independence relation I_2 for H_2 states that no events on process 1 and 2 commute. On process 3, events from v (namely $3?1(a)$ and $3?1(b)$ commute with events from w (namely $3?2(\bar{a})$ and $3?2(\bar{b})$. There is no commutations among events from v, and no commutations among events from w.

It is easy to check that both H_1 and H_2 are globally-cooperative. Indeed, notice first that the letters a and b behave exactly the same. We can then forget about them for global cooperativeness, and draw the communication graph considering only 6 letters $P_1!P_2, P_2?P_1, P_2!P_3, P_3?P_1, P_1!P_3, P_3?P_1$. Every elementary cycle of H_1 contains a V_i and a W_i. Since v_i, w_i are non empty words, every of these 6 letters appear in every loop of H_1. In particular, we have the undirected relation $P_1!P_2 - P_2?P_1 - P_2!P_3 - P_3?P_2 - P_3?P_1 - P_1!P_3$, which proves that the graph is connected. In the same way, every loop of H_2 contains a $X\bar{X}$, hence every of the 6 letters appear in every elementary cycle (and loop) of H_2. This time, the graph is connected, but through another undirected path: $P_3?P_1 - P_1!P_3 - P_1!P_2 - P_2?P_1 - P_2!P_3 - P_3?P_2$. Hence, both H_1 and H_2 are globally-cooperative.

Assume that $Vis(H_1) \cap Vis(H_2) \neq \emptyset$. Let $M \in Vis(H_1) \cap Vis(H_2)$. Let v be the projection of M on alphabet $P_1!P_3(a), P_1!P_3(b)$, and w the projection of M on alphabet $P_1!P_2(\bar{a}), P_1!P_2(\bar{b})$. Now, because $M \in Vis(H_2)$ and since there is no commutation on process P_1 allowed by I_2, we get that $v = w$, confusing $P_1!P_2(\bar{a})$ with $P_1!P_3(a)$ and $P_1!P_2(\bar{b})$ with $P_1!P_3(b)$.

Second, because $M \in Vis(H_1)$, there exists a sequence $i_1 \cdots i_n \in J^*$ with $M \in Vis(V_{i_1} \circledcirc W_{i_1} \circledcirc \cdots V_{i_n} \circledcirc W_{i_n})$. Since by I_1, there is no commutation among letters of v, the projection v of M on alphabet $P_1!P_3(a), P_1!P_3(b)$ is the same as the projection of $V_{i_1} \circledcirc W_{i_1} \circledcirc \cdots V_{i_n} \circledcirc W_{i_n}$. That is, $v = v_{i_1} \cdots v_{i_n}$ (confusing letter a with $P_1!P_3(a)$ and letter b with $P_1!P_3(b)$). In the same way, $w = w_{i_1} \cdots w_{i_n}$ (confusing letter a with $P_1!P_2(\bar{a})$ and letter b with $P_1!P_2(\bar{b})$). That is $v_{i_1} \cdots v_{i_n} = v = w = w_{i_1} \cdots w_{i_n}$, which proves that it is a solution for the PCP problem.

Now, assume that the instance $(v_i, w_i)_{i \in I}$ of PCP has a solution $v_{i_1} \cdots v_{i_n} = w_{i_1} \cdots w_{i_n} = x_1 \cdots x_m$. Consider the following MSC M. We describe M process by process (which is enough to uniquely define M since all MSCs in $Vis(H_1)$ and $Vis(H_2)$ are weak FIFO). On process P_1, M is of the form $P_1!P_3(x_1)P_1!P_2(\bar{x_1}) \cdots P_1!P_3(x_m)P_1!P_2(\bar{x_m})$. On process P_2, M is of the form $P_2?P_1(\bar{x_1})P_2!P_3(\bar{x_1}) \cdots P_2?P_1(\bar{x_m})P_2!P_3(\bar{x_m})$. On process 3, M is of the form $P_3?P_1(a_1)P_3?P_1(b_1)P_3?P_1(c_1)P_3?P_2(\bar{d_1})P_3?P_2(\bar{e_1})P_3?P_1(\bar{f_1}) \cdots P_3?P_1(a_n)$ $P_3?P_1(b_n)P_3?P_1(c_n)P_3?P_2(\bar{d_n})P_3?P_2(\bar{e_n})P_3?P_1(\bar{f_n})$, where for all j, $v_{i_j} = a_j b_j c_j$ and $w_{i_j} = d_j e_j f_j$. It is easy to see that $M \in Vis(H_1) \cap Vis(H_2)$, which ends the proof.

2.6 Proof of Theorem 25

The main principle is to build a finite state automaton whose states remember the labels of events that must appear in the *future* of messages (respectively in the *past*) in any MSC of $Vis(H)$.

Formally, for a causal MSC $B = (E, \alpha, \{\sqsubseteq_p\}, \mu)$ $(e, f) \in \mu$ a message of B, we define the future and past of (e, f) in B as follows:

$$\begin{aligned} \text{Future}_B(e, f) &= \{a \in \Sigma \mid \exists x \in E, f \leq x \wedge \alpha(x) = a\} \\ \mathit{Past}_B(e, f) &= \{a \in \Sigma \mid \exists x \in E, x \leq e \wedge \alpha(x) = a\} \end{aligned}$$

Notice that for a message $m = (e, f)$, we always have $\alpha(e) \in \mathit{Past}_B(m)$ and $\alpha(f) \in \text{Future}_B(m)$. For instance, in Figure 4.11, $\mathit{Past}_B(m_1) = \{p!q(Q), q?p(Q), q!p(A)\}$.

Intuitively, if a letter of the form $p!q(m)$ is in the future of a message (e, f) in a causal MSC B, then any occurrence of message m that is appended to B *is* in the future of (e, f). Hence, this message can not appear in the window of (e, f). Note that a symmetric property holds for the past of (e, f). Furthermore, from the definition of future and past of a message, we easily obtain the following proposition.

Proposition 16 *Let $B = (E, \alpha, \{\sqsubseteq_p\}, \mu)$ and $B' = (E', \alpha', \{\sqsubseteq'_p\}, \mu')$ be two causal MSCs, and let $m \in \mu$ be a message of B. Then we have:*

$$\begin{aligned} &\mathit{Future}_{B \circledcirc B'}(m) = \mathit{Future}_B(m) \cup \{a' \in \Sigma \mid \exists x, y \in E' \\ &\quad \exists a \in \mathit{Future}_B(m) \text{ s.t. } \alpha(y) = a' \wedge x \leq' y \wedge a \; D_{loc(a)} \; \alpha(x)\} \end{aligned}$$

$$\begin{aligned} &\mathit{Past}_{B' \circledcirc B}(m) = \mathit{Past}_B(m) \cup \{a' \in \Sigma \mid \exists x, y \in E' \\ &\quad \exists a \in \mathit{Future}_B(m) \text{ s.t. } \alpha(y) = a' \wedge y \leq' x \wedge a \; D_{loc(a)} \; \alpha(x)\} \end{aligned}$$

This proposition is important, as it can be used to show that the futures and past of a message can be computed incrementally. Let $H = (N, \longrightarrow, \mathcal{B}, n_0, F)$ be a causal HMSC. Consider a path ρ of H with $\circledcirc(\rho) = B_1 \circledcirc \cdots \circledcirc B_\ell$ and a message m in B_1. First, the sequence of sets $\text{Future}_{B_1}(m)$, $\text{Future}_{B_1 \circledcirc B_2}(m)$, ..., $\text{Future}_{B_1 \circledcirc \cdots \circledcirc B_\ell}(m)$ is non-decreasing. Using proposition 16, these sets can be computed on the fly and with a finite number of states. Similar arguments hold for the past sets. Now consider a message (e, f) in a causal MSC B labelling some transition t of H. With the above observation on Future and *Past*, we can show that, if there is a bound $K_{(e,f)}$ such that the window of a message (e, f) in the causal MSC generated by any path containing t is bounded by $K_{(e,f)}$, then $K_{(e,f)}$ is at most $b|N||\Sigma|$ where $b = \max\{|B| \mid B \in \mathcal{B}\}$. Further, we can effectively determine whether such a bound $K_{(e,f)}$ exists by constructing a finite state transition system whose states memorize the future and past of (e, f). We are now ready to prove Theorem 25.

Theorem 25 *Let $H = (N, \longrightarrow, \mathcal{B}, n_0, F)$ be a causal HMSC. Then we have:*

(i) If H is window-bounded, then H is K-window-bounded, where K is at most $b \cdot |N| \cdot |\Sigma|$, where $b = max\{|B| \mid B \in \mathcal{B}\}$.

(ii) Further, we can effectively determine whether H is window-bounded in time $O(s \cdot |N|^2 \cdot 2^{|\Sigma|})$, where s is the sum of the sizes of causal MSCs in $\mathcal{B}$.

Proof:[of Theorem 25(i)] Suppose that H is not k-window-bounded, where $k = b \cdot |N| \cdot |\Sigma|$. Let $B \in \mathcal{B}$ be a causal MSC in $\mathit{caMSC}(H)$, and let the pair (e, f) be a message of B. Let ρ be a path of H, and $V \in \mathit{Vis}(\circledcirc(\rho))$ be an MSC such that the message (e, f) is crossed by $k + 1$ messages in V. Recall that a causal MSC contains at most $b/2$ messages. Then, ρ contains at least $2|\Sigma|$ occurrences of

a node n', such that the label of n' contains at least a message m' that crosses (e, f) in V. Without loss of generality, we can consider that n' is repeated at least $|\Sigma|$ times after B in ρ (else we apply a symmetric proof, considering repetitions of B' occurring before B). That is, ρ is of the form $\cdots n_0 \rightarrow \cdots n_1 \rightarrow \cdots n_2 \rightarrow \cdots n_{|\Sigma|} \rightarrow \cdots$, where $n_1 = \cdots = n_{|\Sigma|} = n'$.

Let us denote by E_i the label of the prefix $\cdots n_0 \rightarrow \cdots n_1 \rightarrow \cdots n_2 \rightarrow \cdots n_i$ of ρ, for $i = 0, 1, 2, \ldots, |\Sigma|$. Consider the sequence of sets $F_i = \text{Future}_{E_i}(e, f)$, $i = 0, 1, \ldots, |\Sigma|$. Each F_i is a subset of Σ and the sequence F_0, F_1, ..., $F_{|\Sigma|}$ is non-decreasing. Hence, we can find $\ell \leq |\Sigma|$ such that $F_\ell = F_{\ell+1}$. This means that the path ρ', which is computed from path ρ by repeating twice the cycle between n_ℓ and $n_{\ell+1}$ (n_ℓ excluded but $n_{\ell+1}$ included), have at least one more message m' which crosses m. Thus, we can exhibit a new execution $V' \in \mathit{Vis}(\circledcirc(\rho'))$ such that (e, f) is crossed by at least $k + 2$ messages. As we can iterate this construction, it means that H is not window-bounded.

We next establish Theorem 25(ii). We shall show that for a given message m, one can decide in an efficient way whether there is a window bound, by constructing a finite state transition system that memorizes Future(m) and $\mathit{Past}(m)$.

For a given causal HMSC $H = (N, N_\iota, \mathcal{B}, N_{fi}, \longrightarrow)$ and a message (e, f) of some causal MSC $B \in \mathcal{B}$, we build the following transition system $\mathcal{A}_{(e,f)} = (Q, Q_\iota, \mathcal{B}, Q_{fi}, \delta)$ that computes the possible futures of (e, f), where:

- $Q = N \times 2^\Sigma$ is a set of states, recalling a node in H and a future,
- $Q_\iota = \{(n, \emptyset) \mid n \in N_\iota\}$ is the set of initial states,
- $(n, X) \in Q_{fi}$ if and only if $n \in N_{fi}$.
- $\delta \subseteq Q \times \mathcal{B} \times Q$ is the least transition relation such that:
  - $\big((n, \emptyset), B, (n', \emptyset)\big) \in \delta$ if $n \xrightarrow{B} n'$
  - $\big((n, \emptyset), B, (n', \text{Future}_B(e, f))\big) \in \delta$ if $n \xrightarrow{B} n'$ and (e, f) belongs to B.
  - $\big((n, X), B, (n', X')\big) \in \delta$, where $B = (E, \lambda, \{\sqsubseteq_p\}, \mu, \leq)$, if $n \xrightarrow{B} n'$, and $X' = X \cup \{\lambda(y) \mid \exists x, y \in E, \exists a \in X \wedge a \; D \; \lambda(x) \wedge x \leq y\}$.

Note that the first rule in the construction of the transition relations of $\mathcal{A}_{(e,f)}$ simply copies the transitions of H. The second rule perform a random choice of a message in a random occurrence of a transition of H labelled by B. This rule is important, as it allows to chose nondeterministically an occurrence (e, f) of a message after an arbitrary path in the HMSC. The last rule updates the futures or pasts after the choice of a message occurrence. A state $q = (n, X)$ in $\mathcal{A}_{(e,f)}$ represents a possible set X of labels in $\text{Future}_{\circledcirc(\rho)}(e, f)$ for some path ρ that ends (respectively starts) at node n in H, and contains a message (e, f). Slightly abusing the notation, we will denote by Future(q) (resp. $\mathit{Past}(q)$) the set X. Note that in any strongly connected subset $C = \{q_1, \ldots, q_k\}$ of $\mathcal{A}_{(e,f)}$ (respectively $\mathcal{A}'_{(e,f)}$), $\text{Future}(q_1) = \text{Future}(q_2) = \cdots = \text{Future}(q_k)$ (resp. $\mathit{Past}(q_1) = \mathit{Past}(q_2) = \cdots = \mathit{Past}(q_k)$). Hence, we will denote by Future(C) (resp. $\mathit{Past}(C)$) the set of observed labels on any state of C.

We can also build a finite state transition system $\mathcal{A}'_{(e,f)}$ that computes the possible pasts of (e, f), by a backward search in the causal HMSC H.

More precisely, $\mathcal{A}'_{(e,f)} = (Q', Q'_\iota, \mathcal{B}, Q'_{\text{fi}}, \delta')$ where

- $Q' = N \times 2^\Sigma$
- $Q'_\iota = N_{\text{fi}} \times \{\emptyset\}$
- $Q'_{\text{fi}} = N_\iota \times 2^\Sigma$
- $\delta' \subseteq Q \times \mathcal{B} \times Q$ is the least relation such that:
  - $\big((n, \emptyset), B, (n', \emptyset)\big) \in \delta$ if $n' \xrightarrow{B} n$
  - $\big((n, \emptyset), B, (n', Past_B(e, f))\big) \in \delta$ $n' \xrightarrow{B} n$ and (e, f) belongs to B.
  - $\big(n, X), B, (n', X')\big) \in \delta$, where $B = (E, \lambda, \{\sqsubseteq_p\}, \mu, \leq)$, if $n' \xrightarrow{B} n$, and $X \neq \emptyset$, and $X' = X \cup \{a' \in \Sigma \mid \exists x, y \in E, \exists a \in Past_B(e, f), \lambda(y) = a' \wedge y \leq x \wedge a \; D \; \lambda(x)\}$

We observe the following properties of the finite state automata $\mathcal{A}_{(e,f)}$ and $\mathcal{A}'_{(e,f)}$.

Lemma 4 *Let $H = (N, N_\iota, \mathcal{B}, \longrightarrow, N_{\text{fi}})$ be a causal HMSC. Let B be a causal MSC in $\mathcal{B}$ and (e, f) a message in B with the label of e being $p!q(m)$. Consider the finite state automata $\mathcal{A}_{(e,f)}$ and $\mathcal{A}'_{(e,f)}$ as constructed above. Then, H is window-bounded iff both of the following conditions hold:*

- *There does* not *exist a strongly connected component C in $\mathcal{A}_{(e,f)}$ and a letter $q!p(m') \in \Sigma$ such that $q!p(m')$ is in $Alph(B) - Future(C)$ for some causal MSC B labelling a transition in C.*
- *There does* not *exist a strongly connected component C in $\mathcal{A}'_{(e,f)}$ and a letter $q!p(m') \in \Sigma$ such that $q!p(m')$ is in $Alph(B) - Past(C)$ for some causal MSC B labelling a transition in C.*

Proof: One direction is straightforward. If any of these strongly connected components exists (either before or after m), then there is an unbounded number of path generating an unbounded number of occurrences of $q!p(m')$ that are not causally related to m. Hence, for each of these path, there is a visual extension where all m' generated by occurrences of the cycle cross m, and the window size of m is not bounded. The other direction is a direct consequence of Theorem 25(i).

Thus Theorem 25(ii) follows from Lemma 4. It remains to establish the complexity claim in Theorem 25(ii). The transition system $\mathcal{A}_{(e,f)}$ has at most $|N| \times 2^{|\Sigma|}$ states, and we have to analyze strongly connected components of $\mathcal{A}_{(e,f)}$. However, as noticed before, every strongly connected component of $\mathcal{A}_{(e,f)}$ enjoys the property to have a second component which is constant. Hence we need to test the property only for *maximal* strongly connected components. Indeed, if C is a strongly connected component of $\mathcal{A}_{(e,f)}$ such that $q!p(m')$ is the label of an event in a causal MSC labeling a transition of C but that is not in Future(C), then we can consider the maximal strongly connected component D of $\mathcal{A}_{(e,f)}$ containing C (it exists since

the union of two non disjoint strongly connected components is again a strongly connected component). Since D is a strongly connected component, its second component Future(D) is constant, hence Future(D) = Future(C). Since $C \subseteq D$, we have that $q!p(m')$ is a label of an event of D and is not in Future(C) = Future(D).

