HABILITATION À DIRIGER DES RECHERCHES

présentée devant

L'Université de Rennes 1
Spécialité : informatique

par

Benoît Caillaud

Analyse, contrôle et synthèse des systèmes concurrents
Analysis, control and synthesis of concurrent systems

Contents

Chapter 1

Introduction

Résumé : *Mes travaux de recherche ont pour principal objet la réalisation par des méthodes effectives des systèmes réactifs et répartis, en partant de spécifications de haut niveau, partielles et hétérogènes. Je me suis attaché à concenvoir des méthodes, des algorithmes et des outils permettant la conception ou la synthèse de logiciel réactif à partir de la combinaison de plusieurs spécifications, décrivant le comportement attendu du système, selon plusieurs points de vues : fonctionnel (synchronisation, conflit, communication), contrôle (sûreté, atteignabilité, vivacité), architecturaux (placement, partitionnemnt, ségrégation), quantitatifs (temps de réponse, coût de communication, disponibilité, fiabilité). Mes travaux de recherche se déclinent selon trois axes détaillés dans les parties suivantes : (I) conception par contrats, théorie des interfaces modales (chapitre 2) et stochastiques (chapitre 3) ; (II) systèmes hétérogènes, réalisations partiellement asynchrones de spécifications synchrones (chapitre 4) et modélisation hybride en utilisant des nombres réels non-standards (chapitre 5) ; (III) synthèse et contrôle des systèmes concurrents et communicants (chapitres 6 et 7).*

1.1 Introduction

My research interests cover the realization by algorithmic methods of reactive and distributed systems from partial and heterogeneous specifications. I have contributed to the design of methods, algorithms and tools capable of synthesizing reactive software from one or several incomplete descriptions of the system's expected behavior, regarding functionality (synchronization, conflicts, communication), control (safety, reachability, liveness), deployment architecture (mapping, partitioning, segregation), or even quantitative performances (response time, communication cost, throughput).

These techniques are better understood on fundamental models, such as automata, Petri nets, event structures, their timed or stochastic extensions. The results obtained on these basic models

are then adapted to those realistic but complex models commonly used to design embedded and telecommunication systems.

My scientific objectives can be summarized as follows:

A focus on a precise type of applications: The design of real-time embedded software to be deployed over dedicated distributed architectures. Engineers in this field face two important challenges. The first one is related to system specification. Behavioral descriptions should be adaptable and composable. Specifications are expressed as requirements on the system to be designed. These requirements fall into four categories: (i) functional (synchronization, conflict, communication), (ii) control (safety, reachability, liveness), (iii) architectural (mapping, segregation) and (iv) quantitative (response time, communication cost, throughput, etc). The second challenge is the deployment of the design on a distributed architecture. Domain-specific software environments, known as *middleware* or *real-time operating systems* or *communication layers*, are now part of the usual software design process in industry. They provide a specialized and platform-independent distributed environment to higher-level software components. Deployment of software components and services should be done in a safe and efficient manner.

A specific methodology: The development of methods and tools which assist engineers since the very first design steps of reactive distributed software. The main difficulty is the adequacy of the proposed methods with standard design methods based on components and model engineering, which most often rely on heterogeneous formalisms and require correct-by-construction component assembly.

A set of scientific and technological foundations: Those models and methods encompassing (i) the distributed nature of the systems being considered, (ii) true concurrency and the system openness to a non-deterministic or even stochastic environment, and (iii) real-time.

My contributions consist in methods, algorithms and tools producing distributed reactive software from partial heterogeneous specifications of the system to be synthesized (functionality, control, architecture, quantitative performances). This means that several heterogeneous specifications (for instance, sequence diagrams and state machines) can be combined, analyzed (are the specifications consistent?) and mapped to lower-level specifications (for instance, communicating automata, or Petri nets).

The scientific approach of my research work begins with a rigorous modeling of problems and the development of sound theoretical foundations. This not only allows to prove the correctness (functionality and control) of the proposed transformations or analysis; but this can also guarantee the optimality of the quantitative performances of the systems produced with our methods (communication cost, response time).

Synthesis and verification methods are best studied within fundamental models, such as automata, Petri nets, event structures, synchronous transition systems. Then, results can be adapted to more realistic but complex formalisms, such as those developed in the realm of *model-based design* (for instance the SysML based HRC formalism designed during the SPEEDS European

project [176, 68]). My research work is divided in three tracks reviewed below: Interface theories supporting modular methods in embedded system design; Analysis and design of heterogeneous systems; Synthesis and control of concurrent systems.

1.2 Modular design of embedded systems with interface theories

1.2.1 Introduction

The design of reusable components calls for rich specification formalisms, with which the interactions of a component with its environment combines expectations with guarantees on its environment. These are captured either by *assume/guarantee contracts* [28, 31], or by automata-theoretic formalisms such as interface automata [78]. In a recent work [149, 148, 150, 152], we have shown that both approaches can be unified in the realm of the modal interface algebra. In this context, we have investigated questions related to the composition and refinement of system-level and component-level requirements captured either as assume/guarantee contracts or modal specifications. This has helped up to characterize the generic properties that an interface theory should have in order to support modular design methods, including top-down and bottom-up approaches and arbitrary combinations of the two, as it is often the case in industrial practice.

1.2.2 A quick review of industry needs

In software engineer's literature, the term *contract* refers to the set of specifications that describe what a system should guarantee under some assumption. Nowadays, OEM perform system design and integration by importing/reusing entire subsystems provided by equipment suppliers. It is crucial that the subsystems are designed according to some rules; which highlights the importance of providing good notions of contracts. According to our understanding of industrial needs, gained during the SPEEDS European project [68, 176], the following list of requirements applies to the notions of *contract* and *interface* in the context of Embedded Systems.

Contracts as legal bindings

Complex embedded and reactive systems are generally developed under a multi-layered OEM-supplier chain. Hence, a contract-based methodology should offer provision for formalizing the technical part of contractual relations. This should be achieved by formalizing, for a considered subsystem: 1/ its context of use (*assumptions*), and 2/ what is expected from the subsystem (*guarantees*).

Making assumptions and guarantees explicit enables design in isolation and facilitates maturity assessment of the system under development. Contracts as legal bindings is also relevant — with a milder understanding of the term "legal" — for the development of different subsystems or different aspects of the system, by different teams of the same company.

Component based development of complex systems

Complexity can be addressed by decomposing systems into components and along different aspects. The consequences of this can be articulated as follows.

1. When developed under a contract-based methodology, subsystems or components should be designable in isolation, by including the needed information regarding possible future contexts of use. Subsystems or components should be substitutable to their specifications. Moreover, their integration should raise no problem.
2. Large systems are concurrently developed for their different *aspects* or *viewpoints* by different teams using different frameworks and tools. Examples of such aspects include the functional aspect, the safety or reliability aspect, the timing aspect (which is central in both time-triggered and event-driven multi-tasking models of computation and communication), and memory and power aspects. Each of these aspects requires specific frameworks and tools for their analysis and design. Yet, they are not totally independent but rather interact. The issue of dealing with multiple aspects or multiple viewpoints is thus essential. This implies that several contracts are associated with a same system, sub-system, or component, namely at least one per viewpoint. These contracts are to be interpreted in a conjunctive way.
3. The need for supporting conjunctive contracts also follows from the current practice in which early requirement capture relies on Doors or even Excel for collecting many individual requirements. These requirements typically consist of English text, semi-formal languages [48] whose sentences are translatable into predefined behavioral patterns, or even graphical scenario languages [73, 110].

Design processes and systems architectures

It is highly desirable that designing by contracts has the mildest possible impact on the design flow and on the possible choices regarding specification and system architectures — each OEM has its own design flow that is part of its competitive advantage. Contract-based design should support top-down and bottom-up navigation between mixed architectures of sub-systems or components described either as sets of contracts or as implementations. Such architectures must be rich enough to support virtual prototyping with design space exploration. Design by contracts should also comply with established formalisms or notations used in different steps of the design flow. Different industrial sectors advocate different system architectures, an example is AUTOSAR in the automotive industry or the proprietary IMA (integrated modular avionics) architectures proposed independently by the two main aircraft manufacturers.

1.2.3 Anatomy of an interface theory

Because of the requirements detailed above, an interface theory is complied to satisfy several generic algebraic properties. First of all, an interface theory deals with objects of two sorts: *Interfaces* that are used to capture system-/component-wide requirements and *implementations*, which

are actual realizations, or models of them, that satisfy the requirements captured by an interface. In most of existing works, *transition systems* [5], formal languages and their weighted, timed or probabilistic extensions are used to define implementations. However, for the time being, we consider implementations as abstract objects and review the expected algebraic properties of interfaces. We assume that we are given a *satisfaction* relation relating implementations to interfaces. Implementations are denoted by $M, N, \ldots$, while interfaces are denoted by $C, D, \ldots$. The satisfaction relation is denoted by $\models$ and write $M \models C$ whenever implementation M satisfies interface C.

The satisfaction relation subsumes a refinement preorder relation on interfaces. Interface C refines interface D if and only if the set of implementations of C is included in the set of implementations of D:

$$C \leq D \text{ iff } \forall M,\ M \models C \ \Rightarrow M \models D$$

Refinement is the corner-stone of all interface theories found in the literature [78, 30, 31, 138, 122, 121], including the two theories presented in chapters 2 and 3 and their corresponding original publications [150, 152, 153, 57]. However, refinement is a semantic relation that turns out to be undecidable or untractable, in a few instances of interface theories. There are even cases where the decidability of the refinement relation is unknown. For all these reasons, it may be necessary to give up completeness (the implication $(\forall M,\ M \models C \ \Rightarrow M \models D) \Rightarrow C \leq D$), at the benefit of decidability and computational complexity. An alternative approach is to restrict interfaces to a sub-class where refinement is not only decidable but can be checked with a reasonable computational complexity. In our work on modal interfaces and constraint Markov chains, we have resorted to this this second approach and assumed interfaces to be deterministic.

Methodological requirements call for supporting conjunctions of interfaces. Some theories provide a conjunction operator for this purpose, but some theories do not (interface automata and general modal transition systems, for instance). In our work, the conjunction operator is instrumental to the theories we have proposed, whether they are assume/guarantee contracts, modal interfaces or constraint Markov chains. This operator satisfies the properties of a logical conjunction operators:

$$\forall M, M \models C \wedge D \iff M \models C \text{ and } M \models D$$

To reflect the hierarchical decomposition of the system into sub-systems and components, implementations on one hand, and interfaces on the other hand, must be equipped with commutative and associative *structural composition* operators that we denote by $\times$ and $\otimes$, respectively. The *parallel composition* operator $\times$ applies to implementations and often represents a partially synchronized product of transition systems, or the set theoretic intersection of two formal languages. Remark that concerning the probabilistic framework presented in chapter 3, parallel composition is defined thanks to a very strong assumption of stochastic independence. Here, we consider the *parallel composition* as an abstract commutative and associative internal composition operator. The counter-part of parallel composition at the level of interfaces is the product operator $\otimes$. Product

$C \otimes D$ is the least interface capturing all possible pairwise parallel compositions of implementations of the two interfaces C and D:

$$C \otimes D \;=\; \min\{X | \forall M, N,\; M \models C \text{ and } N \models D \text{ implies } M \times N \models X\}$$

In essence, produtc $C \otimes D$ is the strongest interface capturing all possible parallel compositions of implementations of C and D. Hence, the product operator allows for independent implementability, meaning that, given a decomposition $C_1 \otimes C_2 \leq D$, interfaces C_1 and C_2 can be realized independently. Given any $M_1 \models C_1$ and any $M_2 \models C_2$, $M_1 \times M_2 \models D$.

Assume/guarantee contract reasoning, component reuse and incremental design methods call for a third composition operator. Indeed, an assume/guarantee contract (A, G) captures all implementations that satisfy G under the assumption that A holds. If A and G are logical propositional formulae, then the contract should be interpreted as the logical implication $A \Rightarrow G$. This explains why in the language theoretic assume/guarantee contracts developed for the SPEEDS project [30, 31], a contract is identified to its maximal implementation $M_{(A,G)} = G \cup \neg A$. In the case of modal interfaces, contract (A, G) should be understood as the residual (or quotient) modal interface $(G \otimes A)/A$. This can be generalized by assuming that interface theories admit a residuation operator $/$ with the following property:

$$C/D = \max\{X | D \otimes X \leq C\}$$

Thanks to this operator, one can kill two birds with one stone. Indeed, residuation can be applied in the context of component reuse. Assume an existing design M is to be reused in order to realize interface C. For complexity reasons, but also perhaps in order to protect intellectual property, M is only known through its more abstract interface D. In essence, M is an arbitrary implementation such that $M \models D$. The designer has to complete his design with an implementation N that acts as an adapter of M so that $M \times N \models C$. Since M is not revealed to the designer, his only resort is to reduce the design problem to the construction of a N such that $N \models C/D$.

In a similar way residuation is instrumental to incremental design methods where a system-level interface C_0 is realized by a set of component-level interfaces $D_1 \ldots D_n$, resulting from a sequence of design steps:

$$\begin{array}{rcl} C_1 & = & C_0/D_1 \\ C_2 & = & C_1/D_2 \\ & \vdots & \\ C_{n-1} & = & C_{n-2}/D_{n-1} \\ D_n & = & C_{n-1} \end{array}$$

This completes the abstract definition of an interface theory: Satisfaction, refinement, product, conjunction and residuation are the five main constituents of an interface theory. There are however many possible instances of interface theories.

1.2.4 A variety of interface theories

We have explored four of interface theories, namely language theoretic assume/guarantee contracts [28], probabilistic assume/guarantee contracts [83, 81], modal interfaces [151, 152, 153] (presented in Chapter 2) and constraint Markov chains [57] (presented in Chapter 3).

Inspired by the methodological requirements and the contract-based modeling formalism developed in the realm of the SPEEDS European project (2006–2010), the modal interface theory unifies R. Alur's and Th. Henzinger's Interface Automata [78] and of J.-B. Raclet's Modal Specifications [150, 152]. This chapter is based on a paper to appear in Fundamenta Informaticae [153].

An implementation of the modal interface algebra is currently being developed. The InterSMV tool extends the syntax of the NuSMV model-checker and is based on symbolic (MTBDD) representations of the set of realizations of a modal interface. InterSMV allows to verify (check for the consistency, satisfaction or refinement) compositions of modal-interfaces or assume/guarantee contracts

Constraint Markov chains are an attempt to transpose the principles of interface theories to the stochastic setting of discrete time Markov chains. This is achieved by considering a generalization of Markov Chains, where probability distributions are not given by extension, but rather, as solutions of a set of constraints.

Remark that the latter theory is not a proper interface theory, as the existence of residuals is an open problem. Indeed, it is not possible to mimic the construction of residuals used in modal interfaces and the construction of constraint Markov chain residuals seems a forlorn hope.

Publications related to this track: [57], [153], [83], [81],[152], [150], [28], [117], [31], [56], [80], [151], [82], [24]

1.3 Analysis and design of heterogeneous systems

This track contributes to the extension along two orthogonal dimensions of the well-established synchronous programming paradigm. Research in this track has been fueled by the challenge embedded system designers have to face in order to reduce design costs: The use of a single paradigm from system level engineering, downto the deployment of software over a distributed asynchronous architecture.

Synchronous programming has proved to be an appropriate paradigm for the design and implementation of control and signal processing hardware/software for a broad range of applications including plant supervision, transport vehicle control, automated production systems and consumer electronics. Howether, It suffers from a very significant shortcoming: The compilation of synchronous programs into distributed software leads to slow and inefficient implementations. The root cause of this inefficiency comes from the need to recreate synchrony, at the expense of a high communication burden. However, synchrony is nothing more than an abstraction that helps engineers to write correct programs and understand how their programs behave. Relaxing synchrony is often a necessary step towards efficient distributed implementations. Chapter 4 presents

a theory where both synchronous systems, and particular asynchronous systems (so-called weak-synchronous systems) can be expressed, combined, analyzed and transformed. Using the properties of weak-endochrony and weak-isochrony, we have characterized a decidable class of synchronous systems that can be desynchronized safely, without any need for additional communication. This work generalizes lattency insensitive circuits [59, 165] and embraces a larger class of systems, with improved asynchrony and less costly communication.

Synchronous programming can be applied earlier at system level design, when a model of the system needs to be build and analyzed. Hybrid modeling plays an important role at this stage of design. Control systems have to be analyzed in the context of a continuous mechanical/electrical/hydraulic environment. In this context, synchrony is implied by the laws of physics. This explains why hybrid modelers such as Simulink [1] or Scicos [58] are based on this paradigm. Unfortunately, some of them suffer from a lack of formal semantics and designers have to cope with the extreme sensitivity of simulation tools to the correct parametrization of the differential equation solvers. However, our focus is on another important aspect of hybrid system's semantics: Synchrony and causality between discrete events. Indeed, industry standard tools such as Simulink or Modelica do not handle event simultaneity very well. This leads to simulation artifacts which prediction and analysis actually require an in-depth knowledge of the hybrid compiler internals. A constructive semantics for hybrid systems, proceeding by successive infinitesimal time steps, is presented in Chapter 5. It is based on non-standard analysis, where infinitely many non-standard real numbers are infinitely close to any given real number. This work is not the first attempt to use non-standard analysis in the context of timed or hybrid systems [162, 40] or to analyze causality and synchrony in reactive systems [95]. However, it is the first time a non-standard semantics is used to address hybrid language design and compilation issues.

Publications related to this track: [30], [147], [143], [23], [19], [20], [140], [74], [141], [17], [18], [21], [168], [142], [22]

1.4 Synthesis and control of concurrent systems

We have explored various techniques to synthesize optimal networks of communicating automata. One of them is based on linear-algebraic Petri-net synthesis algorithms, implemented in the SYNET tool [9]. Based on this journal article, Chapter 6 details how distributed protocols can be synthesized using Petri-net synthesis. The key idea is to decompose the synthesis problem in two steps: Given the specification of a protocol as a finite automaton, (i) synthesize a distributable net, that is a Petri net where places and transitions are mapped to locations of a distributed architecture, and (ii) derive a set of communicating automata from the distributable net. While the second step is automatic and straightforward, the first step is in essence a computer assisted design task, where the distributed Petri-net synthesis algorithm helps the designer to refine the protocol specification into a graph isomorphic to the marking graph of a distributable net. The existence and

1. http://www.mathworks.com/products/simulink/

the automated computation of such a refinement is largely an open problem. See Section 6.6.2 for a discussion of the issue.

We have explored the application of control-synthesis techniques in the context of information system security. We have considered the problem of optimal control so that secret information (a language of system trajectories) are concealed within a system under partial observation from a set of agents (potential attackers), meaning that no agent can gain knowledge about these secrets [11]. A system capable of concealing secret informations is said to be opaque. Opacity has been introduced by Mazaré and coauthors in [133, 47], where they consider the verification of opacity properties. We have followed a different path, by considering the enforcement of concurrent opacity properties as a supervisory control problem, where the issue is to compute, an optimal control enforcing a set of opacity properties on a system [7, 8, 11]. This is detailed in Chapter 7. This chapter is based on our JDEDS paper [11].

Publications related to this track: [55], [109], [11], [9], [50], [51], [159], [160], [8], [15], [54], [52], [7], [53], [49]

1.5 organization of the document

The document is composed of three independent parts: Part I on interface theories, Part II on heterogeneous system modeling, and Part III on supervisory control. Most chapters are independent and self-contained. Only Chapter 3 is not fully self-contained and is best understood after reading Chapter 2.

Part I

Interface Theories for System Design

Chapter 2

Modal interfaces

Résumé : *Les méthodes d'ingénierie des systèmes employées dans l'industrie ont connu ces dernières années des évolutions sensibles, notament en adaptant des techniques de conception par composants issues de l'ingénierie du logiciel. Ces méthodes offrent d'une part une plus grande flexibilité du processus de conception, et permettent d'autre part des activités de conception parallèles. Les concepteurs sont alors amenés à modéliser, en cours de conception, les hypothèses faites sur l'environnement d'un composant et par la suite, à vérifier le bienfondé de ces hypothèses. Il est donc primordial de pouvoir modéliser non seulement les propriétés comportementales du systèmes à réaliser, mais aussi les hypothèses sous lesquelles ces propriétés sont énoncées. Ceci permet de repousser ultérieurement la réalisation des composants, pour permettre une analyse de la composition de ces composants, reposant uniquement sur la manipulation d'abstractions de composants, appelées interfaces.*

Les concepteurs sont donc amenés à manipuler deux sortes d'objets, (i) des réalisations (de composants) et (ii) leurs interfaces. Réalisations et interfaces sont reliées par une relation de satisfaction, qui définit quelles sont les réalisations correctes d'une interface. La relation de satisfaction permet de définir une relation de préordre sur les interfaces, dite relation de raffinement. La nécessité de pouvoir composer les modèles de conception implique que les interfaces doivent pouvoir être combinées selon plusieurs opérateurs de composition : la conjonction des interfaces est requise pour pouvoir combiner les différents points de vues d'un même composant. La composition parallèle et la définition des architectures requiert que les interfaces puissent être composées à l'aide d'une operation de produit. Enfin, une opération de résiduation des interfaces est requise pour pouvoir (i) définir des contrats hypothèses/garanties et (ii) recourir à des méthodes de conception incrémentale, où il s'agit de réduire par étapes succéssives un problème de conception en une suite de problèmes de moindre difficulté.

Il existe de nombreuses instances de théories d'interfaces. L'une d'entre elle est présentée dans ce chapitre, les interfaces modales, qui généralisent à la fois les spécifications modales, les contrats hypothèse/garantie et les automates d'interfaces. Ce chapitre aborde de manière approfondie les propriétés algébriques générales des théories d'interfaces , celles des interfaces modales, les rela-

tions avec d'autres formalismes apparentés (contrats et automates d'interfaces) et les implications quant à leur utilisation comme outil support d'une méthode de conception en ingénierie des systèmes.