Using Tarjan's algorithm [133], we can compute in quadratic time the partition of $\mathcal{A}_{(e,f)}$ into maximal strongly connected components (for each set $X \subseteq 2^{\Sigma}$, we partition the subpart of $\mathcal{A}_{(e,f)}$ with a constant second component being X). Then for each maximal strongly connected component (C, X), it suffices to compute $\lambda(C)$ and to compare it with X, which is linear in n. Hence, the overall complexity of the algorithm is in $O(|N|^2 \cdot 2^{|\Sigma|})$. As, we build the two automata $\mathcal{A}_{(e,f)}$ and $\mathcal{A}_{(e,f)}$ for each occurrence (e, f) of a message in each causal MSC labeling a transition of H, we obtain a complexity in $\mathcal{O}(s.|N|^2.2^{|\Sigma|})$, where s is the sum of the sizes of causal MSCs in $\mathcal{B}$.

3 Proofs for chapter 5

3.1 Proof sketch for theorem 27

Proof sketch The main idea of this proof is to build a structure labeled by MSC projections. An event in the original model can be an atomic action, a send, or a receive event. A non-erased event becomes a new event of a different type. For instance, an event e which is a send to a process p may become the immediate predecessor of another event located on a process q. Hence, the type of e in the projection should be a send to process q. The construction of the CHMSC then works by memorizing for a subset of non-erased event of the path a guess on their type. Such guesses have to be verified when new MSCs are appended to the path. We can show that for any MSC labeling a path of H, only the last node of the path, a finite number of guesses (at most $\wp^2$) during projection and for each guess the information on whether an immediate successor of an non-erased event can appear on a process $p \in \wp$ in the projection. This can be recorded for any path as a triples $(n, to, Live, Dead)$ where n is the last node of a path of H, to is a guess on the type of an event of the followed path (yet unproved guesses), and $Live, Dead$ are mappings from the set of events with unproved guesses to subsets of $\wp$. The exact nature and place of events with unverified guesses in the path need not be remembered, yielding the finiteness of the construction and the complexity statement. Transitions from a node to another is allowed iff it does not contradict the guesses. Accepting nodes are nodes in which all guesses have been proved correct. We refer interested readers to [59] for more details on the construction.

Our construction can be easily modified in the presence of multi-type events to obtain a safe cHMSC (actually an extendion of the model to allow multitype events).

3.2 Proof of Theorem 29

Let us first address the question of the finiteness of the set of generators for a given safe cHMSC. We show below how to construct a compact representation of the generators (atoms) of a given safe cHMSC G. This problem is directly related to the construction for a given safe cHMSC G (or a pHMSC) of an equivalent HMSC, if it exists, and it involves the computation of the smallest MSCs that are factors of some MSC $M \in \mathbb{L}(G)$.

Formally, given a safe cHMSC G we want to compute the set $\mathrm{Gen}(G)$ of *generators* of G defined as follows. An *atomic* MSC M belongs to $\mathrm{Gen}(G)$ if $N = N_1 M N_2$ for some $N \in \mathbb{L}(G)$ and some MSCs N_1, N_2.

Clearly, for a safe cHMSC to be equivalent to an HMSC, $\mathrm{Gen}(G)$ needs to be finite. At the end of this section we will show that if $\mathrm{Gen}(G)$ is finite, then we can construct effectively an equivalent HMSC.

An Automata-Based Representation of Generators

We show now how to construct for a given safe cHMSC G a finite automaton $\mathcal{A}(G)$ that accepts only linearizations of $\mathrm{Gen}(G)$ and such that for every $M \in \mathrm{Gen}(G)$, at least one linearization of M is accepted by $\mathcal{A}(G)$. The idea is to have a non-deterministic automaton $\mathcal{A}(G)$ that works as follows. Given an execution $z \in \mathbb{L}(G)$,

the automaton guesses a factor w of z containing a generator $M \in \mathrm{Gen}(G)$ and extracts the events of M from w. Note that since we cannot choose the linearization z, such a generator M won't be itself a factor of z.

The automaton $\mathcal{A}(G)$ must check two conditions in the definition of the generating set $\mathrm{Gen}(G)$. First, we guess for each event e of the linearization z above a type $k \in \{0, 1, 2\}$, telling whether e belongs to N_1 ($k = 0$), to M ($k = 1$) or to N_2 ($k = 2$). Note that if an event e belongs to N_1, then all its predecessors on the same process must also belong to N_1. Similarly, when an event belongs to N_2, then all its successors on the same process must belong to N_2.

In addition, we check whether the guessed type is consistent w.r.t messages as atoms shall not cut messages. That is, every send mist be of the same type as its associated receive. By maintaining such consistency, we can guarantee guarantee that N_1, N_2 and M are all complete MSCs. We recall that safe cHMSCs are existentially bounded for some bound b, that is there exists an automaton $A^b(G)$ that recognizes all b-bounded linearizations of G. To each state q of $A^b(G)$, one can associate a set $S(q)$ of at most b send event which have not yet been matched to a receive event, a mapping $statP : \wp \longrightarrow \{0, 1, 2\}$ that indicates whether events seen on process p after the current state will belong to N_1, M, or N_2, and a map $statS : S(q) \longrightarrow \{0, 1, 2\}$ that indicates whether a send event for which the corresponding reception is not yet executed belonged to N_1, M, or N_2. With this extension, we to obtain a new automaton $B^b(G)$. Note that $S(q)$ is an information that is already contained in the states of $A^b(G)$. Transitions of $B^b(G)$ are of the form $(q, S, statP, statS) \xrightarrow{e} (q', S', statP', statS')$ in which:

- $q \xrightarrow{e} q'$ is a transition of $A^b(G)$.
- for every $p \in \wp$ we have $statP(p) \leq statP'(p)$
- if e is a send event, then $S' = S \cup \{e\}$ and $statS(e) = statP'(\varphi(e))$
- if e is a reception, and f is the corresponding event, then we require that $statP(\varphi(e)) = statS(f)$. Hence, if an incorrect guess was made (i.e predecessors of e were declared as part of the wrong factor), the reception can not occur. Then $S' = S \setminus \{e\}$.
- final states of $B^b(G)$ are states of the form $(q, S, statP, statS)$ in which q is a final state of $A^b(G)$. Initial states of $B^b(G)$ are states of the form $(q_0, \emptyset, statP, statS)$ in q_0 is the initial state of $A^b(G)$, and for every $p \in \wp$, $statP(p) \in \{0, 1, 2\}$.

The automaton $B^b(G)$ recognizes b-bounded linearizations of G, and can be used to recognize inner factors of these linearizations. These inner factors are sub-words recognized by transitions of the form $(q, S, statP, statS) \xrightarrow{e} (q', S', statP', statS')$ where $StatP(\varphi(e)) = 1$, i.e. transitions performed when the process executing e is chosen to start recognizing M. Note that nothing guarantees in $B^b(G)$ that M is atomic. Proposition 2 gives an algorithm for verifying that a given MSC M is atomic, checking that the connection graph $Conn(M)$ is strongly connected.

We have now to test whether an MSC M is atomic using a finite automaton, that takes as input some arbitrary linearization of M. However, we cannot use directly

the algorithm of [71], since the number of connected components in a linearization is not a priori bounded. In order to handle this, we restate the result of [71] as follows:

Proposition 17 *Let M be an MSC. Then M is atomic if and only if for every pair of processes p, q, there is a path in $Conn(M)$ from the last event of p to the first event of q.*

We say that an event e of M *sees* another event f if there exists a path from e to f in $Conn(M)$. To ensure that the last event in the inner factor M sees the first event of M, we add two other components to $B^b(G)$, namely a set X that recalls the last events on each process seen in the inner factor M, plus a set MS of unmatched send events that belong to M. Let last_p denote the last event on process p seen so far in X. Now, for each pair $(x, p) \in X \times \{1, \ldots, \wp\}$ we record an integer $T(x, p) \in \{0, 1, 2\}$ telling whether x sees the first event on process p in M ($T(x, p) = 2$), or whether x sees last_p but not the first p-event in M ($T(x, p) = 1$), or whether it sees no p-event in M at all ($T(x, p) = 0$). For every event $x \in X$ we also record the unmatched send events seen by x, using a function $TS : X \longrightarrow 2^M S$.

Then, we can obtain an automaton $\mathcal{A}(G)$ which states are tuples of the form $(q, S, statP, statS, X, MS, T, TS)$, with initial states $(q_0, S, statP, statS, \emptyset, \emptyset, T, TS)$ for which $(q, S, statP, statS)$ in an initial state of $B^b(G)$. The final states of $\mathcal{A}(G)$ are states such that $(q, S, statP, statS)$ is accepting in $B^b(G)$ and furthermore $T(x, p) = 2$ for every pair $x, p \in X \times \wp$. Now, transitions of $\mathcal{A}(G)$ are of the form

$$(q, S, statP, statS, X, MS, T, TS) \xrightarrow{e} (q', S', statP', statS', X', MS', T', TS')$$

, where $(q, S, statP, statS) \xrightarrow{e} (q', S', statP', statS')$, is a transition of $B^b(G)$, and :

- $X' = X$ if $statP(\varphi(e)) \in \{0, 2\}$ and $X' = X \setminus last_p \cup \{e\}$ otherwise.
- $MS' = MS$ if $statP(\varphi(e)) \in \{0, 2\}$ and $MS' = MS \cup \{e\}$ is a send event. $MS' = MS \setminus \{f\}$ if e is a receive event and f is the corresponding matched send event.
- $TS' = TS$ if $statP(\varphi(e)) \in \{0, 2\}$. If e is a send event, and $statP(\varphi(e)) = 1$, then for every $x \in X$, $TS'(x) = TS(x) \cup \{e\}$ if $T(x, \varphi(e)) > 0$ and $TS'(x) = TS(x) \cap MS$ otherwise. $TS'(e) = \emptyset$ If e is a receive event corresponding to a sending event $f \in MS$, then $TS'(x) = TS(x) \cup \{e\}$ if $T(x, \varphi(e)) > 0$ ans $TS'(e) = TS(f)$.
- $T'(x, p) = T(x, p)$ if $statP(\varphi(e)) \in \{0, 2\}$. If $statP(\varphi(e)) = 1$ and e is a send event, then $T'(e, p) = 0$ for every $p \neq \varphi(e)$, $T'(e, \varphi(e) = 1$ and $T'(x, \varphi(e)) = T'(x, phi(e))$ for every $x \in X$. If $statP(\varphi(e)) = 1$ and e is a reception of a message m from p to q, and e matched a send event $f \in MS$, then for every $x \in X$, such that $f \in TS(x)$ we have $T'(x, \varphi(e)) = \max(T(x, \varphi(e)), 1)$. We also set $T'(e) = T(f)$.

Clearly, computing the functions T and TS for each event last_p suffices for deciding whether $Conn(M)$ is strongly connected. Indeed, if for every $p, q \in \wp$, $TS(\text{last}_p, q) = 2$, then the factor M currently recognized by $\mathcal{A}(G)$ is an atom. If we project $\mathcal{A}(G)$ on transitions that recognize events of factor M, then we obtain an

automaton $\mathcal{A}^{gen}(G)$ that recognizes linearizations of atoms of G. Then, by checking if $\mathcal{A}^{gen}(G)$ recognizes finite words, one can decide if $Gen(G)$ is finite.

We are now ready to prove Theorem 29:

From the construction, it is obvious that $\mathcal{A}^{gen}(G)$ accepts only (b-bounded) linearizations of $Gen(G)$. Now, it remains to show that every atom in $Gen(G)$ has at least one linearization in $\mathcal{A}^{gen}(G)$. Suppose that an atom M of G is not recognized by $\mathcal{A}^{gen}(G)$. then, as M is an atom, there exists an accepting path ρ of G such that ρ generates a MSC $M_\rho = N_1 \circ M \circ N_2$. A b-bounded linearization w of M_ρ mapping correctly events of M is accepted by $B^b(G)$, as M, N_1, and N_2 are is closed. Hence, M is not recognized if and only if the suword w' corresponding to events of M in w leads to a state in which there exists p, q such that $T(\mathsf{last}_p, q) \neq 2$. We can show by induction on the size of atoms that such situation can not occur.

3.3 Proof of theorem 30

The PSPACE complexity comes from the fact that we have a bound on the size of $\mathcal{A}^{gen}(G)$, and hence we can perform a random walk in the automaton while rembering the number of events of M. If this number exceeds the bound, then a witness was found. If the run stops after a number of steps lower than the bound, no witness was found. As the size needed to remember a state is polynomial, we obtain a NPSPACE algorithm.

Co-NP-Hardness comes from a reduction from the Hamiltonian path problem. Indeed, G is not finitely generated iff one can find a path in $\mathcal{A}^{gen}(G)$ with more than $|\mathcal{A}^{gen}(G)|$ transitions. A graph of size n contains no Hamiltonian path if every path of size n contains at least twice th same node. We do not give here the complete reduction, that can be found in an extended version of [59].

3.4 From finitely generated safe cHMSCs to HMSCs

For simplicity, we will assume that transitions of G are labeled by single events. This results in no loss of generality, as from a general safe CHMSC, one can always split cMSCs to obtain a larger CHMC labeled by CMSCs with only one event.

We first describe intuitively the construction. Let Gen(G) be the finite set of atoms of the safe cHMSC G, and *maxsize* be the size of the largest atom of Gen(G). Let b be the bound on unmatched sends on paths of G. The HMSC H will have states labeled by the atomic MSCs in Gen(G). In addition, we will label each state with some information concerning paths of G that can correspond to the sequence of atoms read so far in H. This additional information consists of a sequence of segments of a path (i.e. a set of consecutive transitions) of G (i.e. a *sub-path*), that match this sequence of atoms. That is, each time we read an atom $A \in$ Gen(G), we guess new segments of the path of G that correspond to the MSC A. We keep track of path segments by recording only the first/last node of each segment and the processes occurring in the segment. Hence, all we need to ensure finiteness of H is to ensure that the number of segments is bounded (which is shown in the claim below).

Let us define H more formally. A segment is a triple (s, f, P), where s, f are nodes of G and $P \subseteq \wp$ is a set of processes such that there exists a path $\rho_{s,f}$ from s to f in G, labeled by a CMSC over a set of events located on processes P. If

two segments (s, f, P) and (s', f', P') are consecutive, that is $f = s'$, then their concatenation is a new segment $(s, f', P \cup P')$. Let $a = (s, f, P)$ and $b = (s', f', P')$ be two non-consecutive segments. Then the insertion of a segment $c = (x, y, Q)$ between a and b is allowed iff there exists a path from f to s that passes through x and y in G, and $Q \cap P' = \emptyset$. The result of the insertion is either a path $a.b.c$ if $f \neq x$ and $y \neq s'$, a path $d.c$, where $d = ab$ is the concatenation of a and b if $f = x$ and $y \neq s'$,a path $a.d'$, where $d' = bc$ is the concatenation of b and c if $f \neq x$ and $y = s'$, or a segment $(s, f', P \cup Q \cup P')$ otherwise.

Let $Path$ be the set of sub-paths of G consisting of at most $(b+1) \cdot \text{maxsize}$ segments. The set of nodes of H is $V = \text{Gen}(G) \times \text{Path}$. A node $(A, \rho) \in V$ is labeled by A.

Moreover, there is an edge in H from a node $(A, \rho) = (s_1, f_1, P_1) \dots (s_k, f_k, P_k)$ to $(A', \rho') = (s'_1, f'_1, P'_1) \dots (s'_{k'}, f'_{k'}, P'_{k'})$ iff one can find a path $\xi = (s''_1, f''_1, P''_1) \dots (s''_q, f''_q, P''_q)$ labeled by A', and such that :

- $f_1 = s''_1$ and $(s'_1, f'_1, P'_1) = (s_1, f_1, P_1).(s''_1, f''_1, P''_1)$
- for every $i \in 2..k$, $P_i \cap P''_1 = \emptyset$
- $(s'_2, f'_2, P'_2) \dots (s'_{k'}, f'_{k'}, P'_{k'})$ is a path obtained by successively inserting segments of ξ in $(s_2, f_2, P_2) \dots (s_k, f_k, P_k)$. Each new insertion of a segment (x, y, Q) at index i is performed if $\bigcup_{j \in i+1..k'} P_j \cap Q = \emptyset$ such that places . Note that this insertion is a guess on how atoms should be interleaved, and all guesses have to be proved correct by eventually reaching an accepting state.

The initial node is $(\emptyset, \varepsilon)$, and the final nodes (A, ρ) are those where ρ is a one-segment, accepting path of G. If one can go from the initial node to an accepting one by reading atoms, guessing subpath, and eventually reaching a single complete path from the initial node of G to one of its accepting nodes, then obviously, the guess was correct.

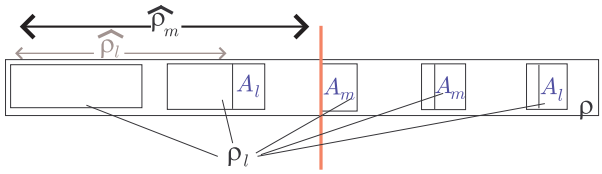
Proof of Theorem 31 Let $M \in \mathbb{L}(H)$, then there exists an initial path of G labeled by M. Conversely, let us consider an MSC $M = A_1 \cdots A_n$, where each A_i belongs to $\text{Gen}(G)$. This MSC labels an accepting path $\rho = v_1 \to \cdots \to v_k$ of G.

For simplicity, we extend the visual order of M to the atoms A_i by letting $A_i \leq A_j$ if there are some events e in A_i, f in A_j with $e < f$. Now, we will assume w.l.o.g. that for every $i < j$ such that $A_i \not\leq A_j$ the first event of A_i in ρ comes before the first event of A_j in ρ. That is, we choose an ordering of the atoms of M according to their first occurrence in ρ.

Claim: Let $l \in \{1, \dots, n\}$ and let ρ_l be the sequence of segments of ρ labeled by $A_1 \cdots A_l$. Then ρ_l consists of at most $(b+1) \cdot \text{maxsize}$ segments.

proof of the claim: We denote by $\widehat{\rho_j}$ the longest prefix of ρ that contains no event of A_j. Let also m be such that $\widehat{\rho_m}$ is the longest prefix $\widehat{\rho_j}$ with $j \leq l$.

The pMSC labeling $\widehat{\rho_m}$ has at most b unmatched sends, among which b' sends belong to ρ_l. Thus, $b'' = b - b'$ is the number of unmatched sends in $\widehat{\rho_m} \setminus \rho_l$ (the difference of two paths is obtained by deleting the nodes of the second path from the first one).



We claim first that there is no complete atomic MSC A_p in $\widehat{\rho_m} \setminus \rho_l$. To see this, note first that such a complete MSC A_p must satisfy $p > l$. However, it can satisfy neither ($A_p \not\leq A_m$ and $A_m \not\leq A_p$), by the choice of the ordering $A_1 \cdots A_n$, nor $A_m < A_p$, since $\widehat{\rho_m}$ has an empty intersection with A_m. Since each incomplete atom of $\widehat{\rho_m} \setminus \rho_l$ contributes with at least one unmatched send in $\widehat{\rho_m}$, we obtain that there are at most $b'' \cdot (\text{maxsize} - 1)$ events in $\widehat{\rho_m} \setminus \rho_l$, thus at most $b'' \cdot (\text{maxsize} - 1) + 1$ segments in $\rho_l \cap \widehat{\rho_m}$.

Moreover, by definition of m there is just one new atom starting in $\rho_l \setminus \widehat{\rho_m}$, namely A_m. Hence there are at most $b' + 1$ different atoms in $\rho_l \setminus \widehat{\rho_m}$ (b' that started already in $\widehat{\rho_m}$ plus A_m). This yields at most $(b' + 1) \cdot \text{maxsize}$ events in $\rho_l \setminus \widehat{\rho_m}$, hence at most $(b' + 1) \cdot \text{maxsize}$ segments. Therefore, we conclude that ρ_l contains at most $(b' + b'' + 1) \cdot \text{maxsize} = (b + 1) \cdot \text{maxsize}$ segments.

Concerning the size of H, for each path segment it suffices to remember the first/last node, and the processes that occurred in the segment. This gives at most $(|G|^2 \cdot 2^{\wp})^{b \cdot \text{maxsize}} = 2^{2b(\log(|G|)+\wp)\text{maxsize}}$ paths consisting of less than $b \cdot \text{maxsize}$ segments.

Let us illustrate the construction of a HMSC from a CHMSC with finitely generated projections. Consider the CHMSC of Figure 12.5. This CHMSC is defined over two processes $\{p, q\}$, and contains 4 nodes n_0, n_1, n_2, n_3, and transitions labeled by two MSCS M_1 and M_2, which describe respectively process A sending message m to process B, and process B receiving message m from process A. Transitions of the CHMSC are (n_0, M_1, n_1), (n_1, M_2, n_3), (n_1, M_2, n_2) and (n_2, M_1, n_1). Let M be the atom consisting of the message m exchanged between A and B. The set of atoms of G is $\{M\}$, hence each state of H is labeled by M (or $\emptyset$). Let us describe the nodes of the HMSC obtained following our construction.

- s_0 is the initial node $(\varepsilon, \emptyset)$
- s_1 is the node $([n0, n_1, \{A\}][n_2, n_3, \{B\}])$
- s_2 $([n0, n_1, \{A\}][n_3, n_4, \{B\}])$. Actually, this is a bad choice, as any path from n_1 to n_3 needs to include an occurrence of CMSC M_2, and hence can not be used to extend this path, as process B appears in the second segment.
- s_3 is the node $[n_0, n_3, \{A, B\}][n_2, n3, \{B\}]$. This node is obtained by assembling path in s_1 with path $([n_1, n_2, \{A\}]n_2, n_3, \{A\}])$ (which creates a complete path fron n_0 to n_3.
- s_4 is the accepting state $([n_0, n_4, \{A, B\})])$. This node can be obtained from s_1 or from s_3 by inserting the path $([n_1, n2, \{A\}], [n3, n_4, \{B\}])$.
- Last, there is a transition from s_3 to itself, as insertion of $([n_3, n_2, \{A\}][n_2, n_3, \{B\}])$ in s_3 gives s_3

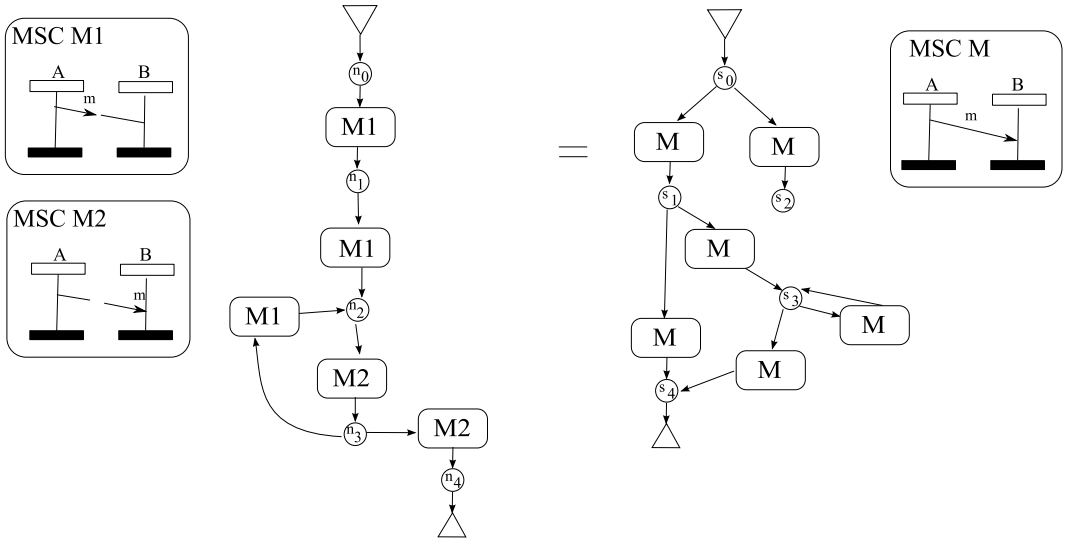


Figure 12.5: Building a HMSC from a finitely generated safe CHMSC

3.5 Proof of Proposition 5

Proof: We know that for any MSC X, $\mathcal{L}in(\mathcal{X})$ is an equivalence class, hence $\mathcal{L}in(\mathcal{X}_1 \parallel \mathcal{X}_2) = [\mathcal{L}in(\mathcal{X}_1) \parallel \mathcal{L}in(\mathcal{X}_2)]$ is immediate.

$\mathcal{L}in(\mathcal{X}_1) \parallel \mathcal{L}in(\mathcal{X}_2) \subseteq \mathcal{L}in(\mathcal{X}_1 \parallel \mathcal{X}_2)$ also follows from the properties of $\mathcal{L}in(\mathcal{X})$, because $\mathcal{X}_1 \parallel \mathcal{X}_2 = Msc\,(\mathcal{L}in(\mathcal{X}_1) \parallel \mathcal{L}in(\mathcal{X}_2))$. Now let $w \in \mathcal{L}in(X)$ and $X \in Msc\,(\mathcal{L}in(X_1) \parallel \mathcal{L}in(X_2))$ for some $X_i \in \mathcal{X}_i$ $(i = 1, 2)$. Again, using the properties of $\mathcal{L}in(\mathcal{X})$, we know that $X = Msc\,(w)$. Therefore, $w \in \mathcal{L}in(X_1) \parallel \mathcal{L}in(X_2)$ by lemma 5 (see below). □

Lemma 5 *$\mathcal{L}in(X_1) \parallel \mathcal{L}in(X_2)$ is closed under the equivalence $\equiv$ (see Def. 49).*

Proof: Let $w \in \mathcal{L}in(X_1) \parallel \mathcal{L}in(X_2)$. We want to show that for any w' in $\mathcal{L}in$ (Def. 48), if $Msc\,(w)$ and $Msc\,(w')$ are isomorphic, then $w' \in \mathcal{L}in(X_1) \parallel \mathcal{L}in(X_2)$. Let $w = \varepsilon_1 \ldots \varepsilon_n$. From Def. 48, $w' = \varepsilon'_1 \ldots \varepsilon'_n$ and there exists a bijection $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $\varepsilon_i = \varepsilon'_{f(i)}$. For $j = 1, 2$ let $E^j = \{i \,|\, 1 \leq i \leq n \wedge \varepsilon_i \in \mathcal{E}^j\}$ and $E'^j = \{i \,|\, 1 \leq i \leq n \wedge \varepsilon'_i \in \mathcal{E}^j\}$, then f restricts and co-restricts to bijections $f_j : E^j \to E'^j$, hence $Msc\,(\pi_j(w))$ and $Msc\,(\pi_j(w'))$ are isomorphic for $j = 1, 2$ (where $\pi_j(w)$ and $\pi_j(w')$ are the respective projections of w and w' on $\mathcal{E}^{j*}$). Therefore, $\pi_j(w') \in \mathcal{L}in(X_j)$ for $j = 1, 2$ and $w' \in \mathcal{L}in(X_1) \parallel \mathcal{L}in(X_2)$. □

3.6 Proof of proposition 6

Proposition 6 is an immediate corollary of proposition 5. Indeed, $\mathcal{L}in^b(\mathcal{X}_1 \parallel \mathcal{X}_2) = \{w \in \mathcal{L}in(\mathcal{X}_1) \parallel \mathcal{L}in(\mathcal{X}_2) \mid w \text{ is b-bounded}$. We know that w is b-bounded iff the projections $\pi_1(w)$ and $\pi_2(w)$ are b-bounded. Hence $\mathcal{L}in^b(\mathcal{X}_1 \parallel \mathcal{X}_2) = \{w \in \mathcal{L}in^b(\mathcal{X}_1) \parallel \mathcal{L}in^b(\mathcal{X}_2) \mid w \text{ is b-bounded}$

3.7 Proof of Proposition 34

Proof: We show that the Post correspondence problem may be reduced to the above decision problem. Given two finite lists of words $u_1, \ldots, u_k$ and $w_1, \ldots, w_k$ on some alphabet Σ with at least two symbols, the problem is to decide whether $u_{i_1} u_{i_2} \ldots u_{i_t} = w_{i_1} w_{i_2} \ldots w_{i_t}$ for some non empty sequence of indices i_j. This problem is known to be undecidable for $k > 7$. Given an instance of the Post correspondence problem, *i.e.* two lists of words $u_1, \ldots, u_k$ and $w_1, \ldots, w_k$ on Σ, consider the two

HMSCs $G_1 = (= N, \rightarrow, \mathcal{M}_1, n_0, n_f)$ $G_2 = (N, \rightarrow, \mathcal{M}_2, n_0, n_f)$, with the same underlying graph $(N, \rightarrow, n_0, n_f)$, constructed as follows (G_1 is partially shown in Figure 12.6).

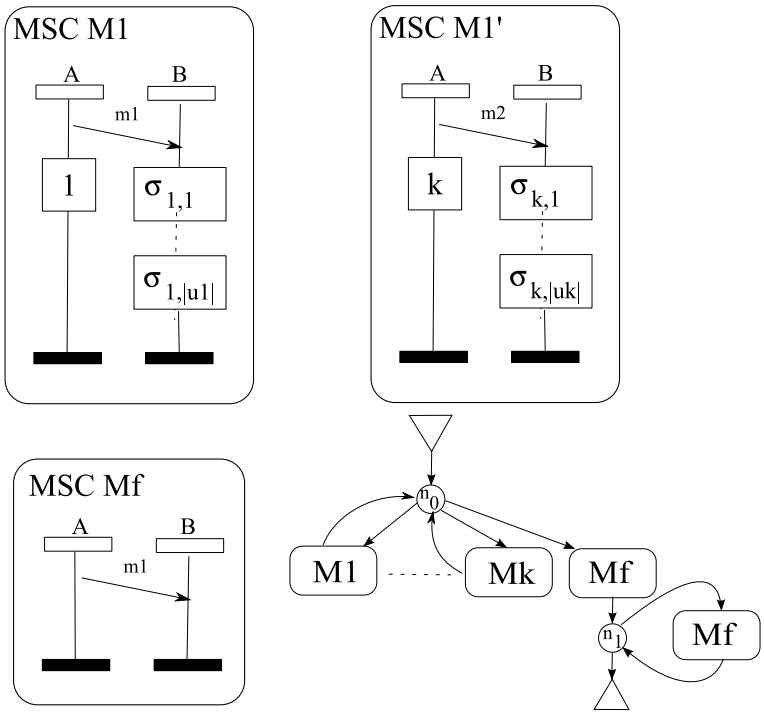


Figure 12.6: Encoding a PCP with a mixed product.

Define $N = \{n_0, n_f\}$, $\mathcal{M}_1 = \{M_i, i \in 1..k\} \cup M_f$, $\mathcal{M}_2 = \{M'_i, i \in 1..k\} \cup \{M'_f\}$. For each $i \in 1..k$ we define M_i as the MSC that starts with a message m_1 from process p to process q. Furthermore, M_i contains an action $p(i)$ located on process p following the sending of m_1, that represent index i of a pair of words (u_i, w_i). Last, M_i contains a sequence of $|u_i|$ atomic actions occurring after reception of m_1 on process q, respectively labeled by $q(\sigma_{i,1}), \ldots, q(\sigma_{i,|u_i|})$. This sequence of actions represents $u_i = \sigma_{i,1}\, \sigma_{i,2} \ldots \sigma_{i,|u_i|}$. M_f is a MSC containing a message m_1 from A to B.

Similarly, we define for each $i \in 1..k$ MSC M'_i as the MSC that starts with a message m_2 from process p to process q. Furthermore, M'_i contains an action $p(i)$ located on process p following the sending of m_2, and a sequence of $|w_i|$ atomic actions occurring after reception of m_2 on process q, respectively labeled by $q(\sigma'_{i,1}), \ldots, q(\sigma'_{i,|w_i|})$ representing $w_i = \sigma'_{i,1}\, \sigma'_{i,2} \ldots \sigma'_{i,|w_i|}$. M'_f is a MSC containing a message m_2 from A to B.

We then define M_f as the MSC that contains a single message m_1 from p to q, and M'_f as the MSC that contains a message m_2 from q to p. We define $\longrightarrow_1 = \{(n_0, M_i \rightarrow n_0) \mid M_i \in \mathcal{M}_1\} \cup \{n_0, M_f, n_f\} \cup \{n_f, M_f, n_f\}$ and $\longrightarrow_2 = \{(n_0, M'_i \rightarrow n_0) \mid M'_i \in \mathcal{M}_2\} \cup \{n_0, M'_f, n_f\} \cup \{n_f, M'_f, n_f\}$.

For $i = 1, 2$ let $\mathcal{X}_i = \mathcal{L}(G_i)$, then $\mathcal{L}in^1(\mathcal{X}_i)$ is a regular representative set for $\mathcal{X}_i$. If the Post correspondence problem has no solution, then $\mathcal{X}_1 \parallel \mathcal{X}_2$ is empty, hence it is existentially bounded. In the converse case, $\mathcal{X}_1 \parallel \mathcal{X}_2$ contains for all B some MSC including a crossing of B messages m'_1 by B messages m'_2, hence it is not existentially bounded. □

3.8 Proof sketch for propositions 7 and 8

Proof Sketch: We use a graph representation of product of MSCs in $\mathcal{L}(G_1) \,|||\, \mathcal{L}(G_2)$. A MSC X in the product can be seen as an interleaving of two MSC X_1, X_2 respectively in $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$, where common events in X_1 and X_2 are merged. For X, we build a graph $(V, \rightarrow_{|||})$ which vertices are events of X_1 and X_2, and which edges are pairs (e, f) such that :

- e, f represent a common event in X_1, X_2, or
- $e <_p f$ in X_1, or
- $e <_p f$ in X_2, or
- $e <_p f$ in X.

The only cycles in this graph should be pairs of common events, as X is an MSC, so X is not existentially b-bounded if adding the Rev_B relation (see chapter 1, proposition 1) creates a cycle. Then one can show that if a cycle exists, then there is a cycle that contains at most 2 events on each process. Then, we can show that there exists a bound K such that if two path ρ_1 and ρ_2 of G_1 and G_2 producing an existentially b-bounded MSC X while "synchronizing"' more than K times on common events, then one can repeat (or remove) a subset of transitions from ρ_2 to obtain a new path ρ'_2 such that $M_{\rho_1} \,|||\, M_{\rho'_2}$ is also a b-bounded MSC of the product. Hence, it shows that if a cycle using Rev_K relation exists for a sufficiently large bound K, appearing in pairs of sufficiently long path to allow such Rev_K edges, then there exists a cycle for a bound that does not exceed B, and smaller witness paths.