2.1 Introduction

Nowadays, systems are tremendously big and complex, resulting from the assembling of several components. These many components are in general designed by teams, working independently but with a common agreement on what the interface of each component should be. As a consequence, mathematical foundations that allow to reason at the abstract level of interfaces is a very active research area. According to our understanding of industrial needs, an interface theory is at least subject to the following requirements:

1. *Satisfaction and satisfiability are decidable.* Interfaces should be seen as specifications whose models are its possible implementations. It should thus be decidable whether an interface admits an implementation and whether a given component implements a given interface.
2. *Refinement entails substituability.* Refinement allows one to replace, in any context, an interface by a more detailed version of it. Refinement should entail substituability of interface implementations, meaning that every implementation satisfying a refinement also satisfies the larger interface. For the sake of controlling design complexity, it is desirable to be able to decide whether there exists an implementation satisfying two different interfaces. This is called *shared refinement*.
3. *Encompassing interfaces with dissimilar alphabets.* Complex systems are built by combining subsystems possessing dissimilar alphabets for referencing ports and variables. It is thus important to properly handle those different alphabets when combining interfaces.
4. *Composition supports independent design.* The interface theory should also provide a combination operator on interfaces, reflecting the standard composition of implementations by, *e.g.* parallel product. This operation must be associative and commutative to guarantee independence in the development. Depending on the model, a notion of compatibility for composition may also be considered, i.e., there can be cases where two systems cannot be composed.
5. *Interfaces are closed under conjunction.* It is the current practice that early requirements capture relies on Doors Databases, or even Excel files containing possibly many textual requirements. Under the current practice, little formal support exists to handle them. Moving ahead can be envisioned by formalizing the notation used for individual requirements. This can be, e.g., achieved by relying on so-called semi-formal languages [48], whose sentences are translatable into predefined behavioral patterns according to several viewpoints. Alternatively, graphical scenario languages could be considered [73, 110]. Composing viewpoints

within a given subsystem calls for the support of the concept of *conjunction* of interfaces in order to combine requirements and check their satisfiability.

6. *Interface quotient supports incremental design and component reuse.* Last but not least, a quotienting operation, dual to composition is crucial to perform incremental design. Consider a desired global specification and the specification of a preexisting component; the quotient specification describes the part of the global specification that remains to be implemented.

Building good interface theories has been the subject of intensive studies (see e.g., [100, 78, 41, 90, 94, 76, 79]). In this chapter we will concentrate on two models: (1) *interface automata* [78] and (2) *modal specifications* [119]. Interface automata is a game semantics based variation of input/output automata which deals with open systems, their refinement and composition, and put the emphasis on interface compatibility. Modal specifications is a language theoretic account of a fragment of the modal mu-calculus logic [92] which admits a richer composition algebra with product, conjunction, and residuation operators.

In interface automata [78], an interface is represented by an input/output automaton [130], *i.e.*, an automaton whose transitions are labeled with *input* or *output* actions. The semantics of such an automaton is given by a two-player game: an *Input* player represents the environment, and an *Output* player represents the component itself. Interface automata do not encompass any notion of model, because one cannot distinguish between interfaces and implementations. Alternatively, properties of interfaces are described in game-based logics, *e.g.*, ATL [1], with a high-cost complexity.

Refinement between interface automata corresponds to the alternating refinement relation between games [2], *i.e.*, an interface refines another one if its environment is more permissive whereas its component is more restrictive. Shared refinement is defined in an ad-hoc manner [85] for a particular class of interfaces [63]. Contrary to most interfaces theories, the game-based interpretation offers an *optimistic* treatment of composition: two interfaces can be composed if there exists at least one environment (i.e., one strategy for the Input player) in which they can interact together in a safe way (i.e., whatever the strategy of the Output player is). This is referred as compatibility of interfaces.

Modal specifications [119] correspond to *deterministic modal automata*, i.e., automata whose transitions are typed with *may* and *must* modalities. A modal specification thus represents a set of models; informally, a must transition is available in every component that implements the modal specification, while a may transition needs not be. The components that implement modal specifications are prefix-closed languages, or equivalently deterministic automata.

Satisfiability of modal specifications is decidable. Refinement between modal specifications coincides with models inclusion. Since components can be seen as specifications where all transitions are typed must (all possible implementation choices have been made), satisfaction is also expressed via alternating simulation. Conjunction is effectively computed via a product-like construction. Combination of modal specifications, handling synchronization products *à la* Arnold and

Nivat [6], and the dual quotient combinators can be efficiently handled in this setting [149, 148].

Interface automata and modal specifications are incomparable models as *must*, *may* and *input*,*output* have orthogonal meanings. Both models have advantages and disadvantages:

- Interface automata is a model that allows to make assumptions on the environment, which is mainly useful to derive a rich notion for composition. Unfortunately, the model is incomplete as conjunction, and quotient are not defined for this game-based model.
- Modal specification is a rich language algebra model on which most of requirements for a good interface theory can be considered. Unfortunately, *may* and *must* modalities are not sufficient to derive a rich notion for composition including compatibility.

It is thus worth considering unification of the frameworks of interface automata and modal specifications. A first attempt was made by Larsen et al. [122, 138] who considered *modal interfaces* that are modal specifications whose actions are also typed in *input* or *output* attributes. Larsen et al. have proposed a product-like construction allowing to address compatibility of modal interfaces. Nevertheless contrary to what is claimed by the authors, this composition operator in [122, 138] is not monotone with respect to the refinement of modal specifications. This fails to ensure that two *compatible* interfaces may be implemented separately.

The present chapter adds a new stone to the cathedral of results on interface theories by (1) correcting the modal interface composition operator presented in [122, 138], (2) drawing a complete picture of the modal interface algebra , and (3) pushing even further the comparison between interface automata, modal automata and modal specifications and modal interfaces.

The rest of the chapter is organized as follows. In Sections 2.2 and 2.3 we recap the theory for modal specifications and interface automata, respectively. In Section 2.4, we present the complete theory for modal interfaces and correct the error in [122, 138]. Finally, in Section 2.5, we draw our conclusion and discuss future extensions for the model of modal interfaces.

2.2 Modal specifications

This section overviews existing results for modal specifications. We start by introducing the framework, then we discuss the extension to several alphabets and study the notions of refinement and implementation. Finally, we present results on combining modal specifications.

2.2.1 The Framework

Following our previous work [150], we will define modal specifications in term of languages, knowing that they can also be interpreted as deterministic automata whose transitions are typed with *may* and *must* modalities. We propose the following definition.

Definition 2.2.1 *A* modal specification *is a tuple* $\mathcal{S} = (A, must, may)$*, where* A *is a finite alphabet and:*

$$must, may : A^* \mapsto 2^A$$

are partial functions satisfying the following consistency *condition:*

$$must(u) \quad \subseteq \quad may(u). \tag{2.1}$$

The fact that $a \in may(u)$ means that action a is allowed after the trace u whereas $a \in must(u)$ indicates that a is required after u. By negation, $a \notin may(u)$ means that a is disallowed after u. The latter is often written $a \in mustnot(u)$. The condition (2.1) naturally imposes that every required action is also allowed. We shall sometimes write $A_{\mathcal{S}}, may_{\mathcal{S}}$, and $must_{\mathcal{S}}$ to refer to the entities involved in the definition of $\mathcal{S}$.

When composing specifications, discrepancies between the modal informations carried out by the specifications may appear. We then consider *pseudo-modal specifications*, denoted ${}^{p}\mathcal{S}$; they are triples satisfying Definition 2.2.1 with the exception of (2.1). For ${}^{p}\mathcal{S}$ a pseudo-modal specification, a word $u \in A^*$ is called *consistently specified* in ${}^{p}\mathcal{S}$ if it satisfies (2.1) and *inconsistent* otherwise; modal specifications correspond exactly to the subclass of *consistent* pseudo-modal specifications, that is pseudo-specifications such that every $u \in A^*$ is consistently specified.

A similar approach has been developed in [128] for a *non-modal* process algebraic framework in which a dedicated predicate is used to model inconsistent processes.

For ${}^{p}\mathcal{S} = (A, must, may)$ a pseudo-modal specification, the *support* of ${}^{p}\mathcal{S}$ is the least *prefix-closed* language $\mathcal{L}_{{}^{p}\mathcal{S}}$ such that: $(i)\ \epsilon \in \mathcal{L}_{{}^{p}\mathcal{S}}$, where ϵ denotes the empty word; and $(ii)\ u \in \mathcal{L}_{{}^{p}\mathcal{S}}$ and $a \in may(u)$ imply $u.a \in \mathcal{L}_{{}^{p}\mathcal{S}}$.

2.2.2 Multiple Alphabets

Large systems are composed of many subsystems possessing their own alphabets for ports and variables. The way those different alphabets are handled when combining subsystems requires some care.

We start with a series of definitions on languages. Let A and C be two alphabets such that $A \subseteq C$. For $v \in C^*$, the *projection* of v on A (denoted $\mathbf{pr}_A(v)$) is the word over A obtained from v by erasing all symbols that do not belong to A. Let $\mathcal{L}$ be a language over A, the *extension* of $\mathcal{L}$ to C is the language $\bar{C}\mathcal{L} = \{v \in C^* \mid \mathbf{pr}_A(v) \in \mathcal{L}\}$.

Definition 2.2.2 *The* shuffle product $\mathcal{L}_1 \times \mathcal{L}_2$ *of two languages* $\mathcal{L}_1 \subseteq A_1^*$ *and* $\mathcal{L}_2 \subseteq A_2^*$ *is given by*

$$\mathcal{L}_1 \times \mathcal{L}_2 = \bar{A}(\mathcal{L}_1) \cap \bar{A}(\mathcal{L}_2), \text{ where } A = A_1 \cup A_2.$$

In modal automata, one has to consider two alphabet extensions: the weak and the strong extension. We shall see that the extension in use will depend on the operation that is performed on modal specification [150].

Definition 2.2.3 (weak and strong extensions) *Let* ${}^{p}\mathcal{S} = (A, must_{{}^{p}\mathcal{S}}, may_{{}^{p}\mathcal{S}})$ *be a pseudo-modal specification and let* $C \supseteq A$.

1. *The* weak extension *of* ${}^p\mathcal{S}$ *to* C *is the pseudo-modal specification* ${}^p\mathcal{S}_{\Uparrow C} = (C, must, may)$ *such that* $\forall v \in C^*$*:*

$$\begin{cases} must(v) & = & must_{{}^p\mathcal{S}}\left(\mathbf{pr}_A(v)\right) \\ may(v) & = & may_{{}^p\mathcal{S}}\left(\mathbf{pr}_A(v)\right) \cup (C - A). \end{cases}$$

2. *The* strong extension *of* ${}^p\mathcal{S}$ *to* C *is the pseudo-modal specification* $\bar{C}{}^p\mathcal{S} = (C, must, may)$ *such that* $\forall v \in C^*$*:*

$$\begin{cases} must(v) & = & must_{{}^p\mathcal{S}}\left(\mathbf{pr}_A(v)\right) \cup (C - A) \\ may(v) & = & may_{{}^p\mathcal{S}}\left(\mathbf{pr}_A(v)\right) \cup (C - A). \end{cases}$$

It is easy to show that $\mathcal{L}_{(\mathcal{S}_{\Uparrow C})} = \mathcal{L}_{(\bar{C}\mathcal{S})} = \bar{C}(\mathcal{L}_{\mathcal{S}})$.

2.2.3 Implementation and refinement

In this section, we study the concepts of *implementation*, *refinement* and *consistency*. We start with implementation, also called *model*.

Definition 2.2.4 (implementation) *Let* ${}^p\mathcal{S} = (A, must, may)$ *be a pseudo-modal specification.*

1. **Equal Alphabets:** *A prefix-closed language* $\mathcal{I} \subseteq A^*$ *is an* implementation *of* ${}^p\mathcal{S}$ *, denoted by* $\mathcal{I} \models {}^p\mathcal{S}$*, if* $\forall u \in \mathcal{I}, must(u) \subseteq \mathcal{I}_u \subseteq may(u)$*, where* $\mathcal{I}_u = \{a \in A \mid u.a \in \mathcal{I}\}$.
2. **Extended Alphabets:** *For* $C \supseteq A$*, a prefix-closed language* $\mathcal{I} \subseteq C^*$ *is a* weak implementation *of* ${}^p\mathcal{S}$*, written* $\mathcal{I} \models_w {}^p\mathcal{S}$*, iff* $\mathcal{I} \models {}^p\mathcal{S}_{\Uparrow C}$ *holds; it is a* strong implementation *of* ${}^p\mathcal{S}$*, written* $\mathcal{I} \models_s {}^p\mathcal{S}$*, iff* $\mathcal{I} \models {}^p\mathcal{S}_{\uparrow C}$ *holds.*

Modal specifications are equivalent to the fragment of the μ-calculus called the conjunctive ν-calculus [92]. Hence, a model for a modal specification is a model for the formula represented by the specification.

Satisfaction can be related to consistently specified words:

Lemma 2.2.5 *If* $\mathcal{I} \models {}^p\mathcal{S}$*, then* $\mathcal{I} \subseteq \mathcal{L}_{{}^p\mathcal{S}}$ *holds and every word of* $\mathcal{I}$ *is consistently specified in* ${}^p\mathcal{S}$*. Similarly, if* $\mathcal{I} \models_w {}^p\mathcal{S}$ *or* $\mathcal{I} \models_s {}^p\mathcal{S}$*, then* $\mathcal{I} \subseteq \bar{C}(\mathcal{L}_{{}^p\mathcal{S}})$ *holds and for every word* $v \in C^*$ *of* $\mathcal{I}$*,* $\mathbf{pr}_A(v)$ *is consistently specified in* ${}^p\mathcal{S}$*.*

We now switch to the case of modal refinement which extends in a natural manner the classical notion of bisimulation on automata. We first consider the case where specifications are defined over the same alphabet:

Definition 2.2.6 *Let ${}^p\mathcal{S}_1 = (A, must_1, may_1)$ and ${}^p\mathcal{S}_2 = (A, must_2, may_2)$ be two pseudo-modal specifications then ${}^p\mathcal{S}_1$* refines *${}^p\mathcal{S}_2$, denoted ${}^p\mathcal{S}_1 \leq {}^p\mathcal{S}_2$, iff for all $u \in \mathcal{L}_{{}^p\mathcal{S}_1}$:*

$$\begin{array}{rcl} may_1(u) & \subseteq & may_2(u) \\ must_1(u) & \supseteq & must_2(u). \end{array}$$

It can be shown that refinement is a preorder relation which implies the inclusion of supports. As a consequence, any two modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_1 \leq \mathcal{S}_2 \leq \mathcal{S}_1$ have equal supports $\mathcal{L} = \mathcal{L}_{\mathcal{S}_1} = \mathcal{L}_{\mathcal{S}_2}$ and moreover, for all $u \in \mathcal{L}$, $may_1(u) = may_2(u)$ and $must_1(u) = must_2(u)$. Thus equivalent modal specifications differ only outside of their support; a unique representant $\mathcal{S} = (A, must, may)$ of equivalence classes of modal specifications can be defined by assuming that for all $u \notin \mathcal{L}_{\mathcal{S}}$, $must(u) = \emptyset$ and $may(u) = A$. Under this assumption, modal refinement is a *partial order relation* on modal specifications. In the following, only modal specifications in this *canonical* form are considered.

Definition 2.2.7 *Let ${}^p\mathcal{S}_1 = (A_1, must_1, may_1)$ and ${}^p\mathcal{S}_2 = (A_2, must_2, may_2)$ be two pseudo-modal specifications with $A_1 \supseteq A_2$ then ${}^p\mathcal{S}_1$* weakly refines *${}^p\mathcal{S}_2$ (which is denoted ${}^p\mathcal{S}_1 \leq_w {}^p\mathcal{S}_2$), iff ${}^p\mathcal{S}_1 \leq {}^p\mathcal{S}_{2\Uparrow A_1}$, and it* strongly refines *${}^p\mathcal{S}_2$, written ${}^p\mathcal{S}_1 \leq_s {}^p\mathcal{S}_2$, iff ${}^p\mathcal{S}_1 \leq {}^p\mathcal{S}_{2\uparrow A_1}$.*

A pseudo-modal specification can be reduced into a modal specification with preservation of its semantic:

Theorem 2.2.8 (consistency) *Either a pseudo-modal specification ${}^p\mathcal{S}$ has no model, or there exists a modal specification $\rho({}^p\mathcal{S})$ having the same alphabet of actions such that $\rho({}^p\mathcal{S})$ possesses the same set of weak and strong implementations:*

$$\begin{array}{rcl} \mathcal{I} \models_w {}^p\mathcal{S} & \Leftrightarrow & \mathcal{I} \models_w \rho({}^p\mathcal{S}) \\ \mathcal{I} \models_s {}^p\mathcal{S} & \Leftrightarrow & \mathcal{I} \models_s \rho({}^p\mathcal{S}) \end{array}$$

We shall call $\rho({}^p\mathcal{S})$ the reduction of ${}^p\mathcal{S}$. The detailed construction of $\rho({}^p\mathcal{S})$ can be found in [150]. We let $\perp$ be a particular modal specification that admits no model and let $\mathcal{L}_\perp$ be the empty set.

We conclude the section with the following theorem that relates refinement and implementation.

Theorem 2.2.9 (implementation and refinement)

1. *Weak and strong implementation and refinement are related as follows: $\models_s \subseteq \models_w$ and $\leq_s \subseteq \leq_w$.*
2. *Weak and strong modal refinement are both sound and complete w.r.t. weak and strong thorough refinement, respectively:*

$$\begin{array}{rcl} \mathcal{S}_2 \leq_w \mathcal{S}_1 & \Leftrightarrow & \{\mathcal{I} \mid \mathcal{I} \models_w \mathcal{S}_2\} \subseteq \{\mathcal{I} \mid \mathcal{I} \models_w \mathcal{S}_1\} \\ \mathcal{S}_2 \leq_s \mathcal{S}_1 & \Leftrightarrow & \{\mathcal{I} \mid \mathcal{I} \models_s \mathcal{S}_2\} \subseteq \{\mathcal{I} \mid \mathcal{I} \models_s \mathcal{S}_1\}. \end{array}$$

As already noticed, modal specifications are equivalent to deterministic modal automata. When allowing for nondeterminism, the theorem above does not hold as modal refinement is no more complete [121].

2.2.4 Operations on modal specifications

Consider two modal specifications $\mathcal{S}_1 = (A_1, must_1, may_1)$ and $\mathcal{S}_2 = (A_2, must_2, may_2)$, we now define their *conjunction*, *parallel product* and *quotient*. We proceed in two steps: we first define these operations when $A_1 = A_2$; the case of different alphabets is then handled by performing a preliminary step of alphabet equalization.

In [150], we argued that alphabet equalization must be different depending on the considered operation. Such an extension must be *neutral*, meaning that it should not constrain what other interfaces may want to require regarding these extra actions.

Conjunction When $A_1 = A_2$, the *conjunction* $\mathcal{S}_1 \wedge \mathcal{S}_2 = \rho(\mathcal{S}_1 \& \mathcal{S}_2)$ where $\mathcal{S}_1 \& \mathcal{S}_2$ is defined by:

$$\begin{array}{rcl} may_{\mathcal{S}_1 \& \mathcal{S}_2}(u) & = & may_1(u) \cap may_2(u) \\ must_{\mathcal{S}_1 \& \mathcal{S}_2}(u) & = & must_1(u) \cup must_2(u). \end{array} \tag{2.2}$$

Observe that it is not guaranteed that $\mathcal{S}_1 \& \mathcal{S}_2$ satisfies (2.1). Hence, we use theorem 2.2.8 and apply the reduction operation ρ in order to obtain a modal specification.

For the general case where $A_1 \neq A_2$, the definition above is applied after an equalization step: $\mathcal{S}_1 \wedge \mathcal{S}_2 = \mathcal{S}_{1 \Uparrow A} \wedge \mathcal{S}_{2 \Uparrow A}$, with $A = A_1 \cup A_2$.

Theorem 2.2.10

$$\mathcal{I} \models_w \mathcal{S}_1 \wedge \mathcal{S}_2 \;\Leftrightarrow\; \mathcal{I} \models_w \mathcal{S}_1 \text{ and } \mathcal{I} \models_w \mathcal{S}_2.$$

The conjunction between $\mathcal{S}_1$ and $\mathcal{S}_2$ is exactly their *greatest lower bound* for the weak refinement relation: $\mathcal{S}_1 \wedge \mathcal{S}_2$ is the greatest specification that weakly refines both $\mathcal{S}_1$ and $\mathcal{S}_2$.

A current practice in the design of a component is to give several specifications, each of them describing a particular requirement. The conjunction of these specifications, enables to check the consistency of these requirements, by deciding satisfiability.

Parallel product When $A_1 = A_2$, the *parallel product* $\mathcal{S} = \mathcal{S}_1 \otimes \mathcal{S}_2$ is defined by:

$$\begin{array}{rcl} may_{\mathcal{S}}(u) & = & may_1(u) \cap may_2(u) \\ must_{\mathcal{S}}(u) & = & must_1(u) \cap must_2(u). \end{array} \tag{2.3}$$

The product of two modal specifications always satisfy the consistency condition. Hence, no reduction is needed. For the general case where $A_1 \neq A_2$, the definition above is applied after an equalization step: $\mathcal{S}_1 \otimes \mathcal{S}_2 = \bar{A}\mathcal{S}_1 \otimes \bar{A}\mathcal{S}_2$.

In an interface theory, it is desirable to be able to develop components in isolation and then to compose them as expected. This is ensured by the product operation as stated with the following theorem.

Theorem 2.2.11

1. *If* $\mathcal{S}'_1 \leq_s \mathcal{S}_1$ *and* $\mathcal{S}'_2 \leq_s \mathcal{S}_2$*, then* $\mathcal{S}'_1 \otimes \mathcal{S}'_2 \leq_s \mathcal{S}_1 \otimes \mathcal{S}_2$.
2. *If* $\mathcal{I}_1 \models_s \mathcal{S}_1$ *and* $\mathcal{I}_2 \models_s \mathcal{S}_2$*, then* $\mathcal{I}_1 \times \mathcal{I}_2 \models_s \mathcal{S}_1 \otimes \mathcal{S}_2$.
3. *Regarding supports:* $\mathcal{L}_{\mathcal{S}_1 \otimes \mathcal{S}_2} = \mathcal{L}_{\mathcal{S}_1} \times \mathcal{L}_{\mathcal{S}_2}$.

Strong refinement has to be used when enlarging the alphabet, as the product is not monotonic with respect to the weak refinement [150].