Proposition 8 is proved by showing that it is sufficient to build an automaton which states remember:

- the last node of a pair of path ρ_1, ρ_2 from G_1, G_2, that agree on their common events.
- A set E of events randomly selected to become witnesses of an existing cycle in the graph representing the shuffled path plus the Rev_B relation.
- The restriction on E of the orderings and Rev_B relations.

Transitions of this automaton select a transition from G_1, or from G_2, or from both if the two transitions share the same common event (recall that we assume safe CHMSCs which transitions are labeled by single events. States are updated accordingly, and randomly choose to remember the last event (it must not be a common event, and at most 2 events per process need to be remembered). Final states are states for which the graph $(E, < \cup Rev_B)$ contains a cycle. Obviously, if the language accepted by this automaton is empty, then $G_1 \,|||\, G_2$ is B-bounded. co-NP-completeness comes from a reduction from 3-CNF-SAT, close to the reduction of chapter 1 used to check regularity, or global cooperation.

3.9 Proof of proposition 9

We can see the construction of a MSC X in $L(X_1 \,|||\, X_2)$ as the union of ordering relations $\leq_1$ and $\leq_2$, plus a choice of a linearization $\leq_{mp}$ of events on the monitor process such that $\leq_1 \cap E^2_{mp} \subseteq <_{mp}$ and $\leq_2 \cap E^2_{mp} \subseteq <_{mp}$. This construction can also be seen as a graph $G = (E, V)$, where $E = E_1 \cup E_2$ and $V = <_1 \cup <_2 \cup <_{mp}$. X is a MSC if and only if G is acyclic. Let us suppose that for a choice of $<_m p$, there exists a cycle $\rho = e_1 \dots e_n$ in G (i.e. X is not an MSC). Clearly, this cycle is not contained in $<_1$ nor in $<_2$, nor in $<_{mp}$. As E_1 and E_2 are located on disjoint instances, except for the monitor process, there exists a path from an event $a \in E_1$ to an event $b_i n E_2$ if and only if there exists an event located on mp appearing in this path. Hence, the cycle can be decomposed as follows $\rho = \alpha_1.\beta_1.\alpha_1 \dots \beta_k$, where α_1, β_k can be the empty word, elements of α_i's belong to X_1 and elements of β_i's belong to X_2. Without loss of generality, we will consider that α_1 is not empty. Each α_i is of the form $e_{s(i)} \dots e_{n(i)}$, and for $i \in 2..k-1$, $e_{s(i)}$ and $e_{n(i)}$ are located on the monitor process. For every pair e, f of consecutive events in α_i, we have $e <_1 f$. We also have $e_{n(1)}$ located on mp and $e_{n(k)}$ is also necessarily located on mp if $\beta_k \neq \varepsilon$. Similarly, we have $\beta_i = f_{s(i)} \dots f_{n(i)}$, and for $i \in 1..k-1$, $f_{s(i)}$ and $f_{n(i)}$ are located on the monitor process. Let us suppose β_k is empty. As ρ is a cycle, we have $(e_n(k), e_s(1)) \in V$, that is, $(e_n(k) <_1 e_s(1))$. We also have $e_{s(1)} <_1 e_{n(1)} <_1 e_{s(2)} <_1 \dots <_1 e_{n(k)}$, hence $<_1$ contains a cycle, which is a contradiction. If $\beta_k \neq \varepsilon$, then $f_{n(k)}$ or $e_s(1)$ is on process mp, as any path from an event of E_1 to and event of E_2 passe through mp, and conversely. Now as ρ is a cycle, then either $f_{n(k)} <_{mp} e_s(1)$, but then $<_{mp}$ is cyclic (a contradiction) $f_{n(k)} <_1 e_s(1)$ i.e. $f_{n(k)}$ is a shared event, but then $<_{mp}$ is cyclic again. The last possibility is if $f_{n(k)} <_2 e_s(1)$, i.e. $e_s(1)$ is a shared event. But then, one can show that $<_2$ becomes cyclic. □

4 Proofs for chapter 6

4.1 Proof sketch for Theorem 39

Proof sketch A DMG G is a grammar. For every legal derivations of G (that is a derivation that leads to the production of a dMSC), one can compute a derivation tree that explains how non-terminals were successively replaced to achieve this legal production. The language of derivation trees of a grammar G is a recognizable tree language, and can be recognized by a bottom-up tree automaton $\mathcal{A}_G$. We then have $\mathcal{L}(\mathcal{A}_G)) = \emptyset$ iff $\mathcal{L}(G) = \emptyset$. Let $G = (\Pi, \mathcal{N}, S, \longrightarrow)$. The tree automaton that recognizes legal derivation trees of G is $\mathcal{A}_G = (Q, Q_F, \mathcal{F}, \delta)$, where $Q = \{q_M \mid M \in \mathcal{M}\} \cup \{(q_N, C) \mid N \in \mathcal{N}\}$ is a set of states, $Q_F = \{(C_0, q_a x, C)\}$ is a final state. In (q_N, C), C is a communication structure, that details how processes identifiers shall migrate when replacing non terminal N.

Such communication structure can be represented as a function from Π to $\Pi \cup \{\perp\}$. Intuitively (π_1, π_2) means that the identifier π_1 migrates to the process known as π_2, and $(\pi_1, \perp)$ means that the identifier π_1 migrates to a process created during rewriting of the non-terminal. In an expresion, *expr* if one attaches a commucication structure to all non-terminal, one can check if the replacement of a non-terminal by *expr* produces results in process identifiers migration declared in a communication structure C. We can also check if the expression can be associated a valid MSC only by looking at terminal MSCs and communication structures labeling non-terminals. There can be up to $|\Pi| + 1^{|\Pi|}$ such structures.

The transition relation δ is built as follows. All leaves of the tree are labeled by states (q_M), where M is the MSC recognized at that leave. Other nodes of the tree are labeled by states (q_N, C). The transition relation is of the form $(q_N, C) \leftarrow q_1, \ldots q_k$. Each q_i represents a recognized terminal or non-terminal and its associated communication structure, that follows the ordering of terminals and non-terminals in some rule rewriting N. Furthermore, the sequence $q_1, \ldots q_k$ shall contains sates (i.e refer to MSCs and communication structures) that guarantee that a valid MSC is produced by this rewriting. Last, the communication pattern realized by MSCs and communicatiàn structures in this sequence of states should be C.

The construction of the tree automaton leads to an exponential blowup is the size of the initial grammar. Then checking emptiness of $\mathcal{A}_G$ can be done in $O(|Q|.(|Q|.\delta))$. □

5 Proofs for chapter 7

This appendix provides a proof for theorem 43, that is we want to show that the original specification given as a HMSC and the synthesized controlled machines exhibit the same behaviors. We proceed in several steps. We first show (lemma 6 that in the synthesized machines, all choices (i.e. events corresponding to the first event of some bMSC) are causally ordered in any execution. We then show (Lemma 7) that for every configuration of a HMSC H reachable after an execution, there exists a finite set of configurations of the synthesized machines reachable by observing the same execution. The last steps show inclusion of specification and implementations languages in both directions by contradiction. Supposing that one can reach a configuration (after executing a prefix O), where H allows firing of an event a but not corresponding configuration of the CFSM allow a leads to a contradiction for all types of events. We consider each type of events and show that the allowing a in one language but not in the other contradicts either the fact that O is a prefix of both the original specification and of the synthesized language, or the fact that choices are ordered.

Let us first define formally how controllers events can be replaced by silent transitions, and how communications are renamed.

Definition 92 *Let $O = (E, \leq, t, \varphi, m)$ be a prefix in $\mathcal{L}(\parallel_{i\in I} K(A_i)|Ci)$. The* restriction *of O to non-control events is a restriction of O to events located on $K(A_i)$'s. We will denote this restriction by $Unc(O)$. The* uncontrolling *of $O = (E, \leq, t, \varphi, m)$ is a renaming function $Ru()$ that replaces communications to and from the controller of a process by direct communications with the process concerned by the sent/received message, and builds the message mapping. $Ru(O) = (E, \leq, t', \varphi, m')$, where $t'(e) = p!q(m)$ if $t(e) = K(A_p)!C_p(m, q, c)$, $t'(e) = p?q(m)$ if $t(e) = K(A_p)?C_p(m, q)$, and $t'(e) = t(e)$ otherwise. Function m' maps the i^{th} sending from p to q with the i^{th} reception on q from p for every pair of processes.*

Note that for a prefix O in $\mathcal{L}(K(A_i)|Ci)$ (i.e. located on a single instance), the message mapping in $Unc(O)$ is an empty relation.

One of the first things to show that all choices in the synthesized machines are causally ordered. Once this property is demonstrated, the rest of the correctness proof is not surprising, as controllers simply enforce correct message receptions using the total order imposed by choice events.

Lemma 6 *For each local HMSC H, the choices events in any behavior of the synthesized communicating machines are totally ordered.*

Proof: We prove this property by induction. Let us denote by P_n the property: For all H, local HMSC, the choices in any behavior of the synthesized communicating machines in a run containing n choices are totally ordered.

Let us first verify this property for $n = 2$. As H is local, then there is only one CFSM that can perform an action (a message sending) from the initial configuration. The next choice can then only be performed after the first one. Hence, the first two choices are ordered.

Let us suppose that the property is verified up to n, and prove that it also holds for $n+1$. Let us suppose a prefix $O \circ O'$ from $\mathcal{L}(\| K(A_i)|C_i)$ with $n+1$ choices, such that O contains n choices. Then, O is of the form $O = \{c_1\} \circ O_1 \ldots \{c_n\} \circ O_n$, where each c_i is a choice event, and such that $\{c_1\} \circ O_1$ is an execution of a prefix of the first bMSC M_1 appearing in this run. Then, $O \circ O'$ can be completed by piece P_1 such that $O \circ O' \circ P_1$ contains a complete execution of the first bMSC M_1 by the controlled CFSM (so far, nothing forces M_1 to be completely executed in $O \circ O'$).

We can now use a nice property of FIFO bMSCs: every bMSC M can be represented by one of its linearizations. Hence, knowing the respective ordering of actions on each process is sufficient to draw a bMSC. Let us now consider any bMSC of the form $M = P \circ P_a \circ P_b \circ P'$, where P, P' are pieces of bMSC, P_a and P_b are pieces containing only actions a and b, respectively. Then, if a and b are located on distinct instances, then M can also be written as $M = P \circ P_b \circ P_a \circ P'$. This property also applies to pieces of bMSCs, and also to CFSM executions, which can be seen as bMSC pieces.

In the behavior $O \circ O' \circ P_1$, all actions of P_1 are concurrent with actions from $\{c_2\} \ldots O_n \circ O'$, as otherwise at least one action in do not need to wait for the execution of an event in P_1 to be fireable, and $O \circ O'$ would not be an execution of our CFSM.

So, $O \circ O' \circ P_1$ can be equivalently rewritten as $O \circ O_{1,1} \circ \ldots O_{1,n} \circ P_1 \circ \{c_2\} \circ O'_2 \ldots \{c_n\} \circ O'_n \circ O'$, where each $O_{1,i}$ is the part of O_i that belongs to M_1 and $O'_i = O_i \setminus O_{1,i}$. Note that P_1 is ensured to be a legal continuation of $O \circ O'$ as no machine can start executing events with tags greater than $0^{\mathcal{B}_H}$ before executing all its tasks in M_1. This also means that one does not have to change the tag of messages appearing in $P1$ to rewrite $O \circ O' \circ P_1$ into $O \circ O_{1,1} \circ \ldots O_{1,n} \circ P_1 \circ \{c_2\} \circ O'_2 \ldots \{c_n\} \circ O'_n \circ O'$ (all messages between controllers in $P1$ will be tagged by a vector associating 0 to all branches except the branch labeled by $M1$ in H).

Let us denote by $P_{2,n} = \{c_2\} \ldots \{c_n\} \circ O'_n \circ O'$ the tagged piece starting at choice event c_2, and by $P'_{2,n}$ the same piece, where all tags are decremented on component M_1. $P'_{2,n}$ is a run with n choices of an HMSC H', which is a copy of H where the initial node is the node reached in H after M_1. Hence, all choices in $P'_{2,n}$ are ordered, and so are choices in w'. Hence, all choices in $O \circ O'$ are totally ordered. □

As choice events are the only moment when a tag is updated, this lemma also means that the set of tags that can appear in an execution is the set of tags labeling choice events, and hence that the tags produced in any run that belongs both to $\mathcal{L}(H)$ and $\mathcal{L}(\| K(A_i)|C_i)$ are the same.

A *configuration* of an HMSC H is an element of $\mathcal{L}(H)$ (i.e. a bMSC piece). Note that each configuration in $\mathcal{L}(H)$ is a prefix of a bMSC generated by a unique minimal path ρ_P of H, as choice events uniquely designate chosen branches (this is ensured by our restrictions). We will say that an action a is fireable from a configuration P of H iff $P \circ \{a\} \in \mathcal{L}(H)$ (where $\{a\}$ is the bMSC piece that contains only action a. This means that either $P \circ \{a\}$ is a prefix of O_{ρ_P}, or that there exists a path $\rho' = \rho_P.(n, M, n')$ such that $P \circ \{a\}$ is a prefix of $O_{\rho'}$.

We can now show that for every prefix O that belongs to the language of H and to the language of the synthesized machines, one can find a finite sets of executions of the controlled architecture that are equivalent to O after renaming and erasing of controllers' events.

Lemma 7 *Let $O \in Ru(Unc(\mathcal{L}(\| K(A_i)|C_i))) \cap \mathcal{L}(H)$ be an execution. Then, there exists a finite set of executions $X = \{O_1, \ldots, O_k\}$ of $(\| K(A_i)|C_i)$ such that $Ru(Unc(X)) = \{O\}$.*

Proof: The events of the controlled automata in executions of the CFSM can be obtained from O by replacing every action on a process p by an action labeled by RU^{-1} in the CFSM execution (for instance $p!q(m)$ becomes $p!C_p(q, m, b)$ for some branch b). The behavior on each controller simply consists in receiving messages from the automaton it controls, and forwarding them to the next controller, or conversely receiving messages from a controller and forwarding them to the automaton they control in the order specified by the branches. Then, every complete message from p to q in O can be mapped to a sequence of 3 messages that "simulate" the sending from a process p to a process q. So, if O has no unreceived message, then all automata in $(\| K(A_i)|C_i)$ are in a configuration with empty communication buffers, and each automaton and controller can only be in one state. Now if there is at least one message m sent from p to q in O but not received, then this means that $(\| K(A_i)|C_i)$ is in a configuration where a message (q, m, b) can be transiting between p and C_p, a message (m, τ) can be transiting between C_p and C_q, or last a message (p, m, b) can be transiting between C_q and q. Figure 12.7-a) shows an execution of some HMSC in which a message $m3$ is sent but not yet received. There can be three configurations corresponding to such situation, and Figure 12.7-b) shows one of them in which a message of type $m3$ is transiting between the controllers of B and A. Hence, the number of configurations in which CFSMs can be while observing $O \in Ru(Unc(\mathcal{L}(\| K(A_i)|C_i))) \cap \mathcal{L}(H)$ and the size of X are finite and depend on the number of unreceived messages.□

From this lemma, one can also deduce that there exists a correspondence between each configuration reachable in the semantics of H and a finite set of configurations of the synthesized machines.

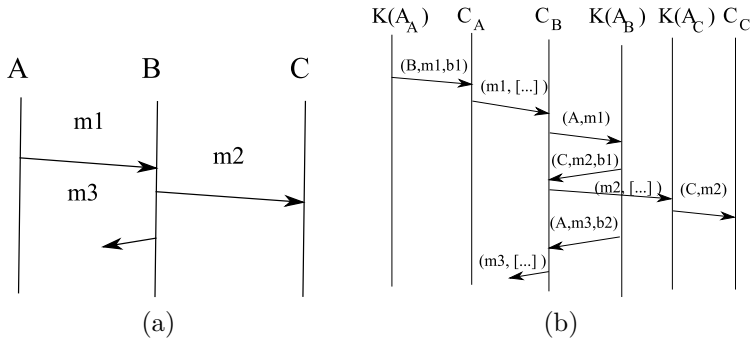


Figure 12.7: Relating HMSCs executions and CFSMs

We are now ready to prove language equality, by showing two inclusions.

Lemma 8 *Let H be a HMSC, $\{A_i\}_{i\in I}$ and $\{A_i\}_{i\in I}$ be respectively the projection of H on its instances, and the synthesized controllers. Then, $Ru(Unc(\mathcal{L}(\| K(A_i)|C_i))) \subseteq \mathcal{L}(H)$*

Proof: For short, we write $\mathcal{L}_1 \subseteq \mathcal{L}_2$ instead of $Ru(Unc(\mathcal{L}(\| K(A_i)|C_i))) \subseteq \mathcal{L}(H)$.

Suppose that there exists a prefix $O \circ \{a\} \in \mathcal{L}_1$ such that $O \in \mathcal{L}_2$, but $O \circ \{a\} \notin \mathcal{L}_2$. O is a configuration of H, and as $O \in \mathcal{L}_1$, there exists a set X_O of possible executions of the synthesized CFSM such that $RU(Unc(X_O))$ (from lemma 7). There also exists at least one execution $O_i \in X_O$ such that after executing O_i, the CFSM is in a configuration C_A in which automaton $K(A_p)$ is in a state allowing firing of a transition (s, σ, s') with $Ru(\sigma) = a$.

Suppose that a is a sending event from p to q, i.e. $a = p!q(m)$ for some m, and $\sigma = K(a_p)!C_p(m, b)$ for some branch b. The sequence of events in O on p and in O_i on $K(A_p)$ are identical, up to renaming. Hence, this means that p and $K(A_p)$ follow the same path ρ of H until the end of O_i (recall that transitions of projected automata are defined from transitions of H). Then, all predecessors of σ in O_i allowing to reach state s have been executed, and all predecessors of a in O_ρ have been executed too in O. Hence, O is a configuration of H that allows for the firing of action a on process p, as projection preserves (up to renaming due to control) sequences of events on each process. This contradicts the fact that a is a sending event. A similar case holds for atomic actions. Hence, a can only be a receive action, i.e. a is of the form $a = p?q(m)$ for some q, m, and $\sigma = K(A_p)?C_p(q, m)$. This means that O_i is an execution that brings the CFSM in a configuration in which the FIFO queue from C_p to $K(A_p)$ has a message m as head (otherwise (q, σ, q') can not be fired).

As mentioned in lemma 7, messages in $Ru(Unc(O_i))$ are simulated by three messages in O_i. Then, O_i is of the form $O_1 \circ \{\sigma_1\} \circ \{\sigma_2\} \circ \{\sigma_3\} \circ \{\sigma_4\} \circ \{\sigma_5\} \circ O_2$, where O_1 is a prefix and O_2 is a piece. We furthermore have $\sigma_1 = K(A_q)!C_q(p, m, b)$ for some branch b, $\sigma_2 = C_q?K(A_q)(p, m, b)$,$\sigma_3 = C_q!C_p(m, \tau)$, $\sigma_4 = C_p?C_q(m, \tau)$, and $\sigma_5 = C_p!K(A_p)(q, m, b)$. If any of these actions is missing in O_i, then σ can not be fired. Such situation is depicted in Figure 12.8-a).

Let us now consider O as a configuration of H. There is a message m sent from q to p but not yet received in O. Event a is not allowed by H from configuration O, however, message m was sent. Hence, a is forbidden because according to the chosen path in H, there are some events $\alpha_1, \ldots \alpha_k$ to execute on instance p before a (i.e. there is piece of bMSC P_a such that $O \circ \{a\}$ is not a configuration of H, but $O \circ P_a \circ \{a\}$ is). This situation is depicted in Figure 12.8-b).

After execution O_i, the automaton $K(A_p)$ has reached a state s, which means that s is reachable in $K(A_p)$ by reading the controlled version of the actions appearing on p in O (i.e. the sequence $Ru^{-1}(\pi_p(O))$). As there exists a transition by (s, σ, s') in $K(A_p)$ and as we know that $\alpha_1, \ldots \alpha_k$ can be executed by process p after O, then state s is a choice, from which at least two transitions (s, σ, s') and (s, c, s_1), where c is an action of the automata corresponding to α_1 (i.e. $RU(c) = \alpha_1$), can be fired. Note that as all choices in H are local, c is necessarily a message reception event.

Events c and σ belong to different branches of the same choice of H, and we have that $\tau(c) \neq \tau(a)$, as events of P_a located on p **have to** be executed before a. From lemma 6 we know that all choices in an execution of the CFSM are totally ordered. Furthermore the tags associated to an execution of an HMSC and to an execution of the synthesized communicating automata are the same. We then have $\tau(c) < \tau(a)$. As c and a are events of choices that concern p, the communication $\sigma_4 = C_q!C_p(m, \tau = \tau(a), b)$ that must occur in O_i before σ can not be executed by

$K(A_p)$ as the message received by C_p at event σ_4 is tagged by a vector τ which is not the expected successor tag on C_p. Hence C_p can not consume it and forward m to $K(A_p)$, unless it has received and forwarded the messages corresponding to the second branch of H, which does not appear in O. Then, receptions on this branch must be executed by $K(A_p)$ before σ. We then have a contradiction, and $\mathcal{L}_1 \subseteq \mathcal{L}_2$.□

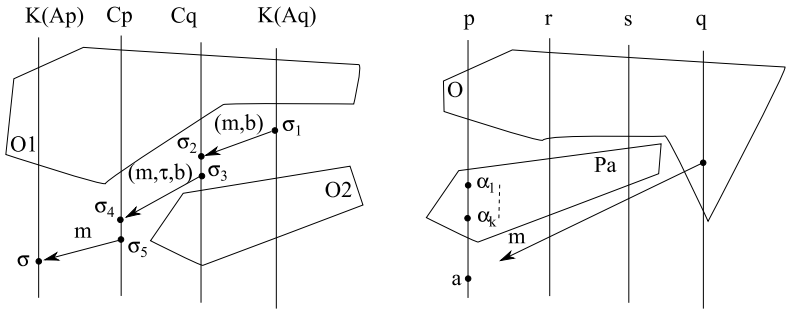


Figure 12.8: Illustration of the proof of Lemma 8

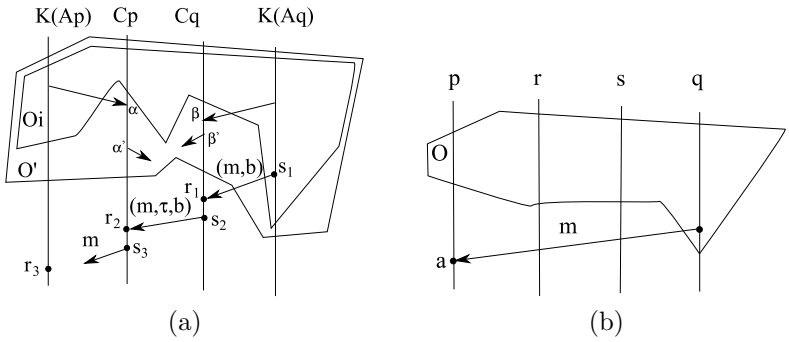


Figure 12.9: Illustration of the proof of Lemma 9

Lemma 9 *Let H be a HMSC, $\{A_i\}_{i\in I}$ and $\{A_i\}_{i\in I}$ be respectively the projection of H on its instances, and the synthesized controllers. Then, $\mathcal{L}(H) \subseteq Ru(Unc(\mathcal{L}(\| K(A_i)|C_i)))$*

Proof: For short, we write $\mathcal{L}_2 \subseteq \mathcal{L}_1$ instead of $\mathcal{L}(H) \subseteq Ru(Unc(\mathcal{L}(\| K(A_i)|C_i)))$.

Let us suppose there exists $O \circ \{a\}$ such that $O \circ \{a\} \in \mathcal{L}_2$, $O \in \mathcal{L}_1$ but $O \circ \{a\} \notin \mathcal{L}_1$. From lemma 7, there exists an execution O_i of the CFSM such that $Ru(Unc(O_i)) = O$. If a is a sending of a message or an atomic action on process p, then $K(A_p)$ must be in a state q from which an transition (q, σ, q') with $RU(\sigma) = a$ is fireable, as the sequence of controlled events corresponding to the projection of O on p is recognized by $K(A_p)$, and as $K(A_p)$ is a deterministic machine. Transition (s, σ, s') is fireable as soon as $K(A_p)$ is in state s, which is the case, and $O_i \circ \{\sigma\}$ is a behavior of the CFSM. So, if a is a sending event or an atomic action, $O \circ \{a\} \in \mathcal{L}_1$, which contradicts the initial hypothesis.

Then, a is a reception $a = p?q(m)$, an O looks like the execution represented in Figure 12.9-b). As shown in lemma 7, a message m from p to q in a configuration of H corresponds to a sequence of three messages in the CFSM execution: $s_1 =$

$K(A_q)!C_q(p,m,b)$, $r_1 = C_q?K(A_q)(p,m,b)$, $s_2 = C_q!C_p(m,\tau)$, $r_2 = C_p?C_q(m,\tau)$, $s_3 = C_p!K(A_p)(q,m,b)$, $r_3 = K(A_p)?C_p(q,m,b)$ (with $Ru(Unc(r_3 = a))$). As the sending of m from q to p appears in execution O, the corresponding sending event s_1 executed by $K(A_q)$ also appears in O_i.

We can now proceed as follows: we first prove that there exists an execution O' of the synthesized CFSM such that $Ru(Unc(O')) = O$, and such that in the configuration reached by the CFSM after O', controller C_p is ready to execute event r_2 if a message is buffered with appropriate tag. We then show that such message exists, and that it is correctly tagged, and hence allows for the reception of r_2, followed by s_3 and r_3.

Execution O is a prefix of a concatenation of bMSCs labeling branches of H, i.e. a sequence of bMSCs $M_1 \circ M_2 \circ \dots M_n$. Among these bMSCs, process p is concerned only by a subset $M_{i1}, \dots M_{ik}$ of them, and process q by another subset $M_{j1}, \dots M_{jk'}$. Sending and reception of message m is O belongs to a bMSC M_{pq} appearing in both sets. From Lemma 7, as $O \in \mathcal{L}_1 \cap \mathcal{L}_2$, there exists an execution O_i of the CFSM such that $Ru(Unc(O_i)) = O$. After O_i, the CFSM is in a configuration in which automaton $K(A_p)$ can fire a transition (s, r_3, s') provided the head of the queue from C_p to $K(A_p)$ is a message m.

One can note that the controller C_k of an automaton $K(A_k)$ systematically receives messages sent by $K(A_k)$ (rule R1) and forwards them to another controller. Similarly, if C_k receives a message from another controller, it forwards it to $K(A_k)$. This means that when A_k receives a message and this reception belongs to a branch b of H, then C_k has necessarily counted this branch in its vectorial clock τ_k, that remembers the number of occurrences of choices concerning k that have occurred so far. This also means that C_k has accepted an incoming message (m, τ) coming from another controller, and that τ was a valid tag at the time of this message reception.

Let us consider again O_i. This execution is a partial execution of $M_1 \circ \dots M_n$ by the CFSM, and contains some elements of M_{pq}, including event s_1. Considering the sequence of events executed by $K(A_p)$ and $K(A_q)$ in O_i, one can also get the sequence of sendings/receptions executed by the controllers C_p and C_q, as for every event of the form $p!q(m)$ in O there exists a pair of events $K(A_p)!C_p(m,b).C_p?K(A_p)(m,b)$, and for every event of the form $p?q(m)$ in O there exists a pair of events $C_p!K(A_p)(i,m,b).K(A_p)?C_p(i,m,b)$ in O_i. However, this does not mean that C_q is ready to execute(or has already executed) r_1 or C_p is ready to execute (or has already executed)r_2 , as some messages may still need to be consumed in the message queues of C_p and C_q. Let us suppose that C_q must receive at least one message, either from another controller, or from $K(A_q)$ before receiving message m. Let us call this reception β and the following retransmission β'. In the first case, β **must** be executed before r_1 if and only if it is a reception of a message that have to be executed to comply with the sequence of receptions defined in some branch of H. In this case, β' is a sending of a message to $K(A_p)$, and as it has to be executed before r_1, then it means that some reception on $K(A_q)$ must be executed before s_1, and then we cannot have $Ru(Unc(O_i)) = O \in \mathcal{L}_1 \cap \mathcal{L}_2$. In the latter case, as reception of messages from controlled automata can be performed without waiting (according to rule R1 of the controllers), then there exists an execution $O_i \circ \{\beta\} \circ \{\beta'\}$ of the CFSM from which r_1 can be executed, and such that $Ru(Unc(O_i \circ \{\beta\} \circ \{\beta'\})) = O$. Similarly, if C_p is in a configuration from which r_3 can not be fired because a reception α followed

by a retransmission of message α' must occur before r_3, then one can show that either this implies that $Ru(Unc(O_i)) \notin \mathcal{L}_1 \cap \mathcal{L}_2$, or that there exists an execution $O_i \circ \{\alpha\} \circ \{\alpha'\}$ such that $Ru(Unc(O_i \circ \{\alpha\} \circ \{\alpha'\})) = O$. Figure 12.9-a) illustrates this situation. The argumentation extends for arbitrary sequences of actions $w_p = \alpha_1.\alpha'_1 \ldots \alpha_i.\alpha'_i$ and $w_q = \beta_1.\beta'_1 \ldots \beta_j.\beta'_j$, $i, j \in \mathbb{N}$ that have to be executed by C_p and C_q before the execution of r_1 and r_2: mandatory reception from a controller implies $Ru(Unc(O_i)) \notin \mathcal{L}_1 \cap \mathcal{L}_2$, and mandatory reception from $K(A_p)$ or $K(A_q)$ can be performed to obtain a larger execution. Note that in the sequences of missing events w_p and w_q C_p and C_q can not be forced to exchange a message, which would imply a reception from another controller, and hence $Ru(Unc(O_i)) \notin \mathcal{L}_1 \cap \mathcal{L}_2$. So events in w_p are independent from events in w_q, and one can find an execution of the CFSM O' that includes O_i, the sequences w_p, w_q, and the two events r_1 and s_2. Hence, after O', controller C_p is in a configuration allowing it to receive the message (m, τ) sent by C_q if τ is a correct tag. This reception corresponds to rule (R2) of the controller. If r_2 can not be executed by C_p but $[\tau_p]_p = [\tau]_p$, then it usually means that r_2 is not the next reception to perform according to the chosen branch, and that there are remaining actions to perform on C_p before allowing r_2. However, we have ruled out this possibility after execution of O'. Hence, the only case remaining is when $[\tau_p]_p \neq [\tau]_p$, and $[\tau]_p$ is not an immediate successor of $[\tau_p]_p$. However, we know that s_1 is a causal consequence of all choices that have been performed in O' up to bMSC M_{pq}, as one can establish a correspondence between messages in O and sequences of messages in O'. So, $\tau[b]$ is exactly the number of occurrences of branch b in $M_1 \circ \cdots \circ M_{pq}$. As C_p has executed all events in w_p required before execution of r_2, that is corresponding to events in $M_1 \circ \cdots \circ M_{pq-1}$ in execution O', and more precisely all receptions of messages coming from other controllers, we necessarily have $[\tau_p]_p[b'] = [\tau]_p[b']$ for every branch $b' \neq b$ of H, and $[\tau_p]_p[b] + 1 = [\tau]_p[b]$. This contradicts the fact that r_2, necessarily followed by s_3 and r_3 can not be executed from O', and hence contradicts $O \circ \{a\} \notin \mathcal{L}_1$. □

Theorem 43 *Let H be an HMSC, and let $\mathcal{A}^{cont} = \|_{i \in I} K(A_i)|C_i$ be its controlled synthesis. Then, $\mathcal{A}^{cont}$ simulates H (up to renaming).*

Proof: The proof of this theorem is is straightforward, as we clearly have a relation between configuration and ations of both models using (lemmas 8 and 9).

5.1 Proof of theorem 44 (well formedness of DMGs

Theorem 44 *For a DMG G, one can decide in doubly exponential time if $L(G)$ is well-formed.*

Proof:[sketch] Let $G = (\Pi, \mathcal{N}, S, \longrightarrow)$ be a DMG. We have seen in the proof of theorem 39 how to build a tree automaton $\mathcal{A}_G$ that accepts all parse trees that correspond to successful derivations of G. Thus, we have $L(\mathcal{A}_G) = \emptyset$ iff $L(G) = \emptyset$. To answer the well-formedness question, we build a tree automaton $L(\mathcal{B}_G)$ for those parse trees that give rise to well-formed MSCs (considering an MSC as a singleton set). One can show that $L(G)$ is realizable iff all MSCs in $L(G)$ are realizable. We deduce that G is well-formed iff $L(\mathcal{A}_G) \setminus L(\mathcal{B}_G) = \emptyset$.