Residuation/quotient The operation of *residuation*, also called *quotient*, is the adjoint of product. Intuitively, the quotient enables to describe a part of a global specification assuming another part is already realized by some component. If $A_1 = A_2$, then the *pseudo*-quotient ${}^p\mathcal{S} = \mathcal{S}_1 /\!\!/ \mathcal{S}_2$ is defined by:

$$\begin{array}{lll}
a \in may_{{}^p\mathcal{S}}(u) \cap must_{{}^p\mathcal{S}}(u) & \text{if} & a \in must_1(u) \\
 & \text{and} & a \in must_2(u) \\
a \in must_{{}^p\mathcal{S}}(u) \setminus may_{{}^p\mathcal{S}}(u) & \text{if} & a \in must_1(u) \\
 & \text{and} & a \notin must_2(u) \\
a \in may_{{}^p\mathcal{S}}(u) \setminus must_{{}^p\mathcal{S}}(u) & \text{if} & a \in may_1(u) \\
 & \text{and} & a \notin must_1(u) \\
a \in may_{{}^p\mathcal{S}}(u) \setminus must_{{}^p\mathcal{S}}(u) & \text{if} & a \notin may_1(u) \\
 & \text{and} & a \notin may_2(u) \\
a \notin may_{{}^p\mathcal{S}}(u) \cup must_{{}^p\mathcal{S}}(u) & \text{if} & a \notin may_1(u) \\
 & \text{and} & a \in may_2(u).
\end{array}$$

Due to the second rule, $\mathcal{S}_1 /\!\!/ \mathcal{S}_2$ may have inconsistently specified words. As a consequence, a reduction operation may be needed and the quotient of $\mathcal{S}_1$ by $\mathcal{S}_2$ is $\mathcal{S}_1/\mathcal{S}_2 = \rho(\mathcal{S}_1 /\!\!/ \mathcal{S}_2)$.
For the general case of two different alphabets, the definition above is applied after an alphabet equalization step: $\mathcal{S}_1 \,/\, \mathcal{S}_2 = \mathcal{S}_{1 \Uparrow A} \,/\, \bar{A}\mathcal{S}_2$.

We have the following theorems:

Theorem 2.2.12 *Let* $\mathcal{S}$*,* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *be modal specifications such that* $A_{\mathcal{S}_2} \supseteq A_{\mathcal{S}} \supseteq A_{\mathcal{S}_1}$*. We have*

$$\mathcal{S}_2 \leq_s \mathcal{S}/\mathcal{S}_1 \Leftrightarrow \mathcal{S}_1 \otimes \mathcal{S}_2 \leq_s \mathcal{S}.$$

Theorem 2.2.13 *Let* $\mathcal{S}$*,* $\mathcal{S}_1$ *be modal specifications and* $\mathcal{I}_2$ *a prefix-closed language such that* $A_{\mathcal{I}_2} \supseteq A_{\mathcal{S}} \supseteq A_{\mathcal{S}_1}$*, we have*

$$\mathcal{I}_2 \models_s \mathcal{S}/\mathcal{S}_1 \Leftrightarrow [\forall \mathcal{I}_1 : \mathcal{I}_1 \models_s \mathcal{S}_1 \Rightarrow \mathcal{I}_1 \times \mathcal{I}_2 \models_s \mathcal{S}].$$

2.3 Interface Automata

In [78], de Alfaro and Henzinger introduced *interface automata*, that are automata whose transitions are typed with *input* and *output* actions rather than with modalities. In this section, we briefly overview the theory of interface automata and refer the reader to [78, 75] for more details.

Definition 2.3.1 *An* interface automaton *is a tuple* $\mathcal{P} = (X, x_0, A, \rightarrow)$*, where* X *is the set of* states, $x_0 \in X$ *is the* initial state, A *is the alphabet of* actions, *and* $\rightarrow \subseteq X \times A \times X$ *is the transition relation.*

We decompose $A = A? \uplus A!$, where $A?$ is the set of inputs and $A!$ is the set of outputs. In the rest of the chapter, we shall often use $a?$ to emphasize that $a \in A?$ and $a!$ for $a \in A!$. Observe that if we consider deterministic interface automata, then we can propose a language-based definition similar to the one we gave for modal specifications.

The semantic of an interface automaton is given by a two-player game between: an *input* player that represents the environment (the moves are the input actions), and an *output* player that represents the component itself (the moves are the output actions). Input and output moves are in essence orthogonal to modalities. Interface automata are operational models, they do not encompass any notion of model, and thus neither satisfiability nor consistency, because one cannot distinguish between interfaces and components implementations. Alternatively, properties of interfaces are described in game-based logics, *e.g.*, ATL [1], with a high-cost complexity. Refinement between interface automata corresponds to the alternating refinement relation between games [2], *i.e.*, an interface refines another one if its environment is more permissive whereas its component is more restrictive. There is no notion of component reuse and shared refinement is defined in an ad-hoc manner [85].

The main advantage of the game-based approach appears in the definition of composition and *compatibility* between interface automata. Following [75], two interface automata are composable if they have disjoint sets of output actions compose by synchronizing on shared actions and interleave asynchronously all other actions.

Definition 2.3.2 (Product of interface automata) *Let* $\mathcal{P}_1 = (X_1, x_{01}, A_1, \rightarrow_1)$ *and* $\mathcal{P}_2 = (X_2, x_{02}, A_2, \rightarrow_2)$ *be two interface automata. The product between* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *is an interface automaton* $\mathcal{P}_1 \times \mathcal{P}_2 = (X, x_0, A, \rightarrow)$*, where*

- $X = X_0 \times X_1$*;*
- $x_0 = x_{01} \times x_{02}$*;*
- $A = A_1 \cup A_2$*, and* $A? = (A_1? \cup A_2?) \setminus ((A_1? \cap A_2!) \cup (A_2? \cap A_1!))$*, and* $A! = A_1! \cup A_2!$*;*
- $\rightarrow$ *is defined as follows:*
  - *For each action* $a \in A$ *such that* $a \notin A_1 \cap A_2$*, there exists a transition* $(x_1, y_1) \xrightarrow{a} (x_2, y_2)$ *iff there exists* $(x_1) \xrightarrow{a}_1 (x_2)$ *and* $y_1 = y_2$ *or* $(y_1) \xrightarrow{a}_2 (y_2)$ *and* $x_1 = x_2$*.*
  - *For each action* $a \in A_1? \cap A_2?$*, there exists a transition* $(x_1, y_1) \xrightarrow{a?} (x_2, y_2)$ *iff there exists* $(x_1) \xrightarrow{a?}_1 (x_2)$ *and* $(y_1) \xrightarrow{a?}_2 (y_2)$*.*

– *For each* $a \in (A_1? \cap A_2!) \cup (A_2? \cap A_1!)$*, there exists a transition* $(x_1, y_1) \xrightarrow{a!} (x_2, y_2)$ *iff there exists* $(x_1) \xrightarrow{a}_1 (x_2)$ *and* $(y_1) \xrightarrow{a}_2 (y_2)$.

Since interface automata are not necessarily input-enabled [1] (which allows to make assumptions on the environment), in the product $\mathcal{P}_1 \times \mathcal{P}_2$ of two interface automata $\mathcal{P}_1$ and $\mathcal{P}_2$, there may be *illegal states* where one of the automata may produce an output action that is also in the input alphabet of the other automaton, but is not accepted at this state. In most of existing models for interface theories that are based on an input output setting, the interfaces would be declared to be *incompatible*. This is a pessimistic approach that can be avoided by exploiting the game-based semantic. Indeed, the game semantic allows to propose an optimistic approach:

> "Two interfaces can be composed and are compatible if there is at least one environment where they can work together (i.e., where they can avoid the illegal states)."

Deciding whether there exists an environment where the two interfaces can work together is equivalent to checking whether the environment in the product of the interfaces has a strategy to always avoid illegal states. The set of states from which the environment has a strategy to avoid the illegal states whatever the component does can be recursively computed as follows.

Let $Illegal(\mathcal{P}_1, \mathcal{P}_2)$ is the subset of pairs $(x_1, x_2) \in X_1 \times X_2$ such that there exists

$$\begin{array}{rlll} \text{either an action } a \in A_1! \cap A_2? & \text{with} & x_1 \xrightarrow{a!}_1 \\ & \text{but not} & x_2 \xrightarrow{a?}_2 \\ \text{or an action } a \in A_2! \cap A_1? & \text{with} & x_2 \xrightarrow{a!}_2 \\ & \text{but not} & x_1 \xrightarrow{a?}_1 \end{array}$$

where $x \xrightarrow{a}$ means that $x \xrightarrow{a} y$ for some state y. If illegal states exist in the product $\mathcal{P}_1 \times \mathcal{P}_2$, there may still exist refinements of it that possess no illegal states. Such a refinement specifies how the use of the resulting product should be restricted in order to guarantee that illegal states cannot be reached. As proved in [78], such a largest refinement is obtained by backward pruning $\mathcal{P}_1 \times \mathcal{P}_2$ as follows. For $Y \subseteq X$, the set of states of $\mathcal{P}_1 \times \mathcal{P}_2$, let $pre_!(Y)$ be the subset $Z \subseteq X$ of states z such that $z \xrightarrow{a!} y$ for some $y \in Y$ and $a! \in A!$ (an output action of the product). Let $pre_!^0(Y) = Y$ and, for $k \geq 0$, $pre_!^{k+1}(Y) = pre_!\big(pre_!^k(Y)\big)$ and let $pre_!^*(Y) = \bigcup_k pre_!^k(Y)$.

The desired pruning consists in:

– Removing $pre_!^*(Illegal(\mathcal{P}_1, \mathcal{P}_2))$ from X, and
– Removing transitions to states in $pre_!^*(Illegal(\mathcal{P}_1, \mathcal{P}_2))$, and
– Removing unreachable states.

The result of applying the pruning to $\mathcal{P}_1 \times \mathcal{P}_2$ is denoted by

$$\mathcal{P}_1 \parallel \mathcal{P}_2,$$

1. Recall that a system is input-enabled if it can react to any input action in any moment.

and is called the *composition* of the two interface automata. $\mathcal{P}_1$ and $\mathcal{P}_2$ are called *compatible* if applying the pruning leaves the initial state [78].

We recall the two following theorems from [78] that show that interface automata support independent design and substituability.

Theorem 2.3.3 ([78]) *The composition operation is associative and commutative.*

Theorem 2.3.4 ([78]) *Let $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ be three interface automata. If $\mathcal{P}_2$ refines $\mathcal{P}_1$ and the set of shared actions $\mathcal{P}_2 \parallel \mathcal{P}3$ of is included in the set of shared actions of $\mathcal{P}_1 \parallel \mathcal{P}_3$, then $\mathcal{P}_2 \parallel \mathcal{P}_3$ refines $\mathcal{P}_1 \parallel \mathcal{P}_3$.*

Remark 2.3.5 *The operations between interface automata that have been defined so far do not require an explicit treatment of dissimilar alphabets as it is the case for modal specifications.*

2.4 On modal Interfaces

We now present the full theory for *modal interfaces*. Modal interfaces is an extension of modal specifications where actions are also typed with *input* and *output*. This addition allows to propose notions of composition and compatibility for modal specifications in the spirit of interface automata.

The first account on compatibility for modal interfaces was proposed in [122, 138]. In this section, we propose a full interface theory for modal interfaces, which includes composition, product, conjunction, and component reuse via quotient. Moreover, we show that the composition operator proposed in [122, 138] is incorrect and we propose a correction.

We shall start our theory with the definition of *profiles* which are used to type actions of modal specifications with *input* and *output*:

2.4.1 Profiles

For an alphabet of actions A, a *profile* is a function $\pi : A \mapsto \{?, !\}$, labeling actions with the symbols $?$ (for *inputs*) or $!$ (for *outputs*). We write "$a?$" to express that "$\pi(a) =?$", and similarly for the other case. We denote by $A?$ the set of $a \in A$ such that $\pi(a) =?$ and similarly for $A!$. We shall sometimes write by abuse of language, $\pi = (A?, A!)$.

We now discuss operations on profiles. We consider a profile $\pi_1 = (A_1?, A_1!)$ defined over A_1 and a profile $\pi_2 = (A_2?, A_2!)$ defined over A_2.

Product between profiles The composition between π_1 and π_2, which is defined iff $A_1! \cap A_2! = \emptyset$, is the $\pi = (A?, A!)$ such that

$$\pi_1 \otimes \pi_2 : \left\{ \begin{array}{lcll} A! & = & (A_1! & \cup \quad A_2!) \\ A? & = & (A_1? & \cup \quad A_2?) \setminus A! \end{array} \right.$$

Refinement between profiles Profile π_2 refines π_1 (denoted $\pi_2 \leq \pi_1$) iff $A_2 \supseteq A_1$ and both profiles coincide on A_1: $\forall a \in A_1, \pi_2(a) = \pi_1(a)$.

Conjunction between profiles The conjunction between π_1 and π_2 (denoted $\pi_1 \wedge \pi_2$) is the greatest lower bound of the profiles, whenever it exists. More precisely, the conjunction of profiles π_1 and π_2 is defined iff both profiles coincide on their common alphabet: $\forall a \in A_1 \cap A_2, \pi_1(a) = \pi_2(a)$. Whenever defined, the conjunction $\pi_1 \wedge \pi_2$ coincides with π_1 for every letter in A_1 and with π_2 for every letter in A_2.

Quotient between profiles The *quotient* $\pi_1 \,/\, \pi_2$ is defined as the adjoint of $\otimes$, if it exists, namely $\pi_1 \,/\, \pi_2 = \max\{\pi \mid \pi \otimes \pi_2 \leq \pi_1\}$. More precisely, $\pi_1 \,/\, \pi_2$ is defined if and only if $A_1 \supseteq A_2$ and $A_1! \supseteq A_2!$, and is then equal to the profile $\pi = (A?, A!)$ such that

$$\pi_1 \,/\, \pi_2 : \left\{ \begin{array}{lcl} A! & = & A_1! \setminus A_2! \\ A? & = & A_1? \setminus A_2?. \end{array} \right.$$

2.4.2 The framework of modal interfaces

We now formally introduce modal interfaces that are modal specification whose actions are also labeled with *input* and *output* attributes. We will consider the language representation in the spirit of [149, 150], while Larsen et al. followed the automata-based representation (the two representations are equivalent).

Definition 2.4.1 (Modal Interface) *A* modal interface *is a pair $\mathcal{C} = (\mathcal{S}, \pi)$, where $\mathcal{S}$ is a modal specification on the alphabet $A_\mathcal{S}$ and $\pi : A_\mathcal{S} \to \{?, !\}$ is a* profile.

A model for a modal interface is a tuple $(\mathcal{I}, \pi')$, where $\mathcal{I}$ is a prefix-closed language and π' is a profile for $\mathcal{I}$. We say that $(\mathcal{I}, \pi')$ *strongly implements* $(\mathcal{S}, \pi)$, written $(\mathcal{I}, \pi') \models_s (\mathcal{S}, \pi)$, if $\mathcal{I} \models_s \mathcal{S}$ and $\pi' \leq \pi$, and similarly for *weak implementation*. We say that $(\mathcal{S}_2, \pi_2) \leq_s (\mathcal{S}_1, \pi_1)$ if $\mathcal{S}_2 \leq_s \mathcal{S}_1$ and $\pi_2 \leq \pi_1$, with corresponding definition for weak refinement $\leq_w$. The *composition* of two models is the pair that results from the shuffle product $\times$ of their prefix-closed languages and of the product of their profiles.

2.4.3 Operations on modal interfaces

Operations on modal specifications directly extend to operations on modal interfaces. We have the following definition.

Definition 2.4.2 *Consider two modal interfaces $\mathcal{C}_1 = (\mathcal{S}_1,$ $\pi_1)$ and $\mathcal{C}_2 = (\mathcal{S}_2, \pi_2)$, and let $\star \in \{\wedge, \otimes, /\}$. If $\pi_1 \star \pi_2$ is defined, then*

$$\mathcal{C}_1 \star \mathcal{C}_2 = (\mathcal{S}_1 \star \mathcal{S}_2, \pi_1 \star \pi_2).$$

All the nice properties of modal specifications directly extend to modal interfaces.

Theorem 2.4.3 *Theorems 1 to 6 extend to modal interfaces.*

2.4.4 On compatibility for modal interfaces

In this section, we take advantage of profiles to define a notion of composition with compatibility issue for modal interfaces. We shall recap the solution proposed in [122, 138], then we shall show a counter example to Theorem 10 in [122] and then propose our correction. We first recap the translation from interface automata to modal interfaces, which will help to make the link between modalities and input or output actions.

From interface automata to modal interfaces

We recap the translation from interface automata to modal automata that has been proposed in [122]. In this section, we extend this translation to modal specification, the language-extension corresponding to modal automata.

We consider an interface automaton $\mathcal{P} = (X, x_0, A, \rightarrow)$. We assume $\mathcal{P}$ to be deterministic and we let $\mathcal{L}_\mathcal{P}$ denote the (prefix-closed) language defined by $\mathcal{P}$. The alphabet of $\mathcal{S}_\mathcal{P}$ is $A_{\mathcal{S}_\mathcal{P}} = A$ and modalities are defined for all $u \in A^*_\mathcal{P}$:

$$\begin{array}{lll}
a? \in must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u.a? \in \mathcal{L}_\mathcal{P} \\
a! \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u.a! \in \mathcal{L}_\mathcal{P} \\
a? \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \in \mathcal{L}_\mathcal{P} \\
 & \text{and} & u.a? \notin \mathcal{L}_\mathcal{P} \\
a! \notin may_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \in \mathcal{L}_\mathcal{P} \\
 & \text{and} & u.a! \notin \mathcal{L}_\mathcal{P} \\
a \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \notin \mathcal{L}_\mathcal{P}.
\end{array} \tag{2.4}$$

Theorem 1 of [122] shows that, with the above correspondence, alternating simulation for interface automata and modal refinement for modal interfaces coincide. Regarding supports, we have:

$$\mathcal{L}_{\mathcal{S}_\mathcal{P}} = \mathcal{L}_\mathcal{P} \uplus \{u.a?.v \mid u \in \mathcal{L}_\mathcal{P}, u.a? \notin \mathcal{L}_\mathcal{P}, v \in A^*_\mathcal{P}\}\,. \tag{2.5}$$

It is worth making some comments about this translation, given by formulas (2.4,2.5). Regarding formula (2.5), the supporting language $\mathcal{L}_{\mathcal{S}_\mathcal{P}}$ allows the environment to violate the constraints set on it by the interface automaton $\mathcal{P}$. When this happens—formally, the environment exits the alternating simulation relation—the component considers that the assumptions under which it was supposed to perform are violated, so it allows itself breaching its own promises and can perform anything afterward. One could also see the violation of assumptions as an exception. Then, $\mathcal{L}_{\mathcal{S}_\mathcal{P}}$ states no particular exception handling since everything is possible. Specifying exception handling then amounts to refining this modal interface.

Formula (2.4) refines (2.5) by specifying obligations. Case 1 expresses that the component *must* accept from the environment any input within the assumptions. Case 2 indicates that the component behaves according to best effort regarding its own outputs actions. Finally, cases 3 and 4 express that the violation of its obligations by the environment are seen as an exception, and that exception handling is unspecified and not mandatory.

The composition by Larsen et al. and the bug in Theorem 10 of [122]

We now consider the notion of compatibility for two Modal Interfaces $\mathcal{C}_1 = (\mathcal{S}_1, \pi_1)$ and $\mathcal{C}_2 = (\mathcal{S}_2, \pi_2)$ with $\mathcal{S}_1$ defined over A_1 and $\mathcal{S}_2$ defined over A_2. We assume that $\mathcal{C}_1$ and $\mathcal{C}_2$ do not share common output actions (which is the composability requirement similar to the one for interface automata). We first compute the product between $\mathcal{C}_1$ and $\mathcal{C}_2$ following Definition 2.4.3.

We then define $\mathit{Illegal}(\mathcal{C}_1, \mathcal{C}_2)$ to be the subset of words u belonging to the support of $\mathcal{C}_1 \otimes \mathcal{C}_2$, such that there exists

$$\begin{array}{rl} \text{either} & \text{an action } a \in A_1! \cap A_2? \\ & \text{with } a \in may_1(u_1) \setminus must_2(u_2) \\ \text{or} & \text{an action } a \in A_2! \cap A_1? \\ & \text{with } a \in may_2(u_2) \setminus must_1(u_1), \end{array} \tag{2.6}$$

where $u_1 = \mathbf{pr}_{A_1}(u)$ and similarly $u_2 = \mathbf{pr}_{A_2}(u)$. Getting rid of illegal runs is performed as follows. For U a set of words of Modal Interface $\mathcal{C}$, let $pre_!(U)$ be the set

$$pre_!(U) \;=\; \{v \in \mathcal{L}_{\mathcal{C}} \mid \exists a! \in may(v), v.a! \in U\}$$

Let $pre_!^0(U) = U$, and, for $k \geq 0$, $pre_!^{k+1}(U) = pre_!\big(pre_!^k(U)\big)$. Finally, let $pre_!^*(U) = \bigcup_k pre_!^k(U)$.

The composition of two modal interfaces is obtained from their product by removing states in $pre_!^*(U)$, following the approach outlined for interface automata. Two modal interfaces are compatible if the pruning with the illegal words do not remove the empty word. The composition between $\mathcal{C}_1$ and $\mathcal{C}_2$ is denoted $\mathcal{C}_1 \parallel \mathcal{C}_2$.

Theorem 10 in [122, 138] says that

> *"(Independent Implementability). For any two composable modal interfaces $\mathcal{C}_1$, $\mathcal{C}_2$ and two implementations $(\mathcal{I}_1, \pi_1)$ and $(\mathcal{I}_2, \pi_2)$. If $(\mathcal{I}_1, \pi_1) \leq \mathcal{C}_1$ and $(\mathcal{I}_2, \pi_2) \leq \mathcal{C}_2$, then it holds that $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \leq \mathcal{C}_1 \parallel \mathcal{C}_2$."*

The following example shows that Theorem 10 in [122, 138] is wrong.

Example 2.4.4 *Figure 2.1 depicts two Modal Interfaces $\mathcal{C}_1$ and $\mathcal{C}_2$; $may \setminus must$ actions are depicted using dashed arrows whereas solid arrows corresponds to $must$ actions. $\mathcal{I}_1$ and $\mathcal{I}_2$ are implementations of $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. Alphabets are indicated for each modal interface. Parallel composition according to [122] is named $[\mathcal{C}_1 \parallel \mathcal{C}_2]_0$. Word $c?.a!$ is illegal since in the state reached after this run $\mathcal{C}_1$ may offer $b!$ whereas $\mathcal{C}_2$ may (in fact will) not accept it. However, $c?.a!$ is in the product of the two implementations.*

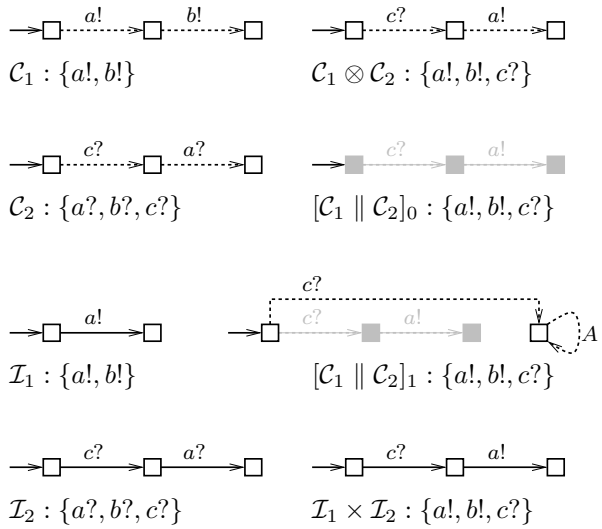


Figure 2.1: Counterexample regarding compatibility. Grey-shaded states are to be removed.