As for $\mathcal{A}_G$, the construction of $\mathcal{B}_G$ will use a communication structure, that recalls which process is known. To illustrate the main idea of $\mathcal{B}_G$, we use the DMG G from

Figure 6.4. The left hand side of Figure 12.10 depicts a parse tree t of G. More precisely, t is the parse tree that corresponds to the derivation depicted in Figure 6.5. We, therefore, call t legal. Note that, for technical reasons, the functions f from a rule $A \longrightarrow_f expr$ are located at the non-terminal A to which the rule is applied. The crucial point of the construction is to record, during a derivation, only a bounded amount of information on the current communication structure of the system. A communication structure is basically a partitioning of the set of process identifiers together with a binary relation that provides information on what processes know of other processes. The right hand side of Figure 12.10 depicts a run of $\mathcal{B}_G$ on t. States, which are assigned to nodes, are framed by a rectangle. A state is hence either a pair of communication structures (together with a non-terminal, which is omitted), or an element from m$\mathbb{P}$ that occurs in G. Our automaton works bottom-up. Let us have a look at the upper right leaf of the run tree, which is labeled with its state $\mathcal{M}_{12}$. Suppose that, when it comes to executing $\mathcal{M}_{12}$, the current communication structure C_0 of the system contains two processes carrying π_1 and π_2, respectively, that know each other (represented by the two edges). When we apply $\mathcal{M}_{12}$, the outcome will be a new structure, C_1, with a newly created process that does not carry a process identifier anymore. Henceforth, the process carrying π_1 is known to that carrying π_2, but the converse does not hold. Names of nodes are omitted; instead, identical nodes are combined by a dotted line. We conclude that applying the rule $A \longrightarrow_{f_A} \mathcal{M}_{12}$ can have the effect of transforming C_0 into C_1. Therefore, (C_0, A, C_1) will be a state that can be assigned to the (A, f_A)-labeled node, as actually done in our example run. It is very important here to note that the first structure C_0 of a state (C_0, A, C_1) is *reduced* meaning that it can restrict to nodes carrying process identifiers. The structure C_1, however, might keep some unlabeled nodes, but only those that stem from previously labeled ones. Hence, the set of states of $\mathcal{B}$ will be finite, though exponential in the size of G. Like elements of m$\mathbb{P}$, a triple (C_0, A, C_1) can be applied to a communication structure so that the sequence of states that label the successors of the root transform D_0 into D_1. We can reduce D_1 towards D_2 by removing the middle node, as it does not carry a process identifier nor arises from an identifier carrying node. Thus, (D_0, S, D_2) is the state assigned to the root. It is final, as D_0 consists of only one process, which carries all the process identifiers. A final state at the root finally ensures that the run tree represents a derivation that starts in the initial configuration gathering all process identifiers, and ends in a well-formed MSC. □

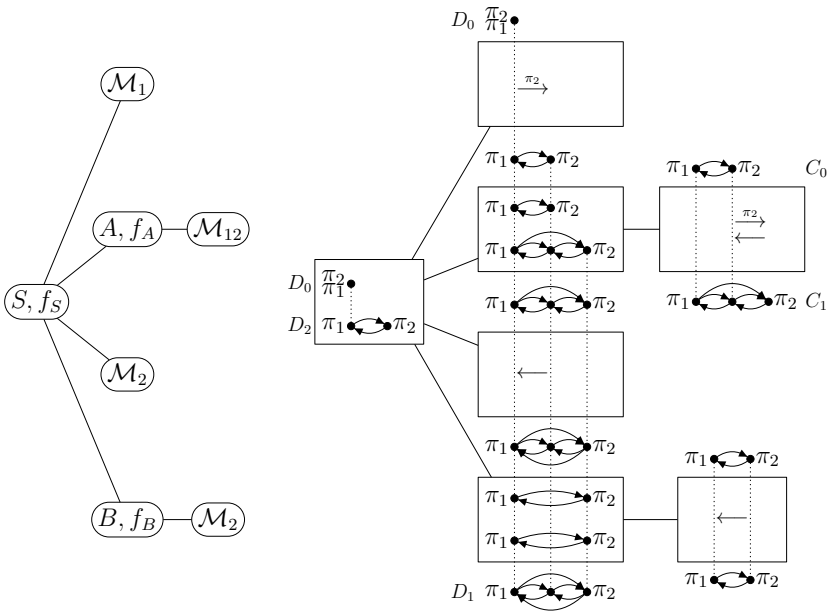


Figure 12.10: A legal parse tree of G and a run of $\mathcal{B}_G$

6 Proofs for chapter 8

6.1 Undecidability for satisfiability of LPOC

Theorem 51 *Satisfiability of LPOC formulae is an undecidable problem.*

proof This theorem is proved by reduction from the Post Correspondence Problem (PCP). The reduction is similar to that used in decision problems related to message sequence charts (see for instance [60]).

Consider an instance of PCP with Σ a finite alphabet such that $|\Sigma| > 1$, and $(g_1, h_1), \ldots, (g_n, h_n)$ a finite sequence of pairs of words over Σ. A solution (if it exists) is a finite sequence of indices $j_1, j_2, \ldots, j_t$ in $\{1, 2, \ldots, n\}$ such that $g_{j_1} g_{j_2} \ldots g_{j_t} = h_{j_1} h_{j_2} \ldots h_{j_t}$. The reduction is as follows. We pick $\mathcal{P} = \{p, q, r\}$. Let $\mathcal{AP}_p = \{a^p \mid a \in \Sigma\} \cup \{i^p \mid i \in \{1, 2, \ldots, n\}\}$, $\mathcal{AP}_q = \{a^q \mid a \in \Sigma\} \cup \{i^q \mid i \in \{1, 2, \ldots, n\}\}$, and $\mathcal{AP}_r = \{i^{pr}, i^{qr} \mid i \in \{1, 2, \ldots, n\}\}$. We take $\mathcal{AP}_{ex} = \mathcal{AP}_p \cup \mathcal{AP}_q \cup \mathcal{AP}_r$, and $\mathcal{AP}_{ob} = \emptyset$. Let O be the empty observation, that is, a computation with no states. Thus, any computation is an explanation for O.

We encode solutions to the PCP instance by computations which have the form illustrated in Figure 12.11. The total ordering of states on each process is drawn as a vertical line with the minimum state at the top. The downward-sloping arrows represent pairs (s, s') of states in the successor relation such that s, s' are on different locations. The label of each state indicates its valuation. We can construct formulae Φ_p,Φ_q,Φ_r such that $\{\Phi_p, \Phi_q, \Phi_r\}$ is satisfiable iff there exists a computation W with $W \models \{\Phi_p, \Phi_q, \Phi_r\}$ and W represents a solution to the PCP instance. It will then follow that the PCP instance has a solution iff there exists an explanation for O satisfying $\{\Phi_p, \Phi_q, \Phi_r\}$.

In the sequel, we outline the construction of Φ_p, Φ_q and Φ_r. For $A \subseteq \mathcal{AP}_p$, we write $VAL(p, A)$ for the formula $loc_p \wedge \downarrow_{1, \mathcal{AP}_p}(T)$ where T is the computation with a singleton state of location p and valuation A. In other words, the formula $VAL(p, A)$ asserts that the current state has location p and valuation A. We define the notations

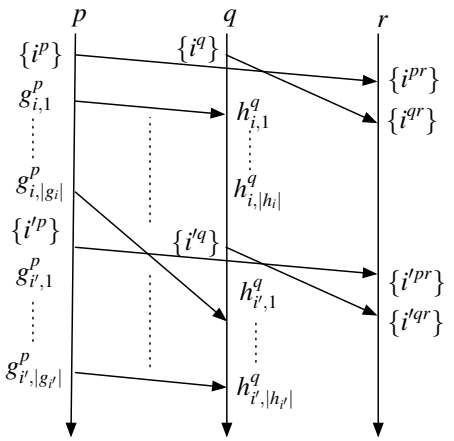


Figure 12.11: Encoding solutions to PCP with computations

$VAL(q, A)$, $VAL(r, A)$ in the same way. The formula Φ_p is the conjunction of the following formulae.

(P1) $\bigvee_{i\in\{1\ldots n\}} VAL(p, \{i^p\})$. It asserts that the valuation of the minimum state of S_p is $\{i^p\}$, for some i in $\{1 \ldots n\}$.

(P2) $\mathsf{EG}_p(\bigwedge_{i\in\{1\ldots n\}} (VAL(p, \{i^p\}) \to \varphi_i))$ where each φ_i is given as follows. Let $g_{i,j}$ be the j-th letter of g_i, for $j = 1, \ldots, |g_i|$. Then φ_i asserts that there exist m states s_1, s_2, ..., $s_{|g_i|}$ in S_p with s_1 being a successor of the current state, and s_{j+1} a successor of s_j for $j = 1, \ldots, |g_i| - 1$. Furthermore, the valuation of s_j is $\{g^p_{i,j}\}$ for $j = 1, 2, \ldots, |g_i|$, and either $s_{|g_i|}$ has no successor in S_p, or $s_{|g_i|}$ has a successor in S_p whose valuation is $\{\ell^p\}$ for some ℓ in $\{1, \ldots, n\}$.

More precisely, $\varphi_i = \downarrow_{|g_i|+1,A_p}(T) \vee (\vee_{\ell=1,2,\ldots,n} \downarrow_{|g_i|+1,A_p}(T'_\ell))$, where T is the computation $(\{t_1, t_2, \ldots, t_{|g_i|}\}, \eta_T, \leq_T, V_T)$ with $\eta_T(t_j) = p$ for every $j = 1, 2, \ldots, |g_i|$, $t_1 \leq_T t_2 \leq_T \ldots \leq_T t_{|g_i|}$, and $V_T(t_j) = \{g^p_{i,j}\}$ for every $j = 1, 2, \ldots, |g_i|$. Similarly, for each $\ell = 1, 2, \ldots, n$, T'_ℓ is the computation $(\{t_1, t_2, \ldots, t_{|g_i|}, t_{|g_i|+1}\}, \eta_{T'}, \leq_{T'}, V_{T'_\ell})$ with $\eta_{T'_\ell}(t_j) = p$ for every $j = 1, 2, \ldots, |g_i|+1$, $t_1 \leq_{T'_\ell} t_2 \leq_{T'} \ldots \leq_{T'} t_{|g_i|+1}$, $V_{T'_\ell}(t_j) = \{g^p_{i,j}\}$ for every $j = 1, 2, \ldots, |g_i|$, and $V_{T'_\ell}(t_{|g_i|+1}) = \{\ell^p\}$.

(P3) $\mathsf{EG}_p(\bigwedge_{a\in\Sigma} (VAL(p, \{a^p\}) \to \mathsf{EX}_q\, VAL(q, \{a^q\})))$. Intuitively, this asserts that every state of valuation $\{a^p\}$ of p is "matched" by a state of q with valuation $\{a^q\}$. We emphasize that the matchings are guaranteed to be one-to-one and order-preserving. This is due to the fact that for every pair p, q of processes $<<_{pq}$ is "fifo", that is one-to-one and order-preserving. More precisely, for each state s of S_p, there is at most one state s' of S_q such that $s << s'$ (and thus $s << pqs'$). Similarly, for each state s of S_q, there is at most one state s' of S_p such that $s' << s$ (and thus $s' << pqs$). Finally, for any s_1,s_2 of S_p, s'_1, s'_2 of S_q such that $s_1 << s'_1$, $s_2 << s'_2$, we have that $s_1 \leq s_2 iff s'_1 \leq s'_2$.

(P4) $\mathsf{EG}_p(\bigwedge_{i\in\{1,2,\ldots,n\}} (VAL(p, \{i^p\}) \to \mathsf{EX}_r\, VAL(r, \{i^{pr}\})))$. That is, for every state of p with valuation $\{i^p\}$, there is a "matching" state of r with valuation $\{i^{pr}\}$.

The formula Φ_q asserts the conjunction of the following conditions.

(Q1) The valuation of the minimum state of q is $\{i^q\}$ for some i in $\{1, 2, \ldots, n\}$. This is similar to case (P1) in the construction of Φ_p.

(Q2) For each $i \in \{1, 2, \ldots, n\}$, if a state of q has valuation $\{i^q\}$, then there exist $|h_i|$ subsequent states s_1, s_2, ..., $s_{|h_i|}$ of q, whose valuations are, respectively, $\{h^q_{i,j}\}$, with $h_{i,j}$ being the j-th letter of h_i, for $j = 1, 2, \ldots, |h_i|$. Further, either $s_{|h_i|}$ has no successor of location q, or $s_{|h_i|}$ has a successor of location q and valuation $\{\ell^q\}$ for some ℓ in $\{1, 2, \ldots, n\}$. The detailed formula of this case can be constructed in the same way as case (P2) in the construction of Φ_p.

(Q3) For each $a \in \Sigma$, if a state s of q has valuation $\{a^q\}$, then $\downarrow_{1,\mathcal{AP}_p}(T)$ holds at s, where T is the computation containing a singleton state of location p and valuation $\{a^p\}$. That is, every state of valuation $\{a^q\}$ of q is "matched" by a state of valuation $\{a^p\}$ of p.

(Q4) For each $i \in \{1, 2, \ldots, n\}$, if a state of q has valuation $\{i^q\}$, then it has a successor of location r and valuation $\{i^{qr}\}$. This is similar to case (P4) of Φ_p.

Finally, Φ_r asserts the conjunction of the following conditions.

(R1) The minimum state of r has valuation $\{i^{pr}\}$ for some i in $\{1, 2, \ldots, n\}$.

(R2) For each index $i \in \{1, 2, \ldots, n\}$, if a state of r has valuation $\{i^{pr}\}$, then it has a successor state, say, s, of location r and valuation $\{i^{qr}\}$. Further, either s has no successor of location r, or s has a successor of location r and valuation $\{\ell^{pr}\}$ for some $\ell \in \{1, 2, \ldots, n\}$. The detailed formula can be constructed in the same way as case (P2) of Φ_p.

(R3) For each $i \in \{1, 2, \ldots, n\}$, if a state s of r has valuation $\{i^{pr}\}$, then $\downarrow_{1,\mathcal{AP}_p}(T)$ holds at s, where T is the computation containing a singleton state of location p and valuation $\{i^p\}$.

(R4) For each $i \in \{1, 2, \ldots, n\}$, if a state s of r has valuation $\{i^{qr}\}$, then $\downarrow_{1,\mathcal{AP}_q}(T)$ holds at s, where T is the computation containing a singleton state of location p and valuation $\{i^q\}$.

It is easy to see that the constructions of Φ_p, Φ_q, Φ_r encode a PCP instance, and that there exists a computation satisfying Φ_p, Φ_q, Φ_r if and only if the corresponding instance of PCP has a solution. Even if one considers a single formula starting from a single minimal state, a similar encoding enforcing Φ_p, Φ_q, Φ_r at immedaite successors of the minimal state can be used. This completes the proof for Theorem 51. □

7 Proofs for chapter 9

7.1 Proof of Proposition 12

Proof: It is easy to see that if $h_{O,M}$ exists, then it is the (unique) function $h_{O,M} : E_O \to E_M$ that sends the k-th event of E_O on instance p onto the k-th event of $\pi_{\Sigma_{obs}}(M)$ on instance p for all $k \in \mathbb{N}$ and $p \in \mathcal{P}_{obs}$ (due to the fact that O is totally ordered on each process, and must be closed by precedence). If $h_{O,M}$ does not preserve causal ordering for some events located on distinct processes, then O can not be a sub-order of a prefix of M. Conversely, if O is a sub-order of a prefix of M, then for every ordered pair of events $e \leq_O f$ in O, we have $h_{O,M}(e) \leq_M h_{O,M}(f)$.□

Corollary 4 *Given an observation O such that $O \rhd_{\Sigma_{obs}} M$ and an observation O' such that O' is a prefix of O or O' is a sub-order of O, then $O' \rhd_{\Sigma_{obs}} M$. Furthermore, if $O' = \Pi_{\Sigma'}(O)$ for some alphabet $\Sigma' \subseteq \Sigma_{obs}$, then $O' \rhd_{\Sigma'} M$*

Proof: The proof is straightforward, as it is sufficient to consider the restriction of $h_{O,M}$ to events of O' to obtain a matching relation from O' to M.□

7.2 Proof of Proposition 13

Proposition 13 *Let O be an observation and M be a MSC. Then, checking whether $O \rhd_{\Sigma_{obs}} M$ can be done in $O(|M| + | \leq_O |.| \leq_M |)$.*

Proof: The first step to verify a matching relation is to build the mapping $h_{O,M}$ from O to M, that is compare sequences of observable events along each process. This can be computed in linear time in the size of M. Then, for each pair of events (a, b) appearing in $\leq_O$ we have to verify that $h_{O,M}(a) \leq_M h_{O,M}(b)$. □

7.3 Proofs for diagnosis algorithms (section 4)

In this part of the appendix, we will detail how to build an automaton that recognizes explanations for an observation O contained in a HMSC H. As already mentioned, observations may be collected either in a centralized or a distributed way, and observed events can be sent to supervising mechanisms via asynchronous communications. Hence, the model of our system can describe runs which projections are all larger than the observation collected so far. Note however that thanks to the prefix condition, our framework does not impose observations to be **complete** projections of an MSC labeling an accepting path of H, but should only embed into an explanation.

Our goal is to build incrementally the set of all explanations provided by a HMSC. This means in particular that if we study MSC concatenations, we should be able to test whether it is worth or not continuing along a path of a HMSC. This leads us to introduce the notion of *compatibility* defined as follows:

Definition 93 *A MSC M is* compatible *with an observation O if and only if there exists a MSC M' such that $O \rhd_{\Sigma_{obs}} M \circ M'$. Given a HMSC H and a path $\rho \in \mathcal{P}_H$, then M_ρ is* compatible *with an observation O (w.r.t HMSC H) if and only if there exists ρ' such that $\rho\rho' \in \mathcal{P}_H$ and $O \rhd_{\Sigma_{obs}} M_{\rho\rho'}$.*

When H is clear from the context, we will drop the reference to H and simply write that M_ρ is compatible with O, or even ρ is compatible with O. It is worthwhile noting that when M and O are compatible, then there exists a unique maximal embedding function h that sends a prefix of O onto events of M, and such that any embedding h' of O into $M \circ M'$ is an extension of h. Hence, the unique embedding h of O into some MSC in $\mathcal{F}_H$ can be built incrementally.

We will build a new automaton whose nodes are product of a node of the original HMSC with the subset of events of O observed so far, that will be called the *progress* of the observation. For instance, a path leading to the product state (v, E_O) should generate an execution that embeds O.

Next we outline some difficulties we will face up in order to build this automaton. First, we can remark that it is not possible to say that a path $\rho = n_0 \xrightarrow{M_1} n_1 \dots \xrightarrow{M_k} n_k$ is not an explanation of O just by considering the projections $\Pi_{\Sigma_{obs}}(M_1), \dots, \Pi_{\Sigma_{obs}}(M_k)$. This is basically due to the fact that in general $\Pi_{\Sigma_{obs}}(M_1 \circ M_2) \neq \Pi_{\Sigma_{obs}}(M_1) \circ \Pi_{\Sigma_{obs}}(M_2)$: the former may provide more ordering on projected events than the latter (see for instance the MSCs M_1 and M_2 in Figure 12.12). For similar reasons, we cannot use as a basis for diagnosis a copy of the original HMSC which transitions are labeled by projections of MSCs.

Second, another difficulty is to know the influence of unobservable events and of concatenation on the causal ordering of observable events. As already mentioned, valid explanations may contain an arbitrary number of unobserved events. Fortunately, we can always keep an abstract and bounded representation of these unbounded orders, by projecting runs of our models on observable events, and recalling some causalities. This abstraction of runs will be modeled by a partial function $g_{O,M} : \mathcal{P} \longrightarrow 2^O$ that associates to each instance $p \in \mathcal{P}$ the observed events of O preceding the last event (observed or not) on instance p at the end of the MSC labeling some path of H. More formally, for an observation O and a MSC M compatible with O, we have

$$g_{O,M}(p) = h_{O,M}^{-1}\Big(\downarrow\big(max_{\leq_M}(p)\big)\Big) \cap Dom(h_{O,M}),$$

where $h_{O,M}$ is the (unique) maximal embedding of prefixes of O in M. Notice that function $g_{O,M}$ defines an abstraction of an MSC that is not redundant with the order contained in O, since M can contain more ordering on observed events than O.

Let us illustrate the use of function g with an example. Consider the two MSCs of figure 12.12, and the observation alphabet $\Sigma_{obs} = \{a, b, b', c, c'\}$. O_1 and O_2 are the the projections of M_1 and M_2 on Σ_{obs}. We can remark that $\Pi_{\Sigma_{obs}}(M_1 \circ M_2)$ and $O_1 \circ O_2$ comport isomorphic sets of events, but define different causal orderings on theses events (b and c' are causally ordered in $\Pi_{\Sigma_{obs}}(M_1 \circ M_2)$ but not in $O_1 \circ O_2$. The reason is that the causality from $Medium$ to $Receiver$ induced by message $Info$ is lost during projection. Let us suppose that MSC M_1 has been played as an explanation of observation O_1. Then, the function g_{O_1,M_1} computed after M_1 to explain observation O_1 associates event a to process sender, events $\{a, b\}$ to process $Medium$, and events $\{a, b, c\}$ to process $Receiver$.

Let us now show that for a given observation O, the projections and the function g can be computed incrementally. To do so, we first show how to compute the projection of $M_1 \circ M_2$ according to the projections of M_1 and M_2 (i.e $\Pi_{\Sigma_{obs}}(M_1)$ and $\Pi_{\Sigma_{obs}}(M_2)$) and the function g_{O,M_1}:

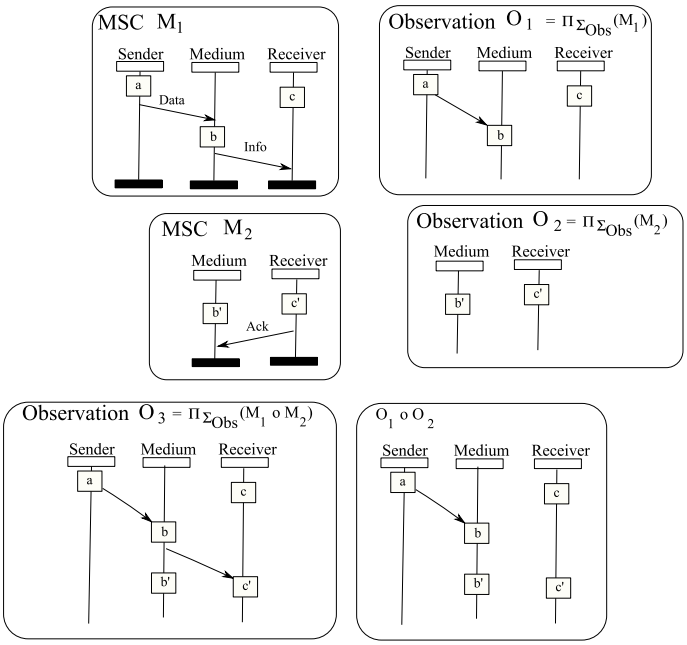


Figure 12.12: Concatenation and Projection

Proposition 18 *Let $M_1 = (E_1, \leq_1, \alpha_1, \mu_1, \varphi_1)$, $M_2 = (E_2, \leq_2, \alpha_2, \mu_2, \varphi_2)$ be two MSCs, and let $\Pi_{\Sigma_{obs}}(M_1) \circ \Pi_{\Sigma_{obs}}(M_2) = (E, \leq, \alpha, \mu, \varphi)$. Then for any observation O such that M_1 is compatible with O,*

$$\Pi_{\Sigma_{obs}}(M_1 \circ M_2) = (E, \leq', \alpha, \mu, \varphi), \text{ where}$$

$$\leq' = \left(\leq \cup \{(x,y) \in \Pi_{\Sigma_{obs}}(M_1) \times \Pi_{\Sigma_{obs}}(M_2) \mid \exists z \leq_2 y, x \in g_{O,M_1}(\varphi(z)\ )\} \right)^*$$

Proof: Let us suppose that there exists (x, y) that are ordered in $\Pi_{\Sigma_{obs}}(M_1 \circ M_2)$, but not in $\leq'$. Then, obviously $x \in E_1$ and $y \in E_2$, and furthermore, $\varphi(x) \neq \varphi(y)$. As (x, y) are ordered in $\Pi_{\Sigma_{obs}}(M_1 \circ M_2)$ without being on the same process, then there exists an event $x' \in E_1$ such that $x \leq x'$, and event $y' \in E_2$ such that $\varphi(x') = \varphi(y')$. Hence, we have that $x \in g(\varphi(x'))$, and $x \leq' y$, contradiction. □

The next step is to show that we can compute the function $g_{O,M_1 \circ M_2}$ from g_{O,M_1} and $M2$.

Proposition 19 *Let $M_1 = (E_1, \leq_1, \alpha_1, \mu_1, \varphi_1)$, $M_2 = (E_2, \leq_2, \alpha_2, \mu_2, \varphi_2)$ be two MSCs such that $M_1 \circ M_2$ is compatible with O. Then, for every $p \in \mathcal{P}$,*

$$\begin{aligned} g_{O,M_1 \circ M_2}(p) &= g_{O,M_1}(p) \cup \{g_{O,M_1}(\varphi(e)) \mid e \leq_2 e', \varphi(e') = p\} \\ &\cup \{e \in h^{-1}_{O,M_1 \circ M_2}(\pi_{\Sigma_{Obs}}(M_2) \cap O) \mid e \leq_2 e', \varphi(e') = p\}, \end{aligned}$$

where $h_{O,M_1 \circ M_2}$ is the largest embedding of prefixes of O into $M_1 \circ M_2$.

Proof: Suppose that there exists a process $p \in \mathcal{P}$ and a event $e \in O$ such that $e \in g_{O,M_1 \circ M_2}(p)$ but e is not in the incremental computation of g. Then, clearly $e \notin g_{O,M_1}(p)$. If $e \in M_1$, then $e \in g_{O,M_1 \circ M_2}(p)$ if and only if there exists a causal

chain of events $h_{O,M_1\circ M_2}(e) < e_1 < ...e_i < e_j < ... < e_n$ in $M_1 \circ M_2$ such that $e_n \in E_2$ and is located on process p, $e_i \in M_1$ and $e_j \in M_2$ are located on the same process. Hence by definition, $e \in g_{O,M_1}(\varphi(e_i))$ and also belongs to the incremental construction of $g_{O,M_1\circ M_2}(p)$. If $e \in M_2$, then $e \in g_{O,M_1\circ M_2}(p)$ if and only if there exists a causal chain $h_{O,M_1\circ M_2}(e) < ... < e_n$ in M_2 such that $e_n \in E_2$ is located on process p. This case is also captured by the incremental construction. □

Hence it is sufficient when studying an arbitrary long path ρ of a HMSC that is compatible with an observation O to memorize the finite set of observed events in M_ρ and g_{O,M_ρ} to be able to build incrementally a faithful projection of the MSC labeling any continuation of this path (and make sure that this continuation is still compatible with the observation). We are now ready to build the product $\mathcal{A}_{O,H}$ of an observation O on an alphabet Σ_{obs} and a HMSC H:

Definition 94 *Given a HMSC H and an observation O on an alphabet Σ_{obs}, $\mathcal{A}_{O,H}$ is an automaton defined by: $\mathcal{A}_{O,H} = (Q, \delta, \mathcal{M}, q_0, F')$, where δ is a new transition relation, $Q \subseteq N \times Prefix(O) \times \mathcal{F}$, where $\mathcal{F}$ is the set of functions from $\mathcal{P}$ to 2^O.*

- *$q_0 = (n^i, M_\varepsilon, g_\emptyset)$, where $g_\emptyset$ is a function over an empty domain.*
- *$\Big((n, E, g), M, (n', E', g')\Big) \in \delta$ with $E \neq O$ iff*
  - *$n \xrightarrow{M} n'$,*
  - *For every process p, either E''_p is a prefix of E_{Op}, or E_{Op} is a prefix of E''_p, where $E'' = E \uplus \pi_{\Sigma_{Obs}}(M)$. When this property holds, then $E' = E_O \cap h^{-1}(E'')$, where h is the largest partial mapping of events of E_O onto events of E'' that preserves local ordering $\leq_p$, and labeling. Note that E' is necessarily a prefix of O. When the property does not hold for MSC M, then the transition is not allowed.*
  - *$g'(p) = g(p) \cup \{g(\varphi(e)) \mid e \leq_M e', \varphi(e') = p\} \cup \{e \in \pi_{\Sigma_{Obs}}(M) \mid e \leq_M e', \varphi(e') = p\}$,*
  - *For all $a, b \in E'$ with $a <_0 b$, either $a, b \in E$ (in this case, the ordering of a and b has already been checked in former transitions), or $h(a) \leq_M h(b)$, or $\exists c \leq_M h(b)$ with $a \in g(\varphi(c))$ (the existence of a causal ordering between a and b is ensured by proposition 18).*
- *$\Big((n, E_O, g), M, (n', E_O, g)\Big) \in \delta$ iff $n \xrightarrow{M} n'$.*
- *$F' = \{(n, E_O, g) \mid n \in F\}$,*

Note that $g(p)$ is updated only when the explanation provided at a given state is incomplete. It is updated to memorize the observable events in the causal past of the last event (observed or not) executed by each instance. Similarly, we make sure during construction of a transition $\Big((n, E, g), M, (n', E', g')\Big) \in \delta$ that any order $a <_O b$ is preserved in E': either $h(a), h(b)$ are predecessors of all events of

M and their ordering was already checked, or they are ordered in M, or $h(a)$ is in M and $h(b)$ is a successor of an event that necessarily occurs after $h(a)$. We hence ensure by construction that for any path ρ of $\mathcal{A}_{O,H}$, M_ρ is compatible with O. Transitions of $\mathcal{A}_{O,H}$ of the form $\Big((n,E,g),M,(n',E',g')\Big) \in \delta$ can be projected to obtain transitions of the form (n,M,n') that are used in H to move from one state to another. We denote by $\mathbb{L}_{\mathcal{A}_{O,H}}$ the set of accepting paths of $\mathcal{A}_{O,H}$, and by $\mathbb{L}_{H,\mathcal{A}_{O,H}} \subseteq \mathcal{P}_H$ the set of paths of H that are projections of $\mathbb{L}_{\mathcal{A}_{O,H}}$ on the first component of each state.

We are now ready to prove theorem 53

Theorem 53 [77] *Let $\mathcal{A}_{O,H}$ be the HMSC computed from O and H, and $\rho \in \mathcal{P}_H$. Then $O \triangleright_{\Sigma_{obs}} M_\rho$ iff $\rho \in \mathbb{L}_{H,\mathcal{A}_{O,H}}$. Moreover, $\mathcal{A}_{O,H}$ is of size $O(|H| \times |O|^{|\mathcal{P}|\times|\mathcal{P}_{Obs}|})$.*
Proof: It is obvious from the construction of δ that any accepting path ρ of $\mathbb{L}_{H,\mathcal{A}_{O,H}}$ generates a MSC M_ρ such that $O \triangleright_{\Sigma_{obs}} M_\rho$, as we forbid any transition where $a <_O b$ and $h_{O,M_\rho}(a) \not\leq_{M_\rho} h_{O,M_\rho}(b)$. Reciprocally, consider $\rho \notin \mathbb{L}_{H,\mathcal{A}_{O,H}}$, then either $\rho \notin H$ and we are done or $\rho \in H$ but $\nexists \rho' \in \mathbb{L}_{H,\mathcal{A}_{O,H}}$ such that ρ is the restriction of ρ' on its first component. Consider ρ_1 and ρ_2 such that $\rho = \rho_1\rho_2$ and ρ_1 is the greatest prefix of ρ that is compatible with a prefix of ρ'. Thus ρ_2 is of the form $(n,M,n').\rho_3$ and $M_{\rho_1} \circ M$ is not compatible with O, which is thus also the case for ρ.

For the complexity statement, notice that a prefix can be uniquely represented by remembering its last event on each observed instance. Hence the number of prefixes of O is lower than $|O|^{|\mathcal{P}_{Obs}|}$. Moreover, notice that g associates to every instance i a prefix of O. At last, notice that for every state (n,E,g), we have $E = \bigcup_{p\in\mathcal{P}} g(p)$, hence E is superfluous as it can be computed from g. We however kept the prefixes of the observation in the definition of states for the sake of readability of the construction of $\mathcal{A}_{O,H}$.□

Theorem 53 means in particular that $\mathbb{L}_{H,\mathcal{A}_{O,H}} = \mathcal{P}_{O,H}$. Hence, $\mathcal{A}_{O,H}$ is the generator of all explanations of observation O provided by the HMSC model H. The restriction of $\mathcal{A}_{O,H}$ to coaccessible states of F' is the *diagnosis* provided for observation O from the HMSC model H.
Remark: Note however that paths in $\mathbb{L}_{H,\mathcal{A}_{O,H}}$ are not the *minimal paths* embedding O, as $\mathcal{A}_{O,H}$ allows **any** transition of H from its accepting states. To consider only **minimal** paths, one should consider only the relation $\delta' = \delta \cap \{\big((n,E,g),M,(n',E',M')\big) \mid E \neq E_O\}$, and the set of accepting nodes $F' = \{(n,E_O,g)\}$. For a centralized offline diagnosis performed with a complete observation, this has no importance. However, we will see in section 4.2 that when the diagnosis problem is split into sub-problems, it is important to return **all** paths embedding the observation.

Let us now show the construction of the diagnosis automaton on an example Consider the HMSC H and the observation O of Figure 12.13. The HMSC describes the behavior of three processes $P1, P2, P3$. Let us denote by e_1 the occurrence of action a in O and by e_2 the occurrence of action b.

Let us suppose that we have equipped a distributed system to observe any occurrence of actions a and b and that we obtain the observation O. Clearly, $n_0 \xrightarrow{M1} n_0 \xrightarrow{M2} n_1$ is not an explanation of O for the observation alphabet $\Sigma_{obs} = \{a,b\}$, as a and b are not causally related in $M1 \circ M2$. The automaton $\mathcal{A}_{O,H}$ computed from O and H with this observation alphabet is given in Figure 12.14. The transitions

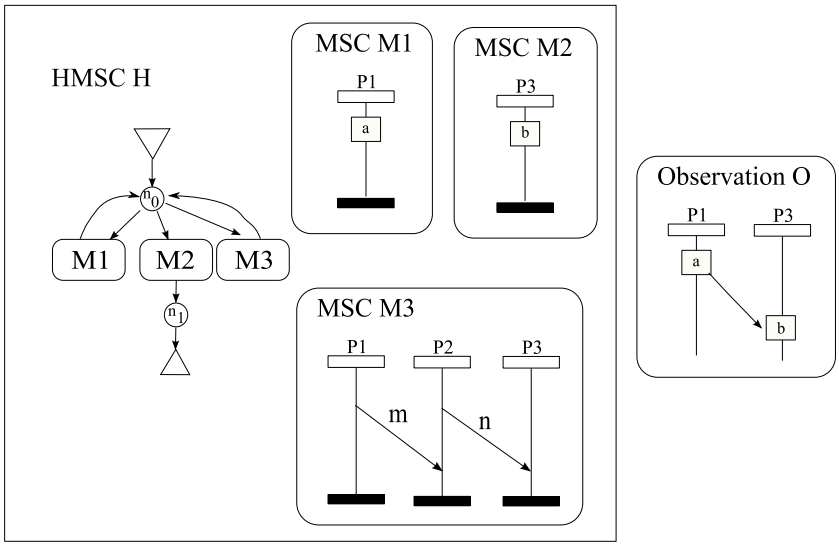


Figure 12.13: A HMSC example and an observation

with a dark cross symbolize transitions of the original HMSC that cannot be fired in the diagnosis automaton. For example, from the initial state, the transition labeled by $M2$ cannot be used, as any path ρ starting with this transition would not allow a matching from O to M_ρ. One can easily verify that O matches any MSC composition of the form $M3^* \circ M1 \circ M3 \circ M3 \circ M3^* \circ M2$. Note that if we choose as observation alphabet $\Sigma_{obs} = \{a, b, !m\}$, the observation O has no explanation in H.