The correction

Call *exception* any word in $\mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ from which the environment has no strategy to prevent the occurrence of an illegal word, meaning that an illegal word can be obtained from the exception by following only output actions.

Definition 2.4.5 (compatibility) *The* exception language *of modal interfaces $\mathcal{C}_1$ and $\mathcal{C}_2$ is the language $\mathbb{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2} = pre^*_!(Illegal(\mathcal{C}_1, \mathcal{C}_2))$. Modal interfaces $\mathcal{C}_1$ and $\mathcal{C}_2$ are said to be* compatible *if and only if the empty word ϵ is not in $\mathbb{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$.*

Definition 2.4.6 (parallel composition) *Given two modal interfaces $\mathcal{C}_1$ and $\mathcal{C}_2$, the* relaxation *of $\mathcal{C}_1 \otimes \mathcal{C}_2$ is obtained by applying the following pseudo-algorithm to $\mathcal{C}_1 \otimes \mathcal{C}_2$:*

for all v ***in*** $\mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ ***do***
  for all a ***in*** A ***do***
    if $v \notin \mathbb{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$ ***and*** $v.a \in \mathbb{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$ ***then***
      for all w *in* A^* ***do***
        $must(v.a.w) := \emptyset$
        $may(v.a.w) := A$
      end for
    end if
  end for
end for

If $\mathcal{C}_1$ and $\mathcal{C}_2$ are compatible, the relaxation of $\mathcal{C}_1 \otimes \mathcal{C}_2$ is called the parallel composition *of $\mathcal{C}_1$ and*

$\mathcal{C}_2$, denoted by $\mathcal{C}_1 \parallel \mathcal{C}_2$. Whenever $\mathcal{C}_1$ and $\mathcal{C}_2$ are incompatible, the parallel composition $\mathcal{C}_1 \parallel \mathcal{C}_2$ is defined as the inconsistent modal specification $\perp$.

If the environment performs an $a?$ to which the "**if** ... **then** ..." statement applies, then illegal words may exist for certain pairs $(\mathcal{I}_1, \mathcal{I}_2)$ of strong implementations of $\mathcal{C}_1$ and $\mathcal{C}_2$. If this occurs, then $\mathcal{C}_1 \parallel \mathcal{C}_2$ relaxes all constraints on the future of the corresponding runs — Nothing is forbidden, nothing is mandatory: the system has reached a "universal" state. This parallels the pruning rule combined with alternating simulation, in the context of interface automata.

Example 2.4.7 *We now show that our relaxation allows to correct the counter example stated in Figure 2.1. We observe that our relaxation procedure yields $[\mathcal{C}_1 \parallel \mathcal{C}_2]_1$, with $A = \{a!, b!, c?\}$, which has $\mathcal{I}_1 \times \mathcal{I}_2$ as an implementation.*

Associativity of the parallel composition operator is one of the key requirements of an interface framework, since it enables independent design of sub-systems. Unlike in [122, 138], where associativity is only mentioned, we can now state the following theorem:

Theorem 2.4.8 *The parallel composition operator is commutative and associative.*

Thanks to the interplay between modalities and profiles, knowledge about exceptions is preserved by parallel composition. This is the very reason why it is associative. Indeed, the last $a?$ action in exception runs of the from $v.a?$ comes with a may modality. In this way, it is distinguished from normal inputs which come with a must modality. When taking the parallel composition with another modal interface with a profile such that $a?$ is also an input, the resulting modality is a may. In this way, knowledge of the occurrence of an exception is preserved. Whenever this input action $a?$ is composed with an output $a!$, this results in an illegal run, meaning that an exception will be triggered earlier.

As for interface automata (Theorem 4 in [78]), strong refinement preserves compatibility, assuming that the refined modal interface does not introduce new shared actions.

Lemma 2.4.9 *Given any three modal interfaces $\mathcal{C}_i$, $i = 1...3$, such that $\mathcal{C}_2 \leq_s \mathcal{C}_1$ and $A_1 \cap A_3 \supseteq A_2 \cap A_3$:*

- *$\mathbf{pr}_{A_1 \cup A_3}(Illegal(\mathcal{C}_2, \mathcal{C}_3))$ is included in $Illegal(\mathcal{C}_1, \mathcal{C}_3)$;*
- *$\mathbf{pr}_{A_1 \cup A_3}\left(\mathbb{E}_{\mathcal{C}_2 \parallel \mathcal{C}_3}\right)$ is included in $\mathbb{E}_{\mathcal{C}_1 \parallel \mathcal{C}_3}$.*

Proof: *Consider an illegal word $u \in Illegal(\mathcal{C}_2, \mathcal{C}_3)$ for $\mathcal{C}_2 \otimes \mathcal{C}_3$. This means that there exists an action $a \in A_2 \cap A_3$ such that (i) either a is an output of $\mathcal{C}_2$ and an input of $\mathcal{C}_3$, such that $a \in may_2(\mathbf{pr}_{A_2}(u))$ and $a \notin must_3(\mathbf{pr}_{A_3}(u))$, or (ii) a is an input of $\mathcal{C}_2$ and an output of $\mathcal{C}_3$, such that $a \notin must_2(\mathbf{pr}_{A_2}(u))$ and $a \in may_3(\mathbf{pr}_{A_3}(u))$.*

By Definition 2.2.7, u is also in $\mathcal{L}_{A2 \cup A3 \mathcal{C}_1 \otimes \mathcal{C}_3}$. By Definition 2.2.3, $u' = \mathbf{pr}_{A_1 \cup A_3}(u)$ belongs to $\mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_3}$.

Since it is assumed that $A_2 \cap A_3 \subseteq A_1 \cap A_3$, *action* a *belongs to* $A_1 \cap A_3$. *By Definition 2.2.6, either* a *is an output of* $\mathcal{C}_1$ *and an input of* $\mathcal{C}_3$, *such that* $a \in may_1(\mathbf{pr}_{A_1}(u'))$ *and* $a \notin must_3(\mathbf{pr}_{A_3}(u'))$, *or (ii)* a *is an input of* $\mathcal{C}_1$ *and an output of* $\mathcal{C}_3$, *such that* $a \notin must_1(\mathbf{pr}_{A_1}(u'))$ *and* $a \in may_3(\mathbf{pr}_{A_3}(u'))$. *Meaning that* $u' \in Illegal(\mathcal{C}_1, \mathcal{C}_3)$, *which proves the first part of the lemma.*

Next, recall that $A_1! \cup A_3!$ *is included in* $A_2! \cup A_3!$. *Hence, the projection of the backward closure* $\mathbf{pr}_{A_1 \cup A_3}(pre_!^*(Illegal(\mathcal{C}_2, \mathcal{C}_3)))$ *is included in the backward closure of the projection* $pre_!^*\big(\mathbf{pr}_{A_1 \cup A_3}(Illegal(\mathcal{C}_2, \mathcal{C}_3))\big)$, *which is in turn included in* $pre_!^*(Illegal(\mathcal{C}_1, \mathcal{C}_3))$, *thanks to the previous part of the Lemma.* ∎

Corollary 2.4.10 (compatibility preservation) *Given any three modal interfaces* $\mathcal{C}_i$, $i = 1...3$, *such that* $\mathcal{C}_2 \leq_s \mathcal{C}_1$ *and* $A_1 \cap A_3 \supseteq A_2 \cap A_3$. $\mathcal{C}_1$ *compatible with* $\mathcal{C}_3$ *implies that* $\mathcal{C}_2$ *and* $\mathcal{C}_3$ *are also compatible.*

Proof: *This is an immediate consequence of Lemma 2.4.9. Assume* $\mathcal{C}_2$ *and* $\mathcal{C}_3$ *incompatible, meaning that* $\epsilon \in \mathbb{E}_{\mathcal{C}_2 \parallel \mathcal{C}_3}$. *By Lemma 2.4.9,* $\epsilon = \mathbf{pr}_{A_1 \cup A_3}(\epsilon) \in \mathbb{E}_{\mathcal{C}_1 \parallel \mathcal{C}_3}$. *Hence* $\mathcal{C}_1$ *and* $\mathcal{C}_3$ *are also incompatible.* ∎

Contrary to interface automata for which $\mathcal{C}_1 \parallel \mathcal{C}_2$ is a refinement of $\mathcal{C}_1 \otimes \mathcal{C}_2$ [78], relaxation of modal interfaces amounts to compute an abstraction of the product:

Lemma 2.4.11 *Given two modal interfaces* $\mathcal{C}_1$ *and* $\mathcal{C}_2$*:*

$$\mathcal{C}_1 \otimes \mathcal{C}_2 \leq \mathcal{C}_1 \parallel \mathcal{C}_2$$

Proof: *Two cases are possible:*

- *if* $u \in \mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_2} \backslash \mathbb{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$: $must_{\mathcal{C}_1 \otimes \mathcal{C}_2}(u) = must_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u)$ *and* $may_{\mathcal{C}_1 \otimes \mathcal{C}_2}(u) = may_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u)$;
- *if* $u \in \mathbb{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$ *then* $u \in \mathcal{L}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$ *and* $must_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u) = \emptyset$ *and* $may_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u) = A$.

Thus, $must_{\mathcal{C}_1 \otimes \mathcal{C}_2}(u) \supseteq must_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u)$ *and* $may_{\mathcal{C}_1 \otimes \mathcal{C}_2}(u) \subseteq may_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u)$. ∎

Theorem 10 stated in [122, 138] now holds for the parallel composition operator.

Theorem 2.4.12 (independent implementability) *For any two modal interfaces* $\mathcal{C}_1$, $\mathcal{C}_2$ *and two implementations* $(\mathcal{I}_1, \pi_1)$, $(\mathcal{I}_2, \pi_2)$ *such that* $(\mathcal{I}_1, \pi_1) \models_s \mathcal{C}_1$ *and* $(\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_2$, *it holds that* $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_1 \parallel \mathcal{C}_2$.

Proof: *If* $(\mathcal{I}_1, \pi_1) \models_s \mathcal{C}_1$ *and* $(\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_2$, *then, by Theorem 2.4.3,* $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_1 \otimes \mathcal{C}_2$. *By the previous lemma and by the generalization of Theorem 1 in Theorem 2.4.3:* $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_1 \parallel \mathcal{C}_2$. ∎

2.5 Conclusion and future work

This chapter presents a *modal interface* framework, a unification of interface automata and modal specifications. It is a complete theory with a powerful composition algebra that includes operations such as conjunction (for requirements composition) and residuation (for component reuse but also assume/guarantee contract based reasoning [150]). However, the core contribution of the chapter is a parallel composition operator that reflects a rich notion of compatibility between components, actually correcting that parallel composition proposed in [122, 138].

There are several possible directions for future research. A first step would be to implement all the concepts and operations presented in the chapter and evaluate the resulting tool on concrete case studies. Extensions of modal specifications can be investigated, where states are described as valuations of a set of variables just as it has been the case for interface automata [63, 76].

Another promising direction would be a timed extension of modal interfaces. In [79], de Alfaro et al. proposed *timed interface automata* that extends timed automata just as interface automata extend finite-word automata. The semantics of a timed interface automaton is given by a timed game [77, 44], which allows to capture the *timed dimension* in composition. Up to now, composition is the only operation that has been defined on timed interface automata. In [64], Chatain et al. have proposed a notion of refinement for timed games. However monotony of parallel composition with respect to this refinement relation has not been investigated yet. In [38], *timed modal specifications* are proposed. As modal specifications, timed modal specifications admit a rich composition algebra with product, conjunction and residuation operators. Thus, a natural direction for future research would be to unify timed interface automata and timed modal specifications. This would imply a translation from timed interface automata to timed modal specifications.

Finally, we believe it is worth studying the logical expressiveness of timed modal specifications/interfaces, as it has been the case for modal specifications [92].

Chapter 3

Constraint Markov Chains

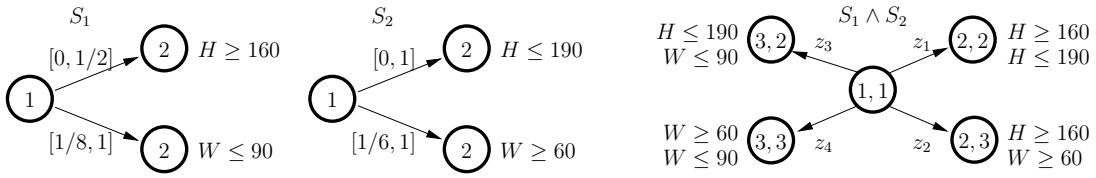
Résumé : *En suivant les mêmes lignes directrices que le chapitre 2, le présent chapitre a pour objectif de proposer une théorie d'interfaces stochastiques, les chaînes de Markov à contraintes (CMC) . Une CMC est essentiellement une chaîne de Markov à temps discret dont les probabilités de transition ne sont ni fixées ni données par extension, mais au contraire, sont solutions d'un ensemble de contraintes exprimées par des formules du premier ordre interprétées sur les réels. Une CMC définit donc un ensemble éventuellement infini de réalisations, c'est à dire de chaînes de Markov. Ce chapitre s'attache à montrer que les CMC forment une théorie d'interface, à l'exception de l'opération de residuation dont l'existence est'a ce jour une question ouverte.*

3.1 Introduction

In this chapter we introduce *Constraint Markov Chains* (CMCs) as a foundational specification formalism for component-based development of probabilistic systems. In particular, we provide constructs on CMCs supporting refinement, consistency checking, logical as well as structural composition of specifications – all indispensable ingredients for a compositional design methodology.

Over the years several process algebraic frameworks have been proposed for describing and analysing probabilistic systems based on Markov Chains and Markov Decision Processes, e.g. [103, 3, 127]. Also, a variety of probabilistic logics have been proposed for expressing properties of such systems, e.g. PCTL [99]. Both approaches support refinement between specifications using various notions of probabilistic (bi)simulation (e.g., [91, 113]) and logical entailment (e.g. [102]). Whereas the process algebraic approach favors structural composition (e.g. parallel composition), the logical approach favors logical combinations (e.g. logical conjunction). Neither of the two approaches supports both structural and logical composition.

For functional analysis the notion of Modal Transition Systems (MTS) [119] provides a use-



(a) IMCs S_1 and S_2 (b) CMC $S_1 \wedge S_2$

Figure 3.1: IMCs: non-closure under conjunction

ful specification formalism supporting refinement as well as logical and structural composition and with recent applications to Interface Theories [122, 150]. Generalizing the notion of Modal Transition Systems to the non-functional analysis of probabilistic systems, the formalism of Interval Markov Chains (IMCs) was introduced in [113] with notions of satisfaction and refinement generalizing probabilistic bisimulation. Informally, an IMC extends the notion of Markov Chains by having transitions labelled by *intervals* (open or closed) of allowed probabilities rather than individual probabilities.

In more recent work, IMCs have been subject to further study: a weaker (yet sound) refinement for IMCs is introduced [91], and model checking procedures for PCTL for such systems are considered [164, 91, 65]. In a very recent work [112] a composition operation has been studied for IMCs augmented with may and must transitions very much in the spirit of [119].

However, the expressive power of IMCs is inadequate to support both logical and structural composition. To see this, consider two IMCs, S_1 and S_2, in Figure 3.1 specifying different probability constraints related to the height (H) and weight (W) of a given random person. Attempting to express the conjunction $S_1 \wedge S_2$ as an IMC by simple intersection of bounds gives $z_1 \leq 1/2$, $1/6 \leq z_2 \leq 1/2$, $1/8 \leq z_3$ and $1/6 \leq z_4$. However, this naive construction is too coarse and does not adequately capture conjunction: whereas $(z_1, z_2, z_3, z_4) = (1/2, 1/6, 1/8, 5/24)$ is a solution to the above constraints the resulting overall probability of reaching a state satisfying H≥ 160, i.e. $z_1 + z_2 = 2/3$, clearly violates the upper bound $1/2$ specified in S_1. What is needed is the ability to express dependencies between the probabilities z_1, z_2, z_3, z_4 besides that of being a probability distribution, i.e. $z_1 + z_2 + z_3 + z_4 = 1$. Obviously, the correct conjunctive combination is expressed by the three constraints $z_1 + z_2 \leq 1/2, 1/8 \leq z_3 + z_4, 1/6 \leq z_2 + z_4$, exceeding the expressive power of IMCs. Similarly, simple examples demonstrate that IMCs are also not closed under parallel composition.

Constraint Markov Chains (CMCs) are a further extension of Markov Chains allowing arbitrary constraints on the next-state probabilities from any state. Whereas linear constraints suffice for closure under conjunction, polynomial constraints are, as we shall see, necessary for closure under parallel composition. We define notions of satisfaction and (weak) refinement for CMCs conservatively extending similar notions for IMCs. In particular, as a main theorem, we prove that for deterministic CMCs the notion of weak refinement is complete with respect to the inclusion of implementation-sets. In addition, we provide a construction, which for any CMC S returns a deter-

ministic CMC $\rho(S)$ containing S with respect to weak refinement. Finally, we show that refinement between CMCs with polynomial constraints can be decided in essentially single exponential time.

3.2 Constraint Markov Chains

Let A, B be sets of propositions with $A \subseteq B$. The *restriction of* $T \subseteq B$ *to* A is given by $T\mathbf{proj}_{,A}(\equiv) T \cap A$. If $T \subseteq 2^B$, then $T\mathbf{proj}_{,A}(\equiv) \{W\mathbf{proj}_{,A}(|) W \in T\}$. For $V \subseteq A$ define the *extension of* V *to* B as $T{\uparrow}^B{\equiv} \{W \subseteq B \mid W\mathbf{proj}_{,A}(=) T\}$, so the set of sets whose restriction to A is T. Lift it to sets of sets as follows: if $T \subseteq 2^A$ then $T{\uparrow}^B{\equiv} \{W \subseteq B \mid W\mathbf{proj}_{,A}(\in) T\}$. Let $M \in [0,1]^{n\times k}$ be a matrix and $x \in [0,1]^{1\times k}$ be a vector. We write M_{ij} for the cell in ith row and jth column of M, M_p for the pth row of M, and x_i for the ith element of x. Finally, M is a *correspondence matrix* iff $0 \leq \sum_{j=1}^{k} \Delta_{ij} \leq 1$ for all $1 \leq i \leq n$.

Definition 3.2.1 *A* Markov Chain (MC in short) *is a tuple* $\langle\{1,\ldots,n\}, o, M, A, V\rangle$*, where* $\{1,\ldots,n\}$ *is a set of states containing the initial state* o*,* A *is a set of atomic propositions,* $V : \{1,\ldots,n\} \to 2^A$ *is a state valuation, and* $M \in [0,1]^{n\times n}$ *is a probability transition matrix:* $\sum_{j=1}^{n} M_{ij} = 1$ *for* $1 \leq i \leq n$.

We now introduce *Constraint Markov Chains* (CMCs in short), a finite representation for a possibly infinite set of MCs. Roughly speaking, CMCs generalize MCs in that, instead of specifying a concrete transition matrix, they only constrain probability values in the matrix. Constraints are modeled using a *characteristic function*, which for a given source state and a distribution of probabilities of leaving the state evaluates to 1 iff the distribution is permitted by the specification. Similarly, instead of a concrete valuation function for each state, a *constraint on valuations* is used. Here, a valuation is permitted iff it is contained in the set of admissible valuations of the specification.

Definition 3.2.2 *A* Constraint Markov Chain *is a tuple* $S = \langle\{1,\ldots,k\}, o, \phi, A, V\rangle$*, where* $\{1,\ldots,k\}$ *is a set of states containing the initial state* o*,* A *is a set of atomic propositions,* $V : \{1,\ldots,k\} \to 2^{2^A}$ *is a set of admissible state valuations. and* $\phi : \{1,\ldots,k\} \to [0,1]^k \to \{0,1\}$ *is a* constraint function *such that if* $\phi(j)(x) = 1$ *then the* x *vector is a probability distribution:* $0 \leq x_i \leq 1$ *and* $\sum_{i=1}^{k} x_i = 1$.

An *Interval Markov Chain* (IMC in short) [113] is a CMC whose constraint functions are represented by intervals, so for all $1 \leq i \leq k$ there exist constants α_i, β_i such that $\phi(j)(x) = 1$ iff $x_i \in [\alpha_i, \beta_i]$.

Example 3.2.3 *Two parties, a customer and a vendor, are discussing a design of a relay for an optical telecommunication network. The relay is designed to amplify an optic signal transmitted over a long distance over an optic fiber. The relay should have several modes of operation, modeled*

by four dynamically changing properties and specified by atomic propositions a, b, c, and e (see Figure 3.2a).

The customer presents CMC S_1 (Figure 3.2b) specifying the admissible behavior of the relay from their point of view. States are labeled with formulas characterizing sets of valuations. For instance, "$(a+b+c \geq 2) \wedge (e=0)$" at state 2 of S_1 represents $V_1(2) = \{\{a,b\},\{b,c\},\{a,c\},\{a,b,c\}\}$, where a, b, c, and e range over Booleans. State 1 specifies a standby mode, where no signal is emitted and only marginal power is consumed. State 2 is the high power mode, offering a high signal/noise ratio, and hence a high bitrate and low error rate, at the expense of a high power consumption. State 3 is the low power mode, with a low power consumption, low bitrate and high error rate. The customer prescribes that the probability of the high power mode (state 2) is higher than 0.7.

The vendor replies with CMC S_2 (Figure 3.2c), which represents possible relays that they can build. Because of thermal limitations, the low power mode has a probability higher than 0.2.

A state u of S is *reachable* from a state i if there exists a probability distribution, or a vector $x \in [0,1]^k$, with a nonzero probability x_u, which satisfies $\phi(i)(x)$. A CMC S is *deterministic* iff for every state i, states reachable from i have pairwise disjoint admissible valuations:

Definition 3.2.4 *Let $S = \langle \{1,\ldots,k\}, o, \phi, A, V\rangle$ be a CMC. S is* deterministic *iff for all states $i,u,v \in \{1,\ldots,k\}$, if there exists $x \in [0,1]^k$ such that $(\phi(i)(x) \wedge (x_u \neq 0))$ and $y \in [0,1]^k$ such that $(\phi(i)(y)(\wedge y_v \neq 0))$, then we have that $V(u) \cap V(v) = \emptyset$.*

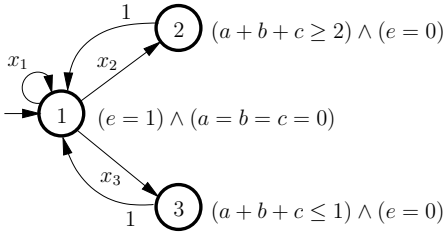
In our example both S_1 and S_2 are deterministic specifications. In particular states 2 and 3, reachable from 1 in both CMCs, have disjoint constraints on valuations (see Figure 3.2).

We relate CMC specifications to MCs implementing them, by extending the definition of satisfaction presented in [113] to observe the valuations constraints and the full-fledged constraint functions. Crucially, like [113], we abstract from syntactic structure of transitions—a single transition in the implementation MC can contribute to satisfaction of more than one transition in the specification, by distributing its probability mass against several transitions. Similarly many MC transitions can contribute to satisfaction of just one specification transition. This redistribution of probability mass is described by correspondence matrices. Consider the following example:

Example 3.2.5 *We illustrate the concept of correspondence matrix between Specification S_1 (given in Figure. 3.2b) and Implementation P_2 (given in Figure 3.2e). The CMC S_1 has three outgoing transitions from state 1 but, due to constraint function in 1, the transition labeled with x_1 cannot be taken (the constraint implies $x_1 = 0$). The probability mass going from state 1 to states 2 and 3 in P_2 corresponds to the probability allowed by S_1 from its state 1 to its state 2; The redistribution is done with the help of the matrix Δ given in Figure 3.2h. The ith column in Δ describes how big fraction of each transition probability (for transitions leaving 1) is associated with probability x_i in S_2. Observe that the constraint function $\phi_1(1)(0,0.8,0.2) = \phi_1(1)((0,0.7,0.1,0.2) \times \Delta)$ is satisfied.*

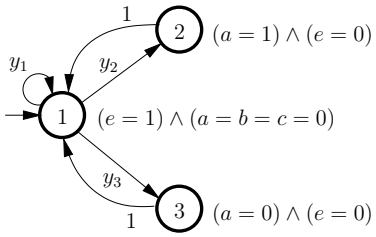
a	ber $\leq 10^{-9}$	The bit error rate is less than 1 per billion bits transmitted.
b	br > 10Gbits/s	The bit rate is higher than 10 Gbits/s.
c	$P < 10$W	Power consumption is less than 10 W.
e	Standby	The relay is not transmitting.

(a) Atomic propositions in the optic relay specifications



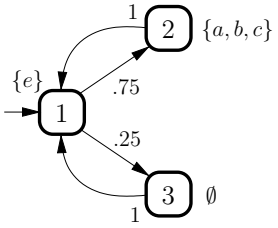
$\phi_1(1)(x) \equiv (x_1 = 0) \wedge (x_2 \geq 0.7) \wedge (x_2 + x_3 = 1)$

(b) CMC S_1, the customer specification of the optic relay

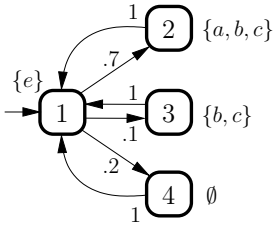


$\phi_2(1)(y) \equiv (y_1 = 0) \wedge (y_3 \geq 0.2) \wedge (y_2 + y_3 = 1)$

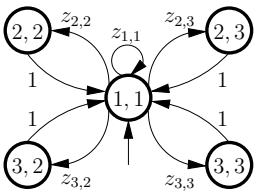
(c) The manufacturer specification, S_2, of the optic relay



(d) Markov Chain P_1 satisfying S_1 and S_2

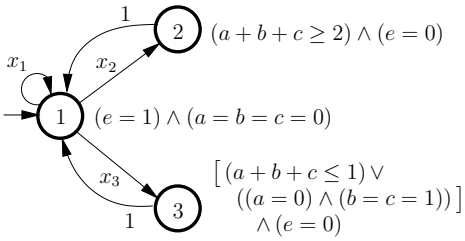


(e) Another Markov Chain P_2 satisfying S_1 and S_2



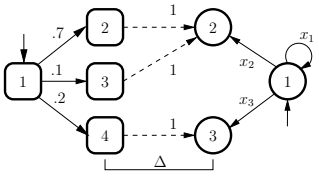
$\phi_3(1,1)(Z) \equiv \big[(\forall j,\ z_{1,j} = 0) \wedge (z_{2,2} + z_{2,3} \geq 0.7)$
$\wedge\, (z_{2,2} + z_{2,3} + z_{3,2} + z_{3,3} = 1)\big]$
$\wedge\, \big[(\forall i,\ z_{i,1} = 0) \wedge (z_{2,3} + z_{3,3} \geq 0.2)\big]$

(f) Conjunction S_3 of S_1 and S_2. Constraints on propositions, pairwise conjunctions (intersections) of constraints of S_1 and S_2, are left out to avoid clutter



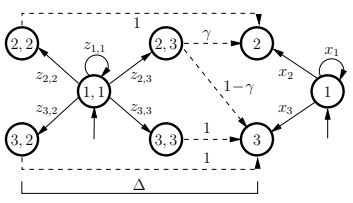
$\phi_4(1)(x) \equiv (x_1 = 0) \wedge (x_2 \geq 0.7)$
$\wedge\, (x_3 \geq 0.2) \wedge (x_2 + x_3 = 1)$

(g) CMC S_4 generalizing S_3, so $S_3 \preceq S_4$



$$\Delta = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(h) Correspondence for initial states of P_2 and S_1



$$\Delta = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \gamma & 1-\gamma \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

(i) Weak refinement for initial states of S_3 and S_4

Figure 3.2: Examples

Definition 3.2.6 *Let* $P = \langle \{1,\dots,n\}, o_P, M, A_P, V_P \rangle$ *be a MC and* $S = \langle \{1,\dots,k\}, o_S, \phi, A_S, V_S \rangle$ *be a CMC with* $A_s \subseteq A_p$*. Then* $\mathcal{R} \subseteq \{1,\dots,n\} \times \{1,\dots,k\}$ *is a* satisfaction relation *between states of* P *and* S *iff whenever* $p\,\mathcal{R}\,u$ *then (1)* $V_P(p)\mathbf{proj}_{,A_S}(\in) V_S(u)$*, and (2) there exists a correspondence matrix* $\Delta \in [0,1]^{n\times k}$ *such that (a) for all* $1 \le p' \le n$ *with* $M_{pp'} \neq 0$*,* $\sum_{j=1}^{k} \Delta_{p'j} = 1$*; (b)* $\phi(u)(M_p \times \Delta)$ *holds and (c) if* $\Delta_{p'u'} \neq 0$ *then* $p'\,\mathcal{R}\,u'$*.*

We write $P \models S$ iff there exists a satisfaction relation relating o_P and o_S, and call P an *implementation* of S. The set of all implementations of S is given by $[\![S]\!] \equiv \{P \mid P \models S\}$. Rows of Δ that correspond to reachable states of P always sum up to 1. This is to guarantee that the entire probability mass of implementation transitions is allocated. For unreachable states, we leave the corresponding rows in Δ unconstrained. P may have a richer alphabet than S, in order to facilitate abstract modeling: this way an implementation can maintain local information using an internal variable.

Remark 3.2.7 *Our semantics for CMCs follows the Markov Decision process (MDP in short) semantics tradition [164, 65]. In the literature, the MDP semantic is opposed to the Uncertain Markov Chain (UMC in short) semantics where the probability distribution from each state is fixed a priori.*

3.3 Consistency, Refinement and Conjunction

We now study the notions of consistency, refinement, and conjunction for Constraint Markov Chains.

3.3.1 Consistency

A CMC S is *consistent* if it admits at least one implementation. We now discuss how to decide consistency. A state u of S is *valuation consistent* iff $V(u) \neq \emptyset$; it is *constraint consistent* iff there exists a probability distribution vector $x \in [0,1]^{1\times k}$ such that $\phi(u)(x) = 1$. It is easy to see that if *each state* of S is both valuation and constraint consistent then S is also consistent. However, inconsistency of a state does not imply inconsistency of the specification. The operations presented later in this chapter may introduce inconsistent states, leaving a question if a resulting CMC is consistent. In order to decide whether S is inconsistent, local inconsistencies are propagated throughout the entire state-space using a *pruning operator* β that removes inconsistent states from S. The result $\beta(S)$ is a new CMC, which may still contain some inconsistent states. The operator is applied iteratively, until a fixpoint is reached. If the resulting CMC $\beta^*(S)$ contains at least one state then S is consistent. Also S has the same models as $\beta^*(S)$.

We define β formally. Let $S = \langle \{1,\dots,k\}, o, \phi, A, V \rangle$. If o is locally inconsistent then let $\beta(S) = \emptyset$. If S does not contain inconsistent states then $\beta(S) = S$. Else proceed in two steps. First for $k' < k$ define a function $\nu : \{1,\dots,k\} \to \{\bot, 1,\dots,k'\}$, which will remove inconsistent

states. All locally inconsistent states are mapped to $\perp$. For all $1 \leq i \leq k$ take $\nu(i) = \perp$ iff $[(V(i) = \emptyset) \vee (\forall x \in [0,1]^k,\ \phi(i)(x) = 0)]$. All remaining states are mapped injectively into $\{1, \ldots, k'\}$: $\nu(i) \neq \perp \Rightarrow \forall j \neq i,\ \nu(j) \neq \nu(i)$. Then let $\beta(S) = \langle \{1, \ldots, k'\}, \nu(o), \phi', A, V' \}$, where $V'(i) = V(\nu^{-1}(i))$ and for all $1 \leq j \leq k'$ the constraint $\phi'(j)(y_1, \ldots, y_{k'})$ is: $\exists x_1, \ldots, x_k$ s.t.

$$\Big[\nu(q) = \perp \Rightarrow x_q = 0\Big] \text{ and } \Big[\forall 1 \leq l \leq k',\ y_l = x_{\nu^{-1}(l)}\Big] \text{ and } \Big[\phi(\nu^{-1}(j))(x_1, \ldots, x_k)\Big] \ .$$

The constraint makes the locally inconsistent states unreachable, and then $\perp$ is dropped as a state.

Theorem 3.3.1 *Let $S = \langle \{1, \ldots, k\}, o, \phi, A, V \rangle\}$ be a CMC and $\beta^*(S) = \lim_{n \to \infty} \beta^n(S)$ be the fixpoint of β. For any MC P, we have (1) $P \models S \iff P \models \beta(S)$ and (2) $[\![S]\!] = [\![\beta^*(S)]\!]$.*

3.3.2 Refinement

Refinement is a concept that allows to "compare" two specifications. Roughly speaking, if S_1 refines S_2, then any model of S_1 should also be a model of S_2. In [113], Jonsson and Larsen have proposed a notion of strong refinement between IMCs. This definition extends to CMCs in the following way.

Definition 3.3.2 *Let $S_1 = \langle \{1, \ldots, k_1\}, o_1, \phi_1, A_1, V_1 \rangle$ and $S_2 = \langle \{1, \ldots, k_2\}, o_2, \phi_2, A_2, V_2 \rangle$ be CMCs with $A_2 \subseteq A_1$. The relation $\mathcal{R} \subseteq \{1, \ldots, k_1\} \times \{1, \ldots, k_2\}$ is a* strong refinement relation *between states of S_1 and S_2 iff whenever $v \,\mathcal{R}\, u$ then (1) $V_1(v)\mathbf{proj}_{,A_2}(\subseteq) V_2(u)$ and (2) there exists a correspondence matrix $\Delta \in [0,1]^{k_1 \times k_2}$ such that for all probability distribution vectors $x \in [0,1]^{1 \times k_1}$ if $\phi_1(v)(x)$ holds then (a) $x_i \neq 0 \Rightarrow \sum_{j=1}^{k_2} \Delta_{ij} = 1$; (b) $\phi_2(u)(x \times \Delta)$ holds and (c) if $\Delta_{v'u'} \neq 0$ then $v' \,\mathcal{R}\, u'$. We say that S_1 strongly refines S_2 iff $o_1 \,\mathcal{R}\, o_2$.*

It is easy to see that strong refinement implies implementation set inclusion. However, the converse is not true. The strong refinement imposes a "fixed-in-advance" witness matrix regardless of the probability distribution satisfying the constraint function. We propose a *weak refinement* that is complete for deterministic CMCs. Our definition generalizes the one proposed in [91] for IMCs.

Definition 3.3.3 *Let $S_1 = \langle \{1, \ldots, k_1\}, o_1, \phi_1, A_1, V_1 \rangle$ and $S_2 = \langle \{1, \ldots, k_2\}, o_2, \phi_2, A_2, V_2 \rangle$ be two CMCs, with $A_2 \subseteq A_1$. Then $\mathcal{R} \subseteq \{1, \ldots, k_1\} \times \{1, \ldots, k_2\}$ is a* weak refinement relation *iff whenever $v \,\mathcal{R}\, u$ then (1) $V_1(v)\mathbf{proj}_{,A_2}(\subseteq) V_2(u)$ and (2) for any probability distribution vector $x \in [0,1]^{1 \times k_1}$ such that $\phi_1(v)(x)$, there exists a matrix $\Delta \in [0,1]^{k_1 \times k_2}$ such that (a) for all S_1 states $1 \leq i \leq k_1$, $x_i \neq 0 \Rightarrow \sum_{j=1}^{k_2} \Delta_{ij} = 1$; (b) $\phi_2(u)(x \times \Delta)$ and (c) $\Delta_{v'u'} \neq 0 \Rightarrow v' \,\mathcal{R}\, u'$. We say that CMC S_1 (weakly) refines S_2, written $S_1 \preceq S_2$, iff $o_1 \,\mathcal{R}\, o_2$.*

It is easy to see that the weak refinement implies implementation set inclusion. Showing the converse is more involved. We postpone it to Section 3.5.

Example 3.3.4 *Figure 3.2i illustrates a family of correspondence matrices parameterized by γ witnessing the weak refinement between initial states of S_3 and S_4 (defined in Figure 3.2). The actual matrix used in proving the weak refinement depends on the probability distribution vector z that satisfies the constraint function ϕ_3 of state $(1,1)$. Take $\gamma = \frac{0.7-z_{22}}{z_{23}}$ if $z_{22} \leq 0.7$ and $\gamma = \frac{0.8-z_{22}}{z_{23}}$ otherwise. It is easy to see that if $\phi_3((1,1))(z)$ holds, then $\phi_4(1)(z \times \Delta)$ holds.*

3.3.3 Conjunction

Conjunction is a useful operation combining requirements of several specifications.

Definition 3.3.5 *Let $S_1 = \langle\{1,\ldots,k_1\}, o_1, \phi_1, A_1, V_1\rangle$ and $S_2 = \langle\{1,\ldots,k_2\}, o_2, \phi_2, A_2, V_2\rangle$ be two CMCs. The conjunction of S_1 and S_2, written $S_1 \wedge S_2$, is the CMC $S = \langle\{1,\ldots,k_1\} \times \{1,\ldots,k_2\}, (o_1,o_2), \phi, A, V\rangle$ with $A = A_1 \cup A_2$, $V((u,v)) = V_1(u)\uparrow^A \cap V_2(v)\uparrow^A$, and*

$$\phi((u,v))(x_{1,1},x_{1,2},\ldots,x_{2,1},\ldots,x_{k_1,k_2}) \equiv \phi_1(u)(\sum_{j=1}^{k_2} x_{1,j},\ldots,\sum_{j=1}^{k_2} x_{k_1,j}) \wedge \phi_2(v)(\sum_{i=1}^{k_1} x_{i,1},\ldots,\sum_{i=1}^{k_1} x_{i,k_2}).$$

Conjunction is an operation that conserves determinism and may introduce inconsistent states (see Example 3.3 below) and thus a use of conjunction should normally be followed by applying the pruning operator β. As we already said in the introduction, the result of conjoining two IMCs is not an IMC in general, but a CMC whose constraint functions are linear.

Example 3.3.6 *Figure 3.2f depicts a CMC S_3 expressing the conjunction of IMCs S_1 and S_2 (see Figures 3.2b–3.2c). The constraint $z_{2,3}+z_{3,3} \geq 0.2$ in state $(1,1)$ cannot be expressed as an interval.*

Finally, the following theorem shows the conjunction of two specifications coincides with their greatest lower bound with respect to the weak refinement (also called *shared refinement*).

Theorem 3.3.7 *Let S_1, S_2 and S_3 be three CMCs. We have $((S_1 \wedge S_2) \preceq S_1) \wedge ((S_1 \wedge S_2) \preceq S_2)$ and $(S_3 \preceq S_1) \wedge (S_3 \preceq S_2) \Rightarrow S_3 \preceq (S_1 \wedge S_2)$.*

3.4 Compositional Reasoning

Let us now turn to studying composition of CMCs. We start by discussing how systems and specifications can be composed in a non-synchronizing way, then we introduce a notion of synchronization. The non-synchronizing *independent* composition is largely just a product of two MCs (or CMCs). We begin with composition of MCs.

Definition 3.4.1 *Let $S_1 = \langle\{1,\ldots,n_1\}, o_1, M', A_1, V_1\rangle$ and $S_2 = \langle\{1,\ldots,n_2\}, o_2, M'', A_2, V_2\rangle$ be two MCs and suppose $A_1 \cap A_2 = \emptyset$. The parallel composition of P_1 and P_2 is the MC $P_1 \parallel P_2 = \langle\{1,\ldots,n_1\} \times \{1,\ldots,n_2\}, (o_1,o_2), M, A_1 \cup A_2, V\rangle$ where: $M \in [0,1]^{(n_1\times n_2)\times(n_1\times n_2)}$ is such that $M_{(p,q)(r,s)} = M'_{pr} \cdot M''_{qs}$; and $V((p,q)) = V_1(p) \cup V_2(q)$.*

We now define independent parallel composition between CMCs.

Definition 3.4.2 *Let* $S_1 = \langle \{1, \ldots, k_1\}, o_1, \phi_1, A_1, V_1 \rangle$ *and* $S_2 = \langle \{1, \ldots, k_2\}, o_2, \phi_2, A_2, V_2 \rangle$ *be CMCs with* $A_1 \cap A_2 = \emptyset$*. The parallel composition of* S_1 *and* S_2 *is the CMC* $S_1 \parallel S_2 = \langle \{1, \ldots, k_1\} \times \{1, \ldots, k_2\}, (o_1, o_2), \phi, A_1 \cup A_2, V \rangle$*, where* $\phi(u, v)(z_{1,1}, z_{1,2}, \ldots z_{2,1}, \ldots, z_{k_1,k_2}) = \exists x_1, \ldots, x_{k_1}, y_1, \ldots, y_{k_2} \in [0, 1]$ *such that* $\forall (i, j) \in \{1, \ldots, k_1\} \times \{1, \ldots, k_2\}$ *we have* $z_{i,j} = x_i \cdot y_j$ *and* $\phi_1(u)(x_1, \ldots, x_{k_1}) = \phi_2(v)(y_1, \ldots, y_{k_2}) = 1$*; Finally,* $V((u, v)) = \{Q_1 \cup Q_2 \mid Q_1 \in V_1(u), Q_2 \in V_2(v)\}$.

Composition preserves determinism. It is worth mentioning that IMCs are not closed under composition. Consider IMCs S and S' given in Figure 3.3a and their composition $S \parallel S'$ given in Figure 3.3b. Assume first that $S \parallel S'$ is an IMC. As a variable z_{ij} is the product of two variables x_i and y_j, if $S \parallel S'$ is an IMC, then one can show that the interval for z_{ij} is obtained by computing the products of the bounds of the intervals over which x_i and y_j range. Hence, we can show that $z_{11} \in [0, 1/2], z_{12} \in [0, 1/3], z_{21} \in [1/6, 1], z_{22} \in [0, 2/3]$. Let $[a, b]$ be the interval for the constraint z_{ij}, it is easy to see that there exists implementations I_1 of S_1 and I_2 of S_2 such that $I_1 \parallel I_2$ satisfies the constrain $z_{ij} = a$ (resp. $z_{ij} = b$). However, while each bound of each interval can be satisfied independently, some points in the polytope defined by the intervals and the constraint $\sum z_{ij} = 1$ cannot be reached. As an example, consider $z_{11} = 0, z_{12} = 1/3, z_{21} = 1/3, z_{22} = 1/3$. It is clearly inside the polytope, but one cannot find an implementation I of $S \parallel S'$ satisfying the constraints given by the parallel composition. Indeed, having $z_{11} = 0$ implies that $x_1 = 0$ and thus that $z_{12} = 0$.

Theorem 3.4.3 *If* S'_1*,* S'_2*,* S_1*,* S_2 *are CMCs then* $S'_1 \preceq S_1 \wedge S'_2 \preceq S_2$ *implies* $S'_1 \parallel S'_2 \preceq S_1 \parallel S_2$*, so the weak refinement is a precongruence with respect to parallel composition. Consequently, for any MCs* P_1 *and* P_2 *we have that* $P_1 \models S_1 \wedge P_2 \models S_2$ *implies* $P_1 \parallel P_2 \models S_1 \parallel S_2$.

As alphabets of composed CMCs have to be disjoint, the composition does not synchronize the components on state valuations like it is typically done for other (non-probabilistic) models. However, synchronization can be introduced by conjoining the composition with a *synchronizer*—a single-state CMC whose constraint function relates the atomic propositions of the composed CMCs.

Example 3.4.4 *The CMC* $S \parallel S'$ *of Figure 3.3b is synchronized with the synchronizer* Sync *given in Figure 3.3c.* Sync *removes from* $S \parallel S'$ *all the valuations that do not satisfy* $(a = d) \wedge (b = \neg c)$*. The resulting CMC is given in Figure 3.3d. Observe that an inconsistency appears in State* $(1, 1)$*. This is because there is no implementations of the two CMCs that can synchronize in the prescribed way. In general inconsistencies like this one can be uncovered by applying the pruning operator, which would return an empty specification. So synchronizers enable discovery of incompatibilities between component specifications in the same way as it is known for non-probabilistic specification models.*

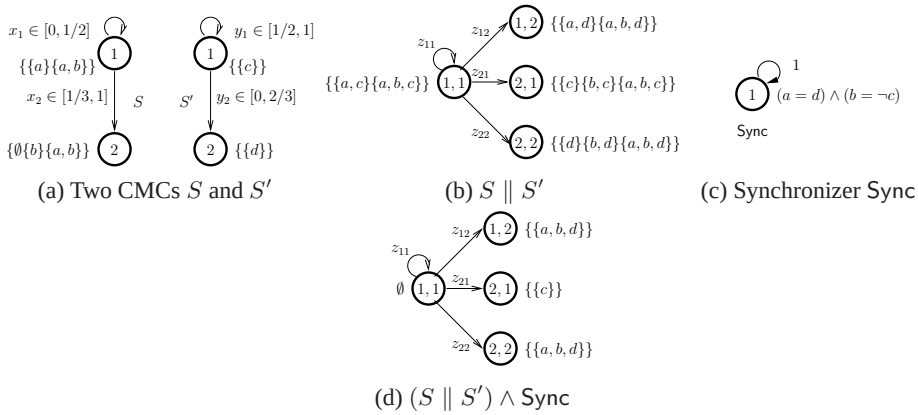


Figure 3.3: Synchronization

The following theorem states that synchronization is associative with respect to composition.

Theorem 3.4.5 *Let S_1, S_2 and S_3 be three CMCs with pairwise disjoint sets of propositions A_1, A_2 and A_3. Let Sync_{123} be a synchronizer over $A_1 \cup A_2 \cup A_3$ and let Sync_{12} be the same synchronizer with its set of propositions restricted to $A_1 \cup A_2$. The following holds $[\![[((S_1 \parallel S_2) \wedge \mathsf{Sync}_{12}) \parallel S_3] \wedge \mathsf{Sync}_{123}]\!] = [\![(S_1 \parallel S_2 \parallel S_3) \wedge S_{123}]\!]$.*

3.5 Deterministic CMCs

Clearly, if all implementations of a specification S_1 are also implementations of a specification S_2, then a designer can consider the former to be a proper strengthening of the latter. Indeed, S_1 specifies implementations that break no assumptions that can be made about implementations of S_2. Thus implementation set inclusion *is* a desirable refinement for specifications. Unfortunately, it is not directly computable. However, as we have already said, the weak refinement soundly approximates it. Had that approximation been complete, we would have an effective decision procedure for implementation set inclusion. Indeed this is the case for an important subclass of specifications: the one of deterministic CMCs. We introduce the definition of *Single Valuation Normal Form*, which plays an important role in both the determinization algorithm and in the proof of completeness.

Definition 3.5.1 *A CMC is in a* Single Valuation Normal Form *if all its admissible valuation sets are singleton ($|V(i)| = 1$ for each $1 \leq i \leq k$).*

It turns out that every consistent CMC (except those that have more than one admissible valuation in the initial state) can be transformed into the normal form preserving its implementation set.

We now present a determinization algorithm that can be applied to any CMC S whose initial state is a single valuation set. This algorithm relies on normalizing the specification first, and otherwise applies an algorithm which resembles determinization of automata. The result of the algorithm is a new CMC whose set of implementations includes the one of S. This weakening character of determinization resembles the known determinization algorithms for modal transition systems [16].

Definition 3.5.2 *Let $S = \langle \{1, \ldots, k\}, o, \phi, A, V \rangle$ be a consistent CMC in the single valuation normal form. Let $m < k$ and $h : \{1, \ldots, k\} \to \{1, \ldots, m\}$ be a surjection such that (1) $\{1, \ldots, k\} = \cup_{v \in \{1,\ldots,m\}} h^{-1}(v)$ and (2) for all $1 \leq i \neq j \leq k$, if there exists $1 \leq u \leq k$ and $x, y \in [0,1]^k$ such that $(\phi(u)(x) \wedge x_i \neq 0)$ and $(\phi(u)(y) \wedge y_j \neq 0)$, then $(h(i) = h(j) \iff V(i) = V(j))$; otherwise $h(i) \neq h(j)$. A deterministic CMC for S is the CMC $\rho(S) = \langle \{1, \ldots, m\}, o', \phi', A, V' \rangle$ where $o' = h(o)$, $\forall 1 \leq i \leq k$, $V'(h(i)) = V(i)$, and for each $1 \leq i \leq m$,*

$$\phi'(i)(y_1, \ldots, y_m) = \exists x_1, \ldots, x_k, \bigvee_{u \in h^{-1}(i)} [(\forall 1 \leq j \leq m,\ y_j = \sum_{v \in h^{-1}(j)} x_v) \wedge \phi(u)(x_1, \ldots, x_k)].$$

Theorem 3.5.3 *Let S be a CMC in* single valuation normal form, *we have $S \preceq \rho(S)$.*

As weak refinement implies inclusion, a direct consequence of Theorem 3.5.3 is that $[\![S]\!] \subseteq [\![\rho(S)]\!]$.

We now state the main theorem of the section.

Theorem 3.5.4 *Let $S_1 = \langle \{1, \ldots, k_1\}, o_1, \phi_1, A_1, V_1 \rangle$ and $S_2 = \langle \{1, \ldots, k_2\}, o_2, \phi_2, A_2, V_2 \rangle$ be two consistent single valuation normal form deterministic CMCs with $A_2 \subseteq A_1$. We have $[\![S_1]\!] \subseteq [\![S_2]\!] \Rightarrow S_1 \preceq S_2$.*

Proof: We present here a sketch of the proof. We construct the refinement relation by relating all pairs of states of S_1 and S_2 for which implementation inclusion holds. Let $\mathcal{R} \subseteq \{1, \ldots, k_1\} \times \{1, \ldots, k_2\}$ such that $v \,\mathcal{R}\, u$ iff for all MC I and state p of I, $p \models v \Rightarrow p \models u$. As we consider pruned CMCs, there exist implementations for all states. Then the usual, albeit complex and long in this case, coinductive proof technique is applied, showing that this relation is indeed a weak refinement relation.

The crucial point of the argument lies in proving the closure property — i.e. that if an S_1 state u advances possibly to u' then indeed the corresponding state v of S_2 can also advance to v' and the (u', v') pair is in $\mathcal{R}$. In other words that implementation inclusion of predecessors implies the implementation inclusion of successors. This is proven in an ad absurdum argument, roughly as follows. Assume that there would exist an implementation I' of u' which is not an implementation of v'. Then one can construct an implementation I'' of u which evolves as I'. This implementation would not implement v' but it could implement some other state of S_2. This case will be ruled out

by requiring determinism and a normal form of S_2. Then the only way for I'' to evolve is to satisfy v' which contradicts the assumption that I' is not an implementation of v'. ⌟ □

Observe that since any consistent CMC with a single valuation in initial state can be normalized, Theorem 3.5.4 holds even if S_1 and S_2 are not in single valuation normal form. We conclude that weak refinement and the implementation set inclusion coincide on the class of deterministic CMCs with at most single valuation in the initial state.

3.6 Constraints and Decidability

In the definition of CMCs, no particular type of constraints is implied, and nothing can be said, for instance on the decidability of refinement. For first order constraints over reals all our operators and relations are computable [170]. Several more tractable classes of constraints can be considered: interval, linear or polynomial constraints. Interval constraints are of the form $\phi(i)(x) = \bigwedge_j \ \alpha_{ij} \leq x_j \leq \beta_{ij}$. Linear constraints are of the form $\phi(i)(x) = x \times C_i \leq b_i$ where C_i is a matrix and b_i a row vector. Polynomial constraints are first order formulas of the form $\phi(i)(x) = \exists y, \bigwedge_j \text{sign}(P_{ij}(x,y)) = \sigma_{ij}$ with P_{ij} being polynomials of arbitrary degrees and $\sigma_{ij} \in \{-1, 0, +1\}$. These classes have increasing expressiveness, and yet what really distinguishes them is their closure properties with respect to the independent parallel and conjunction composition operators. Indeed, only the class of polynomial constraints is closed under independent parallel composition, as polynomial equations of the form $z_{ij} - x_i y_j = 0$ are introduced in the resulting constraints. Concerning the conjunction operator, only the linear and polynomial classes are closed under this composition operator, as the resulting constraints are of the form $\phi(i,j)(x) = \phi_1(i)(x \times M_1) \wedge \phi_2(j)(x \times M_2)$ which in general are not interval constraints.

We now consider the refinement checking problem between CMCs with polynomial constraints: Given S_1 and S_2, two CMCs with polynomial constraints and less than n states and s polynomials of degree d, decide whether S_1 refines S_2. It reduces to checking the validity of $O(n^2)$ instances of the following first order formula: $\forall x, \phi_1(i)(x) \Rightarrow \exists \Delta, \phi_2(j)(x \times \Delta) \wedge \bigwedge_{i'}(\sum_{j'} \Delta_{i'j'} = 1) \wedge \bigwedge_{i',j'}(i' \mathcal{R} j' \vee \Delta_{i'j'} = 0)$ where constraint $\bigwedge_{i'} \sum_{j'} \Delta_{i'j'} = 1$ relates to axiom 2.a of definition 3.3.3, under the assumption that an unreachable dummy universal state is inserted in S_2. Deciding the validity of such formulas can be done by quantifier elimination. The cylindrical algebraic decomposition algorithm [45], implemented in several symbolic computation tools (for instance, Maple [178]) performs this quantifier elimination in time double exponential in the number of variables, even when the number of quantifier alternations is constant [46]. With this algorithm, refinement can be decided in time $O(n^2 2^{2^{n^2}})$. However, considering constraints ϕ contain only existential quantifiers, quantifier alternation is exactly one in our case, and there are quantifier elimination algorithms that have a worst case complexity single exponential only in the number of variables, although they are double exponential in the number of quantifier alternations [14]. Using this algorithm, refinement can be decided in time $O(n^2 s^{n^2} d^{n^2})$.

Deciding whether a CMC is deterministic is of particular importance since refinement is not

complete in the class of non-deterministic CMCs and that determinization is an abstraction in general. Determinism of a CMC with polynomial constraints can also be decided in time single exponential in the size of the CMC. However, this problem becomes polynomial when restricting constraints to be linear inequalities. Consider a CMC S with linear constraints $\phi(i)(x) = x \times C_i \leq b_i$. Recall that CMC S is deterministic if and only if for all states i, j such that $i < j$, $V(i) \cap V(j) \neq \emptyset$ implies for all k, $\{x | x \times C_k \leq b_k \wedge x_i = 0\} = \emptyset$ or $\{y | y \times C_k \leq b_k \wedge y_j = 0\} = \emptyset$. This can be decided in polynomial time using Fourrier-Motzkin elimination [163].

3.7 Related Work and Concluding Remarks

We have presented Constraint Markov Chains—a new model for representing a possibly infinite family of Markov Chains. Unlike the previous attempts [113, 91], our model is closed under many design operations, including composition and conjunction. We have studied these operations as well as several classical compositional reasoning properties, showing that, among others, the CMC specification theory is equipped with a complete refinement relation (for deterministic specifications), which naturally interacts with parallel composition, synchronization and conjunction.

Two recent contributions [91, 112] are strongly related to these results. Fecher et al. [91] propose a definition of weak refinement for Interval Markov Chains that is coarser than the refinement defined in [113] (see also Definition 3.3.2 here). They also give a model checking procedure for PCTL [67] and Interval Markov Chains. Our definition of weak refinement coincides with theirs for Interval Markov Chains, which are a subclass of CMCs. Very recently Katoen and coauthors [112] have extended Fecher's work to *Interactive* Markov Chains, a convenient model for performance evaluation [101, 104]. Their abstraction uses the continuous time version of Interval Markov Chains [114] augmented with may and must transitions, very much in the spirit of [119, 148]. Parallel composition is defined and studied for this abstraction, however conjunction has been studied neither in [91] nor in [112].

In future, it would be of interest to design, implement and evaluate efficient algorithms for procedures outlined in this chapter. We would also like to define a quotient relation for CMCs, presumably building on results presented in [120]. The quotienting operation is of particular importance for component reuse. One could also investigate applicability of our approach in model checking procedures, in the same style as Fecher and coauthors have used Interval Markov Chains for model checking PCTL [91]. Finally the model presented in [112] can probably be extended from intervals to more general constraints.

Part II

Heterogeneous Systems

Chapter 4

Asynchronous Implementation of Synchronous Specifications

Résumé : *La prise en compte de la concurrence, de la communication et des dépendences causales en général est une question centrale pour la conception des systèmes embarqués en réseaux.*

Le paradigme de la programmation synchrone est maintenant reconnu comme l'un des standards industriels du domaine. On le retrouve à plusieurs niveaux de la chaîne de conception des systèmes embarqués de la conception système à la programmation de systèmes de contrôle. Le déploiement efficace de programmes synchrones sur des architectures réparties est cependant un problème difficile mais essentiel pour la programmation à au niveau de ces architectures pour des applications de contrôle. Cette problématique se retrouve dans de nombreuses applications, pour les quelles le processus de conception tient compte du caractère réparti de l'architecture. Cet ainsi que pour les systèmes de commandes de vol mis en œuvre sur les avions Airbus récents, les règles de conception assurent la répartition et donc la désynchronisation des programmes synchrones, sans adjonction de protocoles complexes. À l'autre extrémité du spectre des architectures VLSI constituent aussi un domaine d'application intéressant pour la programmation synchrone. La prééminance des architectures dites network on chips *introduit un fort degé d'asynchronie dans les circuits.*

Dans ces deux cas, Il semble particulièrement utile de savoir transformer un programme synchrone en un réseau asynchrone d'îlots synchrones. Ce sont les architectures localement synchrones, globalement asynchrones *(GALS). C'est une question délicate, et à défaut de pouvoir générer automatiquement les schémas de communication et des synchronisation, il semble particulièrement utile de pouvoir prouver qu'un réseau synchrone de programmes synchrones peut être déployé sur une architecture asynchrone, sans adjonction de synchronisation, mais tout en préservant la sémantique du réseau de programmes.*

Il faut bien se représenter, que le problème de déployer un réseau synchrone sur une architec-

ture asynchrone est uniquement un problème d'optimisation, puisqu'il admet une solution triviale, mais totalement inefficace, tant en terme de communication, que d'utilisation des ressources de calcul. Le véritable problème est plutôt une question de synthèse d'un schéma de synchronisation qui optimise une fonction de coût tenant compte de la communication et du parallélisme du système GALS résultant.

Nous n'avons pas regardé cette question à proprement parler. Nous avons cherché à caractériser l'espace des solutions, le problème d'optimisation relevant plutôt de techniques heuristiques pour des problèmes d'optimisation combinatoire.

Nous avons regardé le problème suivant: étant donné un réseau de programmes synchrones, décider si son déploiement sur une architecture répartie admet les mêmes comportements que le réseau synchrone. Ce problème est indécidable, même quand les programmes synchrones se réduisent a des automates finis.

Nous avons donc considéré des conditions suffisantes au déploiement correct. Plutôt que de se ramener, par la force brute, à un problème de vérification sur la composition asynchrone des programmes synchrones, approche non modulaire qui aboutit en général à un système infini, nous avons recherché des méthodes modulaires ne reposant pas sur la vérification globale, mais plutôt sur une approche modulaire, reposant sur la vérification de relations binaires entre programmes synchrones, pour lesquelles un passage à l'échelle peut être escompté.

*Les conditions suffisantes que nous avons proprosé sont en fait la conjonction de deux propriétés distinctes. La première, dite d'*isochronie*, est locale à un programme synchrone, et est une propriété de déterminisme d'un programme, quand on désynchronise ses entrées. La seconde, dite d'*endochronie *est une propriété portant sur un réseau de programmes et reflète une sorte compatibilité entre programmes synchrones. Ces deux propriétés ont été proposées avec deux variantes, l'une forte, fondée sur une sémantique macro-pas des programmes synchrones (dans laquelle une réaction est décrite par une seule transition), l'autre, plus faible, repose sur une sémantique micro-pas (dans laquelle une réaction est représentée par une suite d'affectations de variables, terminée par une transition spéciale, marquant le passage à la réaction suivante).*

Ces travaux ont été initiés en 1999 à l'occasion d'une collaboration avec Albert Benveniste [21, 23]. Elle s'est ensuite poursuivie en collaboration avec Dumitru Potop-Butucaru, alors postdoc à Rennes [74, 143, 147]. Dumitru Potop-Butucaru, depuis devenu chercheur à l'INRIA à Rocquencourt dans léquipe Aoste, a poursuivi ces travaux, en les orientant vers la recherche de méthode heuristiques pour la desynchronisation correcte de réseaux de programmes synchrones [145, 146]. Cette même problématique a été reprise dans les travaux de thèse de Julien Ouy (doctorat effectué dans l'équipe Espresso à Rennes), mais avec application au langage de programmation Signal [139].

4.1 Introduction

Dealing with concurrency, time and causality in the design of electronic systems has become increasingly difficult as the complexity of the designs grew.

The *synchronous programming model* [98, 26, 144] has had major successes at the specification level because it provides a simpler way to employ the power of concurrency in functional specification. Provided that a few high-level constraints ensure compliance with the synchrony hypothesis, the designer can forget about timing and communication issues and concentrate on functionality. The synchronous model features deterministic concurrency and simple composition mechanisms facilitating the incremental development of large systems. Also, synchronous models are usually easier to analyze/verify/optimize compared to asynchronous counterparts, often because the state-transition representations are smaller.

Synchronous languages like ESTEREL, LUSTRE, and SIGNAL, the quasi-synchronous STATECHARTS modeling methodology, and design environments like SIMULINK / STATEFLOW all benefit from the simplicity of the *synchronous hypothesis*:

1. Cycle-based execution model. Behaviors are sequences of *reactions* indexed by a *global logical clock*.
2. Within each reaction, the behavior is non-divergent and causal, so that the status of every signal is defined prior to being used in computations.

Note that condition 2 empowers the conceptual abstraction that computations and communications are infinitely fast (“zero-time”) and take place at discrete points in time, with no duration. It also allows universally-recognized mathematical models like the Mealy machines and the digital circuits to be used as semantic foundations.

Eventhough the synchronous assumption simplifies system specification and verification, the problem of deriving a correct physical implementation from it does remain [26]. In particular, difficulties arise when the target implementation architecture has a distributed nature that does not match the synchronous assumption because of large variance in computation and communication speeds and because of the difficulty of maintaining a global notion of time. This is increasingly the case in complex microprocessors and Systems-on-a-Chip (SoC), and for many important classes of embedded applications in avionics, industrial plants, and the automotive industry.

For instance, many industrial embedded applications consist of multiple processing elements, operating at different rates, distributed over an extended area, and connected via communication buses. To use a synchronous approach in the development of such applications, one solution is to replace the asynchronous buses with communication infrastructures that comply with a notion of global synchronization. This is examplified by the family of Timed-Triggered Architectures introduced and promoted by H. Kopetz [118]. However, such a fully synchronous implementation must be conservative, forcing the global clock to run as slow as the slowest computation/communication process. The overhead implied by time-triggered architectures and synchronous implementations is often large enough to convince designers to use asynchronous solutions.

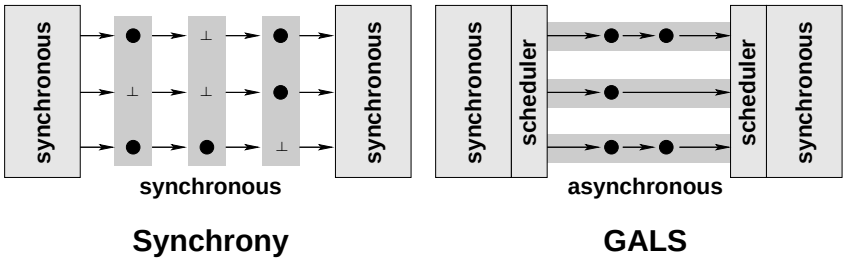


Figure 4.1: From synchrony to GALS. Bullets represent informative values (messages). Vertical gray boxes represent reactions. Horizontal ones represent asynchronous signals.

Gathering advantages of both the synchronous and asynchronous approaches, Globally Asynchronous Locally Synchronous (GALS) architectures are emerging as an architecture of choice for implementing complex specifications in both hardware and software. In a GALS system, locally-clocked synchronous components are connected through asynchronous communication lines. Thus, unlike for a purely asynchronous design, the existing synchronous tools can be used for most of the development process, while the implementation can exploit the more efficient/unconstrained/required asynchronous communication schemes.

We further pursue in this chapter our quest for correct-by-construction deployment of synchronous specifications over GALS architectures.

4.1.1 Informal discussion of the issues

In the *synchronous paradigm* [98, 26, 144], an execution of the program, also called *trace*, is a sequence of reactions, each reaction assigning a unique value (status) to each variable of the program. Not all variables need to be involved in each reaction. However, this is taken into account by extending the domain of values of all variables with an extra symbol $\perp$, which denotes absence. Thus, absence can be tested and used to exercise control.

No global clock exists in the *asynchronous paradigm*, meaning that no notion of reaction exists, and that absence ($\perp$) has no meaning and cannot be sensed. Only the sequences of values on individual channels can be observed, so that an *asynchronous observation* of the execution of a system is a function assigning to each communication channel the sequence of transmitted messages/values. Asynchronously observing a synchronous execution consists of removing the $\perp$ events and the synchronization boundaries of the reactions to obtain an asynchronous observation.

In many cases, applications designed in a synchronous framework will be implemented for use in an asynchronous environment. Two problems arise: First, the synchronous applications must be fitted with wrappers that read the asynchronous inputs and schedule them into reactions before giving them to the program and triggering the program clock (the scheduling operation inserts the missing $\perp$ values). As the synchronous paradigm is often used in the development of

safety-critical systems, input reading and the system itself must be deterministic, or at least predictable. It is therefore essential to consider classes of synchronous specifications that facilitate the development of efficient wrappers which make input reading deterministic while not restricting the behavior of the system.

Second, the implementation must preserve the semantics of the synchronous specification, meaning that the set of asynchronous observations of the specification must be identical to the set of observations of the implementation. Preservation of semantics is important because the advantages of synchrony lie with specification and verification. We would therefore like each implementation trace to be covered by the verification of the synchronous model. This problem is of particular importance when the synchronous specification must be implemented over a distributed architecture (an operation called *desynchronization*). In such cases, input reading and computation must be coordinated between distributed sites, and doing this without a careful analysis can be very inefficient (in terms of speed, consumption, communication, etc.) or simply incorrect.

This chapter addresses the problem of desynchronizing a modular synchronous specification by replacing the communication lines between modules with asynchronous FIFOs. Instead of a single, global wrapper, we shall have one wrapper per system component, as pictured in fig. 4.1. The exact problem we address is that of *characterizing large classes of synchronous components for which small, simple wrappers* [1] *produce deterministic, efficient, and semantics-preserving GALS implementations*. These classes of systems can then be considered as the implementation space, and the remaining problem is that of making given synchronous systems belong to these classes (by adding supplementary signaling). Naturally, a larger implementation space covers better solutions that use less synchronization.

4.1.2 Previous work

Previous approaches to implementing modular synchronous specifications over GALS architectures are respectively based on *latency-insensitive systems*, on *Kahn process networks (KPN)*, and on *endochronous and isochronous systems*.

In the *latency-insensitive systems* of Carloni *et al.* [59], each synchronous component reads every input and writes every output at each reaction. The communication protocols effectively simulate a single-clock system, which is inefficient, but simplifies the implementation.

In a *Kahn process network* [129], requiring that each component has a deterministic input/output behavior implies the determinism of the global system (and thus any wrapper is a good one). Often used, due to its robustness, in the development of embedded systems, the KPN-based approach has been adapted by Caspi *et al.* for the desynchronization of functional dataflow synchronous specifications [61]. Giving the approach its strength, the determinism is also its main drawback, as non-determinism is often useful in the specification and analysis of concurrent systems. We also mention here the approach of Talpin *et al.* [167], which is based on a bounded version of the Kahn principle.

1. For instance, wrappers that trigger a transition as soon as the needed input is available.

The approach based on *endo/isochronous systems* has been proposed by Benveniste *et al.* [23] in order to support the analysis of partial specifications (which can be non-deterministic, or incomplete), to exploit execution modes, and to cover truly concurrent and multi-clock implementations. Informally speaking, a synchronous component is endochronous when the presence and absence of each variable can be inferred incrementally during each reaction from the current state and from the values of already present values. An endochronous component knows how to read its inputs, meaning that no wrapper is needed. Unfortunately, endochronous components can exhibit no internal concurrency, which makes endochrony non-compositional (thus, incremental system development is impossible). Isochrony is a semantics-preservation criterion over pairs of synchronous systems. The work of Singh and Theobald on *generalized latency-insensitive systems* [165] can be seen as implementing endochrony in hardware.

Essential improvement is brought by the work by Potop *et al.* [143] on weak endochrony and weak isochrony. Weak endochrony extends endochrony by allowing operations within a component to run independently when no synchronization is necessary. The notion is compositional, allowing incremental development of large systems. Being formulated in a non-causal [2] framework, this approach is also less constrained than the KPN-based one, allowing non-determinism in the less abstract causal model. The non-causal framework is also the main disadvantage, because it hides implementation properties, like the presence of synchronization or communication deadlocks (which are important in practice).

The distribution of synchronous or strongly synchronized specifications has been studied in many other settings. We only mention here the Time-Triggered Architectures of Kopetz [118], the `ocrep` tool of Girault *et al.* [61], the AAA methodology of Sorel [97], and the desynchronization approach of Cortadella *et al.* [42].

From a more theoretical point of view, our work is closely related to results related to the confluence of asynchronous system models [116]. In this sense, our work is closely related to results concerning the design of delay-insensitive [173, 132, 71], speed-independent [115], and burst-mode [179] circuits (we will come back with a comparison in section 4.4.3).

4.1.3 Contribution

This chapter brings an important improvement over previous work, by allowing us to *reason about concurrency and efficient synchronization in a causal, operational synchronous framework that takes into account the composition through read/write mechanisms*. The approach inherits the advantages of the weak endochrony-based approach: It allows the representation of non-deterministic specifications, takes into account execution and communication modes, and covers concurrent and multi-clock implementations. At the same time, it allows us to reason in a unified model about semantics-preservation and the absence of deadlocks due to synchronization and com-

2. The term *causal/causality* covers here the execution order of the various operations that make up a synchronous reaction. The formalism presented in this chapter has the means of representing this order. Other formalisms, including those of [23, 143], do not.

munication (which are both essential correctness properties of any implementation). As we shall see, *the level of detail is essential in this analysis, as it reveals the strong ties that exist between the two correctness properties, and simplifies the correctness analysis*.

Our main contribution is the definition of a new model for the representation of asynchronous implementations of synchronous specifications. The model covers classical implementations, where a notion of global synchronization is preserved by means of signaling, and globally asynchronous, locally synchronous (GALS) implementations where the global clock is removed. *We use this model to derive criteria ensuring the correct deployment of synchronous specifications over GALS architectures*.

4.1.4 Outline

The remainder of the chapter is organized as follows: Section 4.2 defines the formal framework used throughout the chapter, and section 4.3 gives intuitive examples and explains why the new structures are adapted to modeling and reasoning about the correctness of GALS implementations of synchronous specifications. Sections 4.4 defines criteria ensuring correct desynchronization. A short conclusion is given in section 4.5.

4.2 The model

This section defines our model of asynchronous implementation of a synchronous specification. We structured its presentation into several parts. The subsections 4.2.1, 4.2.2, and 4.2.3 introduce rather standard notations for transition systems (labels, traces, concurrent transition systems, and composition by synchronized product). Subsection 4.2.4 is the first to define communication channels, clocks, and the I/O transition systems which form our basic implementation model. Section 4.2.5 defines the synchronous transition systems – which are I/O transition systems satisfying the synchronous hypothesis. In section 4.2.6 we explain how transition systems are synchronously and asynchronously composed using FIFO models. Recall that intuitive examples are given later, in section 4.3.

4.2.1 Variables and labels

Our components and systems interact with each other and with their environment through *variables*. The *domain* of a variable v is denoted with $\mathcal{D}_v$. Given V a *finite* set of variables, a *label* over V is a partial valuation of its variables. Formally, the set of all labels over V is $\mathcal{L}_V = \prod_{v \in V} \mathcal{D}_v^{\perp}$, where $\mathcal{D}_v^{\perp} = \mathcal{D}_v \cup \{\perp\}$, and $\perp \notin \mathcal{D}_v$ is a special symbol denoting the *absence* of a value. The *support* of a label $l \in \mathcal{L}_V$ is $supp(l) = \{v \in V \mid l(v) \neq \perp\}$. We denote with $\perp_V$ the label of empty support over V. For simplicity, we shall usually write out a label as the set of its non-absent variable valuations. For instance, $< v = 0, u = 1 >_V$ denotes the label over V with support $\{u, v\}$ and which assigns 0 to v and 1 to u. When confusion is not possible, the set V of variables can

be omitted from the notation. Also, when confusion is not possible and we need to save space (for instance in large system representations) we shall drop the "$<\ >$" delimiters.

If $l \in \mathcal{L}_V$ and V' is another set of variables, then the *image* of l through V' is the label $l\mid_{V'} \in \mathcal{L}_{V'}$ that equals l over $V \cap V'$ and equals $\perp$ on $V' \setminus V$.

The labels $l_i \in \mathcal{L}_{V_i}, i = 1, 2$, are *non-contradictory*, denoted $l_1 \bowtie l_2$, if for all $v \in V_1 \cap V_2$ such that $l_i(v) \neq \perp, i = 1, 2$ we have $l_1(v) = l_2(v)$. In this case, we define their:

- *union:* $l_1 \sqcup l_2 \in \mathcal{L}_{V_1 \cup V_2}$, of support $supp(l_1) \cup supp(l_2)$ and which equals l_i over $supp(l_i), i = 1, 2$.
- *intersection:* $l_1 \sqcap l_2 \in \mathcal{L}_{V_1 \cup V_2}$, of support $supp(l_1) \cap supp(l_2)$ and which equals l_i on its support.

The union and intersection operators are associative and commutative. When the labels $l_1, l_2 \in \mathcal{L}_V$ are non-contradictory, we also define their *difference* $l_1 \setminus l_2 \in \mathcal{L}_V$ by $l_1 \setminus l_2(u) = l_1(u)$, if $u \notin supp(l_2)$ and $\perp$, otherwise.

When the non-contradictory labels $l_i \in \mathcal{L}_{V_i}, i = 1, 2$ are equal over $V_1 \cap V_2$, they are called *synchronizable*. Their union is also called in this case *product* and denoted with $l_1 \otimes l_2$. Note that $l_1 \otimes l_2$ equals l_i on V_i, $i = 1, 2$. The labels $l_i \in \mathcal{L}_{V_i}, i = 1, 2$ are *disjoint* if $supp(l_1) \cap supp(l_2) = \emptyset$.

Assume that v and v' are variables with the same domain (*i.e.* $\mathcal{D}_v = \mathcal{D}_{v'}$). Given $l \in \mathcal{L}_V$, with $v \in V$ and $v' \notin V \setminus \{v\}$, the name change operator associates $l[v/v'] \in \mathcal{L}_{V \setminus \{v\} \cup \{v'\}}$ with:

$$l[v/v'](u) = \begin{cases} l(u), \text{ if } u \neq v' \\ l(v), \text{ if } u = v' \end{cases}$$

We define the "sub-label" partial order relation $\leq$ over $\mathcal{L}_V$: $l_1 \leq l_2$ if $\forall v : (l_1(v) \neq \perp \Rightarrow l_1(v) = l_2(v))$.

4.2.2 Traces

A *trace* over the set of variables V is a *finite* sequence of labels over V. The set of all traces over V is denoted with $Traces(V) = \mathcal{L}_V^* = \{(l_i)_{0 \leq i < n} \mid n \in \mathbb{N} \wedge \forall i : l_i \in \mathcal{L}_V\}$. Given a trace $\varphi = (l_i)_{0 \leq i < n}$ we denote $length(\varphi) = n$ and $\varphi[i] = l_i$. Note that any label is a trace of length 1. We denote with ϵ any sequence of length 0. In particular ϵ denotes the empty trace, regardless of the variable set.

Any two traces $\varphi_1, \varphi_2 \in Traces(V)$ can be concatenated (by juxtaposition $\varphi_1\varphi_2$). The trace φ_1 is a prefix of φ_2 (written $\varphi_1 \preceq \varphi_2$) if, by definition, $\varphi_2 = \varphi_1\varphi_3$ for some φ_3. The prefix relation is a partial order over traces. The image operator is extended component-wise on traces: $(l_i)_{0 \leq i < n}\mid_{V'} = (l_i \mid_{V'})_{0 \leq i < n}$. The traces $\varphi_i \in Traces(V_i), i = 1, 2$ are called *synchronizable* if $length(\varphi_1) = length(\varphi_2)$ and if for all j the labels $\varphi_1[j]$ and $\varphi_2[j]$ are synchronizable. In this case, we can define the *product trace* $\varphi_1 \otimes \varphi_2 = (\varphi_1[i] \otimes \varphi_2[i])_{0 \leq i < length(\varphi_1)}$. The product operator is associative and commutative. The *support* of a trace φ, denoted $supp(\varphi)$, is the union of the supports of its labels.

4.2.3 Generalized concurrent transition systems

The *generalized concurrent transition systems* (GCTS) form our (asynchronous) implementation model. GCTSs are step transition systems where steps are syntactic representions of the concurrency between atomic operations (which assign or test a single variable). They generalize the concurrent transition systems of Stark [166], and can be seen as a sub-set of the step transition systems of Mukund [135].

Definition 4.2.1 (generalized concurrent transition system) *A generalized concurrent transition system (GCTS) is a tuple $\Sigma = (S,\ \hat{s},\ V,\ \circ\!\!\rightarrow_\Sigma)$, where S is the set of states (not necessarily finite) , $\hat{s} \in S$ is the initial state, V is the finite set of communication variables, and $\circ\!\!\rightarrow_\Sigma \subseteq S \times \mathcal{L}_V \times S$ is a transition relation satisfying:*

GCTS1 *(void transition): $\forall s \in S :\ s \overset{\perp_V}{\underset{\Sigma}{\circ\!\!\rightarrow}} s$.*

GCTS2 *(prefix closure): If $s \overset{l}{\underset{\Sigma}{\circ\!\!\rightarrow}} s'$ and $l' \leq l$, then there exists $s'' \in S$ such that $s \overset{l'}{\underset{\Sigma}{\circ\!\!\rightarrow}} s''$ and $s'' \overset{l \backslash l'}{\underset{\Sigma}{\circ\!\!\rightarrow}} s'$.*

When there is no ambiguity, Σ can be dropped from the transition relation notation.

We shall say that $\varphi = l_1 \ldots l_n \in Traces(V)$ is a trace of the GCTS $\Sigma = (S,\ \hat{s},\ V,\ \circ\!\!\rightarrow_\Sigma)$ starting in the state $s \in S$ if there exist $s_1, \ldots, s_n \in S$ such that $s \overset{l_1}{\circ\!\!\rightarrow} s_1 \overset{l_2}{\circ\!\!\rightarrow} \ldots \overset{l_n}{\circ\!\!\rightarrow} s_n$. In this case, we also write $s \overset{\varphi}{\circ\!\!\Longrightarrow} s_n$. The set of all traces of Σ starting in s is denoted by $Traces_\Sigma(s)$, and the set of all destination states of such traces is: $RSS_s(\Sigma) = \{s' \in S \mid \exists \varphi :\ s \overset{\varphi}{\circ\!\!\Longrightarrow} s'\ \}$. The *reachable state space of* Σ is $RSS(\Sigma) =_{def} RSS_{\hat{s}}(\Sigma)$.

Generalized concurrent transition systems are composed by means of synchronized product. Consider two GCTSs $\Sigma_i = (S_i,\ \hat{s_i},\ V_i,\ \circ\!\!\rightarrow_{\Sigma_i}), i = 1, 2$, then their product is defined as follows:

$$\Sigma_1 \otimes \Sigma_2 = (S_1 \times S_2,\ (\hat{s_1}, \hat{s_2}),\ V_1 \cup V_2,\ \circ\!\!\rightarrow_{\Sigma_1 \otimes \Sigma_2})$$

where $(s_1, s_2) \overset{l}{\underset{\Sigma_1 \otimes \Sigma_2}{\circ\!\!\rightarrow}} (s_1', s_2') \ \Leftrightarrow\ s_i \overset{l|_{V_i}}{\underset{\Sigma_i}{\circ\!\!\rightarrow}} s_i' , i = 1, 2$. The $\otimes$ operator is well-defined — It preserves the properties GCTS1 and GCTS2. It is also associative and commutative, and:

$$Traces_{\bigotimes_{i=1}^n \Sigma_i}((s_i)_{1 \leq i \leq n}) = \{\bigotimes_{i=1}^{n} \varphi_i \mid \varphi_i \in Traces_{\Sigma_i}(s_i) \text{ pairwise synchronizable}\}$$

The variable name change operator is extended to GCTSs. If $\Sigma = (S,\ \hat{s},\ V,\ \circ\!\!\rightarrow_\Sigma)$, $v \in V$, $v' \notin V \setminus \{v\}$, and $\mathcal{D}_v = \mathcal{D}_{v'}$, then:

$$\Sigma[v/v'] = (S,\ \hat{s},\ V \setminus \{v\} \cup \{v'\},\ \{\ s \overset{l[v/v']}{\underset{\Sigma[v/v']}{\circ\!\!\rightarrow}} s' \ \mid\ s \overset{l}{\underset{\Sigma}{\circ\!\!\rightarrow}} s'\ \})$$

4.2.4 I/O causality. Channels and clocks

In practice, communications between the different components of a system are directed. One component emits a value on a channel, and another reads it. To take this into account, we use *directed communication channels* that are pairs of *directed variables*. We emit a value on a channel c by assigning the variable $!c$, and we receive a value by reading the variable $?c$. The variables $!c$ and $?c$ have the same domain, denoted with $\mathcal{D}_c$. We denote with $\mathcal{C}(V) = \{c \mid !c \in V \text{ or } ?c \in V\}$ the set of channels associated with a set of variables V. To simplify the model, we assume that every channel connects at most one emitter with at most one receiver, meaning that $!c$ is variable of at most one component in the system, the same holding for $?c$ (a simple renaming technique allows the use of multicast, but we shall not cover the subject here). We further assume that the only variables that are not directed are the *clocks* of the synchronous components. A clock is a variable whose domain is $\mathcal{D}_{clk} = \{\top\}$ ($\top$ stands for the "clock tick"). Given a set V of variables we shall denote with $Clocks(V)$ the subset of clock variables, and with $Directed(V) = V \setminus Clocks(V)$ the subset of directed variables. To simplify the notations, we abbreviate the clock tick valuation $\tau = \top$ with τ (for any clock variable τ).

Definition 4.2.2 (I/O transition system) *We say that a GCTS is an I/O transition system when all its variables are either directed or clocks.* ***From now on, this chapter only considers I/O transition systems.***

To reason about desynchronization properties, we shall need the following function, which removes the clock synchronization barriers, so that only messages (and not absence) are visible, along with message ordering on each channel: $\delta : (\mathcal{D}_v^{\perp})^* \to \mathcal{D}_v^*$, defined by $\delta(\epsilon) = \epsilon$ and:

$$\delta(v\varphi) = \begin{cases} v\delta(\varphi), \text{ if } v \neq \perp \\ \delta(\varphi), \text{ otherwise} \end{cases}$$

Using this notation, we extend the relation $\leq$ (first defined on labels) to a preorder over $Traces(V)$: Given $\varphi_1, \varphi_2 \in Traces(V)$, we write $\varphi_1 \leq \varphi_2$ whenever we have $\delta(\varphi_1 \mid_{\{v\}}) \preceq \delta(\varphi_2 \mid_{\{v\}})$ for all $v \in Directed(V)$. If $\varphi_1 \leq \varphi_2$ and $\varphi_2 \leq \varphi_1$, then we say that φ_1 and φ_2 are *asynchronously equivalent*, denoted $\varphi_1 \sim \varphi_2$. When for all $v \in Directed(V)$ we have $\delta(\varphi_1 \mid_{\{v\}}) \preceq \delta(\varphi_2 \mid_{\{v\}})$ or $\delta(\varphi_2 \mid_{\{v\}}) \preceq \delta(\varphi_1 \mid_{\{v\}})$, then we say that φ_1 and φ_2 are *asynchronously non-contradictory*, and write $\varphi_1 \bowtie \varphi_2$. Note that $\bowtie$ extends to traces the non-contradiction relation over labels. Moreover, we can extend the label difference operator to non-contradictory traces by defining the asynchronous difference of traces $\varphi_1 \setminus \varphi_2$ by induction:

$$\begin{cases} \varphi_1 \setminus (l\varphi_2) = (\varphi_1 \setminus l) \setminus \varphi_2 \\ (l_1\varphi_1) \setminus l_2 = (l_1 \setminus l_2)(\varphi_1 \setminus (l_2 \setminus l_1)) \end{cases}$$

If φ is a trace of an I/O transition system, then we denote with $\mid \varphi \mid$ the number of assignments of non-clock variables contained in φ.

4.2.5 Synchronous transition systems

Our synchronous transition systems represent causal synchronous specifications or, equivalently, implementations of synchronous specifications where the global clock is preserved by some communication infrastructure by means of added signalization [3]. A synchronous transition system is an I/O transition system with a single clock variable, and satisfying the synchronous hypothesis and a stuttering-invariance property (which is necessary if we want to derive GALS implementations).

Definition 4.2.3 (synchronous transition system) *A microstep synchronous transition system (for short μSTS) is a tuple $\Sigma = (S, \hat{s}, V, \tau, \circ\!\!\rightarrow)$ where all the variables of V are directed, where τ is a clock variable (the clock of the component), and where $(S, \hat{s}, V \cup \{\tau\}, \circ\!\!\rightarrow)$ is a GCTS satisfying:*

μ**STS1** *(clock transitions): if $s \overset{l}{\circ\!\!\rightarrow} s'$ and $l(\tau) \neq \bot$ then $l\,|_V = \bot_V$.*

μ**STS2** *(stuttering-invariance): $\hat{s} \overset{<\tau>}{\circ\!\!\rightarrow} \hat{s}$ and ($s \overset{<\tau>}{\circ\!\!\rightarrow} s' \;\Rightarrow\; s' \overset{<\tau>}{\circ\!\!\rightarrow} s'$)*

μ**STS3** *(single assignment): two assignments of a same variable must be separated by a clock transition. More exactly, if $s_0 \overset{l_1}{\circ\!\!\rightarrow} s_1 \overset{l_2}{\circ\!\!\rightarrow} \ldots \overset{l_n}{\circ\!\!\rightarrow} s_n$ and $\forall i : l_i \neq \tau$, then $l_1, \ldots, l_n$ are pairwise disjoint.*

Note that axiom μSTS1 identifies the *clock transitions* – with label $< \tau >$ – which are the only transitions where the clock variable is present. Such transitions separate synchronous reactions during which a variable cannot be assigned more than once (cf. axiom μSTS3). A state which is destination of a clock transition is called *synchronizing state*. Given a trace φ of a synchronous system, we can decompose it into reactions $\varphi = Step_0(\varphi) < \tau > Step_1(\varphi) < \tau > \ldots$ where each reaction $Step_i(\varphi)$ contains no clock transition. As the transitions of each $Step_i(\varphi)$ are disjoint, we can denote with $< Step_i(\varphi) >$ the union of all its labels. We shall say that a trace φ is *complete* if it ends with a $< \tau >$ transition. We say that a μSTS is *non-blocking* if from any reachable state there is a path towards a stuttering state. Note that in a non-blocking μSTS any trace can be completed. Blocking systems are considered incorrect.

An isomorphism λ between two GCTSs $\Sigma_i = (S_i, \hat{s}_i, V_i, \circ\!\!\rightarrow_{\Sigma_i}), i = 1, 2$ consists of two bijections $\lambda^S : S_1 \rightarrow S_2$ and $\lambda^V : V_1 \rightarrow V_2$ having the properties: (i) $\forall v : \mathcal{D}_v = \mathcal{D}_{\lambda^V(v)}$, (ii) $\lambda^S(\hat{s_1}) = \hat{s_2}$, and (iii) $s \overset{l}{\underset{\Sigma_1}{\circ\!\!\rightarrow}} s' \;\Leftrightarrow\; \lambda^S(s) \overset{\lambda(l)}{\underset{\Sigma_2}{\circ\!\!\rightarrow}} \lambda^S(s')$, where $\lambda(l)$ denotes the label obtained from l by renaming v with $\lambda^V(v)$ for all $v \in V_1$. If Σ_1 and Σ_2 are I/O transition systems, we say that λ is an isomorphism of I/O transition systems if λ^V maps read variables onto read variables, write variables onto write variables, and clocks onto clocks. If Σ_1 and Σ_2 are μSTSs, then λ is an isomorphism of μSTSs if it is an isomorphism of I/O transition systems.

3. Such as in the *Time-Triggered Architectures* of Kopetz[118].

4.2.6 Synchronous and asynchronous composition

As earlier mentioned, we simplify the model by only allowing point-to-point communication, and we enforce this rule by syntactic means. However, broadcast can be simulated by replicating and renaming variables.

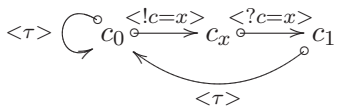
Definition 4.2.4 (composable transition systems) *We say that the I/O transition systems* $\Sigma_i, i = \overline{1,n}$ *are* composable *if their variable sets are mutually disjoint.*

Note that the definition requires not only point-to-point communications (no directed variable is shared by two or more systems), but also the non-overlapping of clock sets (which is natural). Also note that a system can have both $!c$ and $?c$ as variables, thus allowing the representation of systems obtained by composition.

The composition of synchronous and asynchronous systems is defined by means of synchronized product, using FIFO models to represent communication through synchronous and asynchronous channels. To represent synchronous communication, we use 1-place synchronous FIFO models (which are μSTSs themselves). The FIFO model associated with a channel c is:

$$SFIFO(c,\tau) = \\ (\{c_0, c_1\} \cup \bigcup_{x\in\mathcal{D}_c} \{c_x\},\ c_0,\ \bigcup_{x\in\mathcal{D}_c} \{!c = x, ?c = x\},\ \tau,\ \circ\!\!\rightarrow_S)$$

where the transition relation is defined by:



Note that modeling multicast communication (a feature that will not be addressed in this chapter), can simply be done by renaming channel read variables in a component-wise fashion, and then modifying the FIFO model to allow the concurrent read of the value from different sites.

Asynchronous communication involves infinite asynchronous FIFO models (which are not μSTSs):

$$AFIFO(c) = (\mathcal{D}_c^*,\ \epsilon,\ \bigcup_{x\in\mathcal{D}_c} \{!c = x, ?c = x\},\ \circ\!\!\rightarrow_A)$$

where the transition relation contains all the transitions of the form:

$$x_1 \ldots x_n \xrightarrow{<!c=x_{n+1}>} x_1 \ldots x_n x_{n+1} \xrightarrow{<?c=x_1>} x_2 \ldots x_{n+1}$$

Definition 4.2.5 (synchronous composition of μSTSs) *Let $\Sigma_i = (S_i, \hat{s}_i, V_i, \tau_i, \circ\!\!\rightarrow_{\Sigma_i}), i = 1, 2$ be composable μSTSs and let τ be a clock variable. Then, the synchronous composition of Σ_1 and Σ_2 over the base clock τ is:*

$$\Sigma_1 \ |^{\tau} \ \Sigma_2 = \Sigma_1[\tau_1/\tau] \otimes \Sigma_2[\tau_2/\tau] \otimes \bigotimes_{c \in \mathcal{C}(V_1) \cap \mathcal{C}(V_2)} SFIFO(c, \tau)$$

Lemma 4.2.6 (Properties of the synchronous composition) *The synchronous composition of the μSTSs Σ_1 and Σ_2 over the base clock τ is a μSTS of clock τ. The result of the synchronous composition is unique upto renaming of the base clock, so that we can omit the base clock τ from the notation. Moreover, the operator $|$ is associative and commutative, modulo isomorphism.*

In addition, note that synchronizing states of $|_{i=1}^{n}\Sigma_i$ have void communication lines (all synchronous FIFO models are in their unique synchronizing state).

Definition 4.2.7 (asynchronous composition of I/O systems) *Let $\Sigma_i = (S_i, \hat{s}_i, V_i, \circ\!\!\rightarrow_{\Sigma_i}), i = 1, 2$ be composable I/O transition systems. Then, the asynchronous composition of Σ_1 and Σ_2 is:*

$$\Sigma_1 \ \| \ \Sigma_2 = \Sigma_1 \otimes \Sigma_2 \otimes \bigotimes_{c \in \mathcal{C}(V_1) \cap \mathcal{C}(V_2)} AFIFO(c)$$

Lemma 4.2.8 (asynchronous composition properties) *The asynchronous composition of I/O transition systems results in another I/O transition system. The $\|$ operator is associative and commutative. The asynchronous composition of two μSTSs is not a μSTS.*

Proof:(lemmas 4.2.6 and 4.2.8) The operator $\otimes$ is associative and commutative, which implies the associativity and commutativity of the synchronous and asynchronous composition operators. The isomorphism of $\Sigma_1 \ |^{\tau_1} \ \Sigma_2$ and $\Sigma_1 \ |^{\tau_2} \ \Sigma_2$ is given by the renaming of the clock variable. □

4.2.7 Product states and product traces

Note that the state of a synchronous or asynchronous product of I/O systems is not only given by the state of the components, but also by the state of its communication channels. Indeed, given the composable μSTSs $\Sigma_i, i = \overline{1,n}$, connected through the channels $c_i, i = \overline{1,m}$, the state of $|_{i=1}^{n}\Sigma_i$ is $((s_i)_{i=\overline{1,n}}, (c_i^s)_{i=\overline{1,m}})$, and the state of $\|_{i=1}^{n}\Sigma_i$ is $((s_i)_{i=\overline{1,n}}, (c_i^a)_{i=\overline{1,m}})$, where c_i^s denotes states of $SFIFO(c_i, \tau)$ and c_i^a denotes states of $AFIFO(c_i)$.

Nevertheless, for space reasons, we shall consider in this article only examples where the tuple $(s_i)_{i=\overline{1,n}}$ unambiguously identifies the state of the product. Thus, we can use the component state tuple alone to label states.

As should be expected, the synchronous composition binds tighter than the asynchronous one. Indeed, given the composable μSTSs $\Sigma_i = (S_i, \hat{s}_i, V_i, \tau_i, \circ\!\!\rightarrow_{\Sigma_i}), i = \overline{1,n}$, we can map the state space of $|_{i=1}^{n}\Sigma_i$ onto the state space of $\|_{i=1}^{n}\Sigma_i$:

$$\iota : RSS(|_{i=1}^{n}\Sigma_i) \hookrightarrow RSS(\|_{i=1}^{n}\Sigma_i)$$

by mapping for each communication channel c the state of $SFIFO(c,\tau)$ onto the state of $AFIFO(c)$ using: $c_0 \mapsto \epsilon$, $c_1 \mapsto \epsilon$, and $\forall x \in \mathcal{D}_c : c_x \mapsto x$. Similarly, we can define for any $s_i \in S_i, i = \overline{1,n}$ the injective "inclusion morphism" that maps traces of the synchronous product into traces of the asynchronous product:

$$\iota : Traces_{|_{i=1}^{n}\Sigma_i}(s) \hookrightarrow Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$$

defined inductively by $\iota(\epsilon) = \epsilon$, by $\iota(\varphi_1\varphi_2) = \iota(\varphi_1)\iota(\varphi_2)$, and (for labels) by:

$$\iota(l) = \begin{cases} l\,|_{\bigcup_{i=1}^{n} V_i \cup \{\tau_i | i=\overline{1,n}\}}, \text{ if } l \neq <\tau> \\ <\tau_1, \ldots, \tau_n>, \text{ if } l = <\tau> \end{cases}$$

where τ is the base clock of the synchronous composition. With these notations we have:

$$s \overset{\varphi}{\circ\!\!\Longrightarrow}_{|_{i=1}^{n}\Sigma_i} s' \;\Rightarrow\; \iota(s) \overset{\iota(\varphi)}{\circ\!\!\Longrightarrow}_{\|_{i=1}^{n}\Sigma_i} \iota(s')$$

4.2.8 Projection operators. Traces of a GALS system

The operator $\pi_i^{\sigma}()$ projects a state or transition label of the synchronous product $|_{i=1}^{n}\Sigma_i$ onto the corresponding state or transition label of Σ_i. Similarly, $\pi_i^{\alpha}()$ projects states and transitions of the asynchronous product $\|_{i=1}^{n}\Sigma_i$ onto states and transitions of Σ_i. The definition of $\pi_i^{\sigma}()$ and $\pi_i^{\alpha}()$ is trivial, with the exception of $\pi_i^{\sigma}()$ over transition labels, which involves the renaming of the common clock τ to the local clock τ_i.

Note that, while not constrained by global clock synchronization, the traces of $\|_{i=1}^{n}\Sigma_i$ still satisfy a FIFO consistency property that requires that a value is read from a channel only after being emitted. The following definition formalizes this for traces starting with void channels (from the initial state). Intuitively, we require that in any trace of the composed system the sequence of values read from a channel is a prefix of the sequence of values that are written. Moreover, we require that a write operation occurs before the corresponding read operation.

Definition 4.2.9 (FIFO consistency) *Let $\Sigma_i = (S_i, \hat{s}_i, V_i, \tau_i, \circ\!\!\rightarrow_i)$, $1 \leq i \leq n$ be composable μSTSs, and let φ be some trace in $Traces(\bigcup_{i=1}^{n}(V_i \cup \{\tau_i\}))$. We say that φ is FIFO consistent if for each channel c shared between two components we have $\delta(\varphi\,|_{\{?c\}}) \preceq \delta(\varphi\,|_{\{!c\}})$ and the rank of $\delta(\varphi\,|_{\{?c\}})[j]$ in φ is greater than the rank of $\delta(\varphi\,|_{\{!c\}})[j]$ in φ for all $j \leq length(\delta(\varphi\,|_{\{?c\}}))$.*

We can now characterize the traces of $\|_{i=1}^{n}\Sigma_i$:

Lemma 4.2.10 (GALS traces) *Let $\Sigma_1, \ldots, \Sigma_n$ be composable μSTSs and let $s \in RSS(|_{i=1}^{n}\Sigma_i)$ be a synchronizing state. Then, $\varphi \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ if and only if $\forall i : \pi_i^{\alpha}(\varphi) \in Traces_{\Sigma_i}(\pi_i^{\sigma}(s))$ and φ is FIFO consistent.*

Proof: The direct implication is obvious according to the definitions of $\otimes$ and $\pi_i^\alpha()$. Conversely, consider φ a consistent trace such that $\forall i : \varphi_i = \pi_i^\alpha(\varphi) \in Traces_{\Sigma_i}(\pi_i^\sigma(s))$. Then, the consistency of φ allows us to prove, by induction over $length(\varphi)$, that the interleaving of the φ_i's into φ is possible under the composition contraints imposed by the asynchronous FIFOs that take part in $\|_{i=1}^{n} \Sigma_i$. This implies $\varphi \in Traces_{\|_{i=1}^{n} \Sigma_i}(\iota(s))$. □

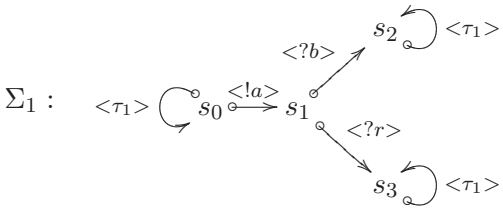
As a corollary, if we are given $\varphi_i \in Traces_{\Sigma_i}(\pi_i^\sigma(s))$, $1 \leq i \leq n$, and if we can order their non-clock transitions in a FIFO-consistent way, then there exists $\varphi \in Traces_{\|_{i=1}^{n} \Sigma_i}(\iota(s))$ such that $\pi_i^\alpha(\varphi)$ and φ_i are identical upto void transitions for all $1 \leq i \leq n$.

4.3 Modelling and correctness of GALS implementations

This section starts by illustrating our definitions with a number of small, but intuitive examples. Based on this intuition, we define in section 4.3.2 the formal correctness criterion. Section 4.3.3 explains why our model is useful in solving the GALS implementation problem.

4.3.1 Examples

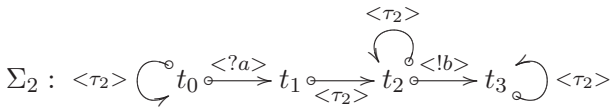
The following μSTS represents a system that emits a message on channel a and then awaits for one message from either channel b or r (*e.g.* for whichever comes first). Data is uninterpreted (not important), therefore not represented. The clock of the system is τ_1, and we shall assume that the directed variable set of Σ_1 is $\{!a, ?b, ?r\}$:



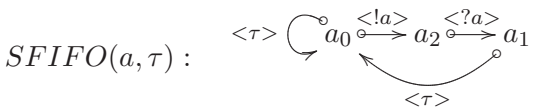
In a more classical macrostep framework, like that of [143], this system would be represented by:

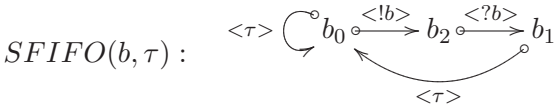
The correspondence between the microstep and macrostep representations of a system is straightforward: The states of the macrostep system are the synchronizing states of the microstep one. The macrostep transitions correspond to full reactions connecting synchronizing states (after forgetting the direction of the signals and the causality between successive labels).

We compose Σ_1 with the μSTS Σ_2, which has the clock τ_2 and the directed variable set $\{?a, !b\}$:

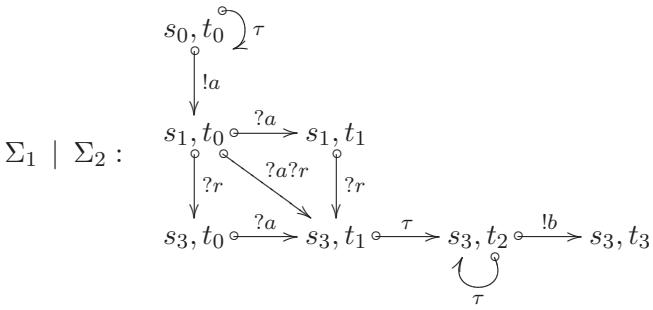


The synchronous composition $\Sigma_1 \mid \Sigma_2$ is done using two synchronous FIFOs, corresponding to the variables/channels a and b:





In this example, data is uninterpreted, only write/read causality and clock synchronization is considered. The composed synchronous system is (we simplified for space reasons the label notations, as explained in section 4.2.1):

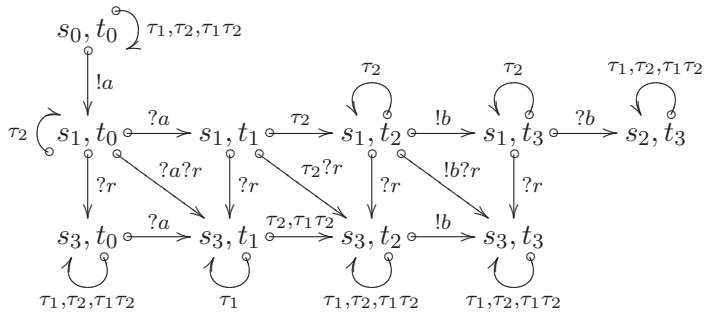


Note that we simplified the notation by not representing the state of the two FIFOs (the initial state having void FIFOs, the status of the FIFOs is fully determined in each state). However, note that the composed system is blocked in state (s_3, t_3) because $SFIFO(b, \tau)$ cannot take a clock transition (data has been written on it, but not read). The system $\Sigma_1 \mid \Sigma_2$ is blocking, thus incorrect.

The asynchronous composition $\Sigma_1 \parallel \Sigma_2$ is done using the two asynchronous FIFOs, figured below:

Recall that in the general case the asynchronous FIFO models are infinite. However, Σ_1 and Σ_2 can emit at most one message on any of the two channels, so our choice does not affect the result of the composition:

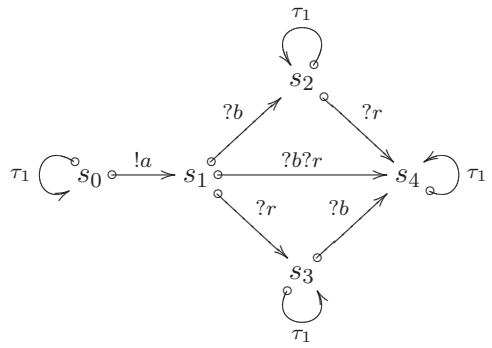
$\Sigma_1 \parallel \Sigma_2$:



It is essential to note that $\Sigma_1 \parallel \Sigma_2$ has traces, like $< !a >< ?a >< \tau_2 >< !b >< ?b >$, that are not asynchronously equivalent to any of the synchronous traces of $\Sigma_1 \mid \Sigma_2$. Such traces are not covered by the verification done on the synchronous model, meaning that the GALS implementation does not preserve the semantics of the specification.

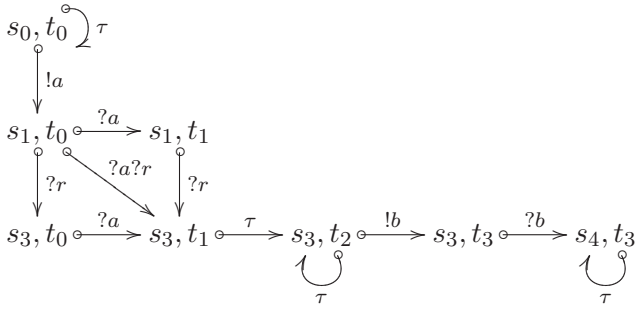
It is also important to note that requiring a one-to-one correspondence between synchronous and asynchronous traces is not a good idea, because for large classes of systems it can be highly inefficient. Consider, for instance, the following system:

Σ_3 :

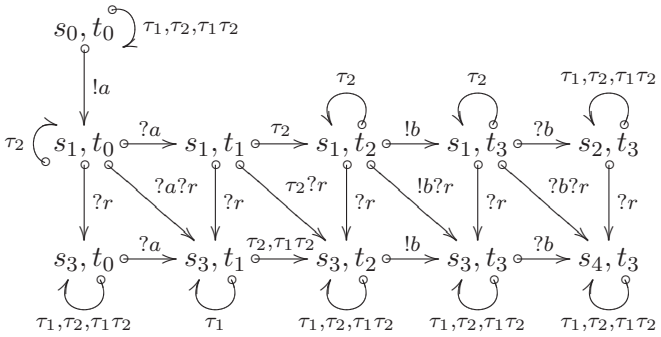


and its synchronous and asynchronous composition with Σ_2:

$\Sigma_3 \mid \Sigma_2$:



$\Sigma_3 \parallel \Sigma_2$:



As expected, the synchronous composition binds tighter than the asynchronous one, but for any trace of $\Sigma_3 \parallel \Sigma_2$ going from (s_0, t_0) to (s_4, t_3) we can find an asynchronously equivalent trace in $\Sigma_3 \mid \Sigma_2$. Such a GALS implementation is obviously correct, because it does not introduce new behaviors. Exploiting the concurrency between different computations (as we do here) to allow the systems to evolve at different rates is a desirable feature because it minimizes communication and consumption. The difference between Σ_1 and Σ_3 is that in Σ_3 the transitions $< ?b >$ and $< ?r >$ are concurrent in state s_1, while in Σ_1 there is a non-deterministic choice between them (meaning that if messages come on both channels, only one will be read, in an unpredictable fashion).

4.3.2 Formal correctness criterion

We already presented, in section 4.1.1, the intuition covering the notion of correctness of a GALS implementation with respect to its microstep synchronous specification. We give here the corresponding formal correctness criterion:

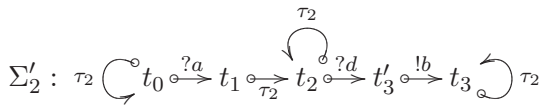
Criterion 1 (correct desynchronization) *Let $\Sigma_i, i = \overline{1,n}$ be composable μSTSs. Then, we shall say that the GALS implementation $\|_{i=1}^{n} \Sigma_i$ is correct* w.r.t. *the synchronous specification $|_{i=1}^{n} \Sigma_i$ if for all synchronizing state s of $|_{i=1}^{n} \Sigma_i$ and for all trace $\varphi \in Traces_{\|_{i=1}^{n} \Sigma_i}(\iota(s))$ there exist $\tilde{\varphi} \in Traces_{\|_{i=1}^{n} \Sigma_i}(\iota(s))$ and $\overline{\varphi} \in Traces_{|_{i=1}^{n} \Sigma_i}(s)$ such that $\varphi \preceq \tilde{\varphi}$ and $\tilde{\varphi} \sim \iota(\overline{\varphi})$.*

In other words, the GALS implementation is correct if any of its traces can be completed with a finite number of transitions to a trace that is asynchronously equivalent to a complete synchronous trace.

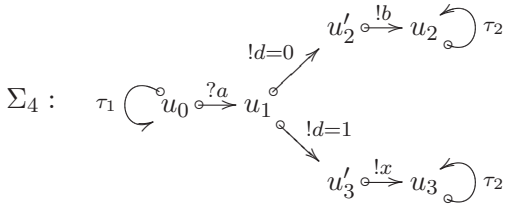
Our criterion is akin to previous correctness criteria [23, 143] defined in a macrostep setting. Most important, criterion 1 allows us to exploit (like in $\Sigma_3 \parallel \Sigma_2$) the concurrency of the synchronous specification to support GALS implementations that are weakly synchronized, yet correct. Important differences exist, though, as our criterion is formulated in a micro-step operational framework that simplifies, as we shall see in section 4.4.4, the definition of sufficient conditions for correctness.

As explained in the introduction, our purpose is now to find sufficient conditions for correctness (in the formal sense of criterion 1) that cover large classes of implementations. We do not

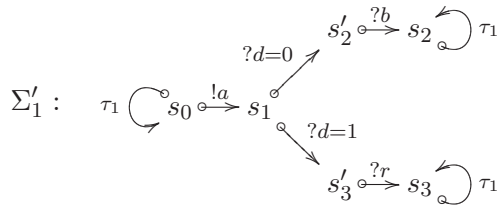
cover here the synthesis problem of transforming given systems to satisfy the correctness criterion. However, we use two examples to give the intuition of future synthesis techniques: First, to correct the composition of Σ_1 with Σ_2, we can simply prevent Σ_2 from firing the transition labeled $< !b >$ by guarding it with a condition that is never fulfilled:



More interesting is the case where we compose Σ_1 with a process Σ_4 that non-deterministically chooses between emitting b or doing something else. In this case, the solution is to signal the non-deterministic choice to Σ_1, so that it can adapt its behavior:



Here, we assumed that the non-deterministic choice between $< !b >$ and $< !x >$ is an essential feature of the specification Σ_4, which must be preserved. To make the composition correct we need to make this choice visible from its asynchronous environment, under the form of a choice over the value of a new channel, named d. Then, we can modify Σ_1 into Σ_1', which uses this signal to decide which message to wait for.



4.3.3 Modeling issues

The I/O transitions systems can be viewed either as microstep specifications, or as asynchronous implementation models. A sub-class of I/O transition systems satisfy the synchronous hypothesis – they have a single clock variable, which determines clock transitions, and no variable is assigned twice between successive clock transitions. Thus, they can be seen as microstep synchronous specifications. The only hypothesis that departs from the classical synchronous model is stuttering-invariance. However, we see stuttering-invariance as a prerequisite for the efficient multi-rate GALS deployment.

If we compare our model to macro-step models like those of [23, 143], every macrostep specification (automaton) has (at least) a microstep implementation. Like