7.4 Offline existence- proof of theorem 54

Theorem 54 *Let H be a HMSC, Σ_{obs} be an observation alphabet, and O be an observation. Deciding whether there exists an explanation for O in H w.r.t. Σ_{obs} is an NP-complete problem.*

Proof: First, let us show that the existence problem is in NP. There exists an explanation for O in H if and only if we can exhibit a path ρ of H such that $O \rhd_{\Sigma_{obs}} M_\rho$. Let us suppose that ρ is a path of length greater than $|O|^2.|\mathcal{P}|.|H|$. Whenever, ρ is an explanation for O, this path has at most $|O|$ transitions labeled by an MSC which contains an event e that is the image of some event of O via the matching function h_{O,M_ρ}. Hence, we can exhibit a subsequence of consecutive transitions in ρ of size greater than $|O|.|\mathcal{P}|.|H|$ that are only labeled by unobservable MSCs, or which labeling MSCs are not used to explain O (that is they comport only events which are not in $h_{O,M_\rho}(E_O)$). Then, two cases can appear. Either ρ' is a suffix (resp. a prefix) of ρ, or not. If ρ' is a suffix (resp. a prefix), then we can remove it from ρ to obtain a smaller explanation. If not, then ρ is of the form $\rho = \rho_1.\rho'.\rho_2$. As ρ' is of size greater than $|O|.|\mathcal{P}|.|H|$, it necessarily contains at least $|O|.|\mathcal{P}|$ cycles of H, and hence it is a sequence of transitions of the form $\rho' = u_1.\beta_1.u_2.\beta_2 \ldots \beta_{|O|.|\mathcal{P}|}.u_{|O|.|\mathcal{P}|+1}$, where each β_i is a cycle of H. Note that each path ρ corresponds to a path of $\mathcal{A}_{O,H}$, but that loops of H are not necessarily loops of $\mathcal{A}_{O,H}$, as nodes of $\mathcal{A}_{O,H}$ contain a reference to an HMSC node, plus a function g (observed events sets are redundant with g, as shown in theorem 53, and can

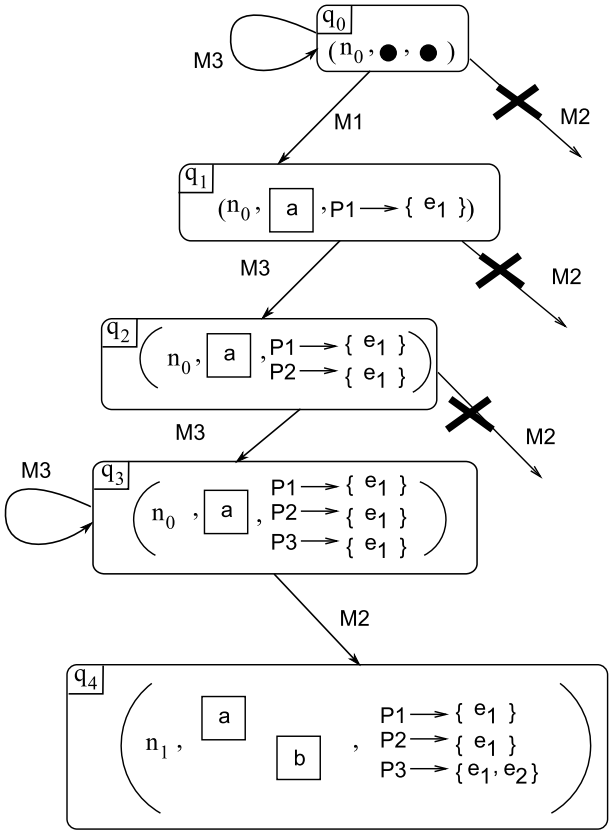


Figure 12.14: A diagnosis automaton

be forgotten). Hence, each β_i is a cycle from a node n_i to a node n_i in H, and is mapped to a path from a node (n_i, E, g_i) to a node (n_i, E, g_{i+1}) in $\mathcal{A}_{O,H}$. If $g_i = g_{i+1}$, then this path of $\mathcal{A}_{O,H}$ is a cycle, and can be removed from ρ to obtain a smaller explanation $\rho_1.u_1\beta_1 \dots \beta_{i-1}.u_i.u_{i+1}.\beta_{i+1} \dots u_{|O|.|\mathcal{P}|+1}.\rho_2$. If $g_i \neq g_{i+1}$, then there exists at least one process such that $|g_i(p)| > |g_{i+1}(p)|$. Let us suppose that for every $i \in 1..|O|.|\mathcal{P}| - 1$, we have that β_i is mapped to a path of $\mathcal{A}_{O,H}$ from (n_i, E, g_i) to (n_i, E, g_{i+1}) with $g_i \neq g_{i+1}$. Then, we necessarily have $g_{|O|.|\mathcal{P}|-1}(p) = E$ for every p, and $\beta_{|O|.|\mathcal{P}|}$ is mapped to a loop of $\mathcal{A}_{O,H}$, and can be removed from ρ to obtain a smaller explanation $\rho_1.u_1\beta_1 \dots \beta_{|O|.|\mathcal{P}|}.u_{|O|.|\mathcal{P}|+1}.\rho_2$. Hence, if an explanation exists for an observation O, then there is necessarily an explanation of length at most $|O|^2.|\mathcal{P}|.|H|$.

Now let us show that for a path $\rho = n_0 \xrightarrow{M_1} n_1 \dots \xrightarrow{M_k} n_k$, we can check in polynomial time whether $O \rhd_{\Sigma_{obs}} M_\rho$. First, the sequential concatenation of all MSCs to obtain M_ρ can be computed in polynomial time in the size of the path. Let m be the maximal size of the causal ordering relation, and n be the maximal number of events in all MSCs of H. We can use the following algorithm to compute the concatenation $M_1 = (E, \leq, \alpha, \mu, \varphi)$ of two MSCs $M_i = (E_i, \leq_i, \alpha_i, \mu_i, \varphi_i)$, with n_i events and causal ordering of size m_i, $i \in 1, 2$.

- Compute $E = E_1 \uplus E_2$
- initalize $\leq$ with $\leq_1 \uplus \leq_2$
- for every process $p \in \mathcal{P}$, find the maximal event x on p in M_1 and the minimal event on p in M_2, and add $x \leq y$ to the ordering relation. Then compute the closure : for every $z \leq_1 x$ and every $y \leq_2 z'$, add $z \leq z'$ to the ordering relation.

This gives a complexity of $O(n1 + n2 + m1 + m2 + p.(n1.m1 + n2.m2 + m1.m2))$ for concatenation. Computing M_ρ resumes to k concatenations of MSCs with less than $k.n$ events and a causal ordering relation of size smaller than $(kn)^2$, and can hence be performed in polynomial time, and results in a MSC with at most kn events and a causal ordering relation of size at most $(kn)^2$. Then, from proposition 13, verifying that $O \rhd_{\Sigma_{obs}} M_\rho$ can be done at most in $O(k.n + | \leq_O |.(k.n)^2)$. Hence, we can guess a path of H to explain O in polynomial time, and check in polynomial time whether it is an explanation for O. So, the existence problem is in NP.

Now, let us show that the existence problem is NP-hard. We proceed by reduction from the $3SAT$ problem. Let $\varphi = C_1 \wedge \dots \wedge C_m$ be a conjunctive formula in normal form over n variables $v_1, \dots, v_n$, with m clauses, and where each clause C_i is of the form $C_i = l_i^1 \vee l_i^2 \vee l_i^3$, and each literal l_i^j refers to a variable in $v_1, \dots, v_n$ or its negation, that is $l_i^j = v_k$ or $l_i^j = \overline{v_k}$, for some $k \in 1..n$. We can build a HMSC H with $n + 3$ nodes, $2n + 2$ transitions and 2n+2 MSCs, an observation alphabet Σ_{obs} and an observation O such that φ is satisfiable iff O has an explanation in H w.r.t. Σ_{obs}. The observation and the HMSC are defined over a set of processes $\mathcal{P} = \{P_{v_1}, \dots, P_{v_n}\} \cup \{P_{c_1}, \dots, P_{c_n}\}$. The observation alphabet Σ_{obs} is composed of $m+1$ letters $\{a_0, a_{c_1}, \dots, a_{c_m}\}$. The observation O comports one occurrence of each letter in Σ_{obs}, and is such that the occurrence of a_0 is located on process P_{v_1}, the occurrence of each a_{c_i} is located on process P_{c_i} and the event labeled by a_0 causally

precedes all other events (see Figure 12.15). The HMSC H, also depicted in Figure 12.15, comports a set of nodes $Q = \{q_0, q_e, q_f\} \cup \{q_{v_1}, \ldots, q_{v_n}\}$, and is labeled by MSCs $Start, End$ and $T_{v_1}, \ldots, T_{v_n}, F_{v_1}, \ldots, F_{v_n}$. The MSC $Start$ consists in a single occurrence of action a_0 located on process P_{v_1}. The MSC End consists in one occurrence of action a_{c_i} on each process P_{c_i}, $i \in 1..m$. Each MSC $T_i = X_1 \circ \cdots \circ X_m \circ V_i$ is a concatenation of $m+1$ MSCs. Each X_j is either an MSC that contains a message from P_{v_i} to P_{c_j} if one of the literals of clause C_j is v_i, and is the empty MSC otherwise. MSC V_i is a message from process P_{v_i} to process $P_{v_{i+1}}$ if $i < n$ and the empty MSC otherwise. Each MSC $F_i = Y_1 \circ \cdots \circ Y_m \circ V_i$ is a concatenation of $m+1$ MSCs. Each Y_j is either an MSC that contains a message from P_{v_i} to P_{c_j} if one of the literals of clause C_j is $\overline{v_i}$, and is the empty MSC otherwise. Finally, there is a transition from q_0 to q_{v_1} labeled by $Start$, a transition from q_e to q_f labeled by End, and two transitions from q_{v_i} to $q_{v_{i+1}}$ respectively labeled by T_i and F_i for every $i \in 1..n-1$, and two transitions from q_{v_n} to q_e respectively labeled by T_n and F_n. Clearly, an occurrence of each action in Σ appears in an explanation ρ of O if and only if ρ is a path from q_0 to q_f.

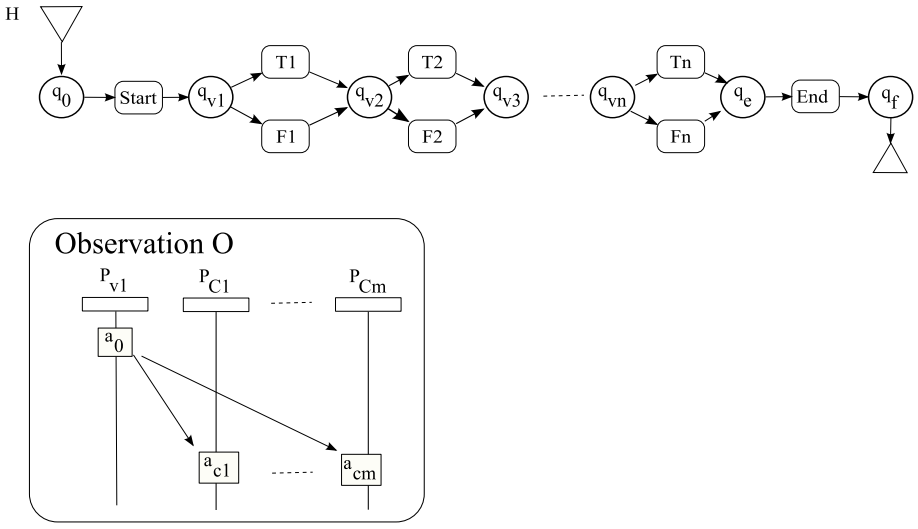


Figure 12.15: Encoding SAT problems with an existence problem

Clearly, φ is satisfiable iff there is a variable assignment such that for every clause C_j, not all literals l_j^1, l_j^2, l_j^3 are evaluated to false, and hence iff there is an explanation that embeds a causal ordering from a_0 to every a_{c_j}. □

Proposition 14 [77] *Let H be a HMSC, Σ_{obs} be an observation alphabet, and O be an observation. Let $\Sigma'_{obs} \subseteq \Sigma_{obs}$. Then if $\Pi_{\Sigma'_{obs}}(O)$ has no explanation from H w.r.t. Σ'_{obs}, then O has no explanation w.r.t. Σ_{obs} from H.*

Proof: Suppose that there exists no explanation for an observation O, with alphabet Σ'_{obs}, but that we can build an automaton $\mathcal{A}_{O,H}$ with observation alphabet Σ_{obs}. In particular, it means that for every path ρ of H, there is no embedding of O into M_ρ w.r.t. Σ'_{obs}. If no embedding exists, then either there exists a process p such that $\Pi_{\Sigma'_{obs} \cap \Sigma_p}(O)$ is not a prefix of $\Pi_{\Sigma'_{obs} \cap \Sigma_p}(M)$, and then M does not embed O with observation alphabet Σ_{obs}, or there exist two events $x \leq_O y$ with labels $\alpha(x), \alpha(y) \in \Sigma'_{obs}$. As projection preserves ordering, x and y are ordered both in O and $\Pi_{\Sigma'_{obs}}(O)$,

and M can not embed O. $\square$

7.5 Splitting the diagnosis

Theorem 55 *For every HMSC H and observation O, we have $\mathbb{L}_{H,\mathcal{A}_{O,H}} = \mathbb{L}_{H,\mathcal{A}_\otimes}$, where $\mathcal{A}_\otimes = \bigotimes_{p\neq q\in\mathcal{P}_{Obs}} \mathcal{A}_{p,q}$.*

Proof : For every pair of processes p, q in $\mathcal{P}_{obs}$, the observation alphabet $\Sigma' = \Sigma_{obs} \cap (\Sigma_p \cup \Sigma_q)$ is contained in Σ_{obs}. Hence, from corollary 4, it is obvious that for every path ρ of H, if M_ρ is an explanation of an observation O, then M_ρ is also an explanation for the projection $\Pi'_\Sigma(O)$ on a the smaller observation alphabet Σ'. Hence $\mathcal{P}_{O,H} \subseteq \mathbb{L}_{H,\mathcal{A}_{p,q}}$ for all $p, q \in \mathcal{P}_{Obs}$, and we have the first inclusion $\mathbb{L}_{H,\mathcal{A}_{O,H}} = \mathcal{P}_{O,H} \subseteq \mathbb{L}_{H,\mathcal{A}_\otimes}$.

For the second inclusion, let $\rho \in \mathbb{L}_{H,\mathcal{A}_\otimes}$. This means in particular that $\rho \in \mathbb{L}_{H,\mathcal{A}_{p,q}}$ for every pair p, q in $\mathcal{P}_{obs}$. Let $h_{O,M_\rho} : E_O \to E_{M_\rho}$ be the (unique) function such that the k-th event of E_0 on instance p is sent by h_{O,M_ρ} to the k-th event of $\alpha^{-1}_{M_\rho}(\Sigma_{Obs})$ on instance p for all k and $p \in \mathcal{P}_{Obs}$. We can denote by $h_{p,q}$ the restriction of h_{O,M_ρ} to events located on $p, q \in \mathcal{P}_{Obs}$. Clearly, $h_{p,q}$ is also the unique mapping defined by $\pi_{\Sigma_{Obs}\cap(\Sigma_p\cup\Sigma_q)}(O) \triangleright M_\rho$.

We have easily that for every $p, q \in \mathcal{P}_{Obs}$, for every event e located on p, $\alpha_O(e) = \alpha_{M_\rho}(h_{O,M_\rho}(e)) = \alpha_{M_\rho}(h_{p,q}(e))$. Then, for every pair of events $e, f \in E_O$ located on p, we have that $e \leq_O f$ implies that $h_{O,M_\rho}(e) = h_{p,q}(e) \leq_{M_\rho} h_{p,q}(f) = h_{O,M_\rho}(f)$.

Now, assume that $e, f \in O$ are causally ordered and located on different instances, p and q. Since $\rho \in \mathbb{L}_{H,\mathcal{A}_{p,q}}$, and since the projection of O on a pair of processes preserves the ordering on projected events, we have $h_{O,M_\rho}(e) = h_{p,q}(e) \leq h_{p,q}(f) = h_{O,M_\rho}(f)$.

At last, assume by contradiction that $h_{O,M_\rho}(E_O)$ is not a prefix of $\pi_{\Sigma_{obs}}(M_\rho)$. Then there exists $e, f \in M_\rho$, located on processes p and q, and of type in Σ_{obs} such that $e \leq_M f$, $e \notin h_{O,M_\rho}(E_O)$ and $f \in h_{O,M_\rho}(E_O)$. However, since $\rho \in \mathbb{L}_{H,\mathcal{A}_{p,q}}$, we know that $h_{p,q}(E_O)$ is a prefix of some sub-order of $\pi_{p,q}(\pi_{\Sigma_{obs}}(M_\rho))$, which gives us a contradiction. $\square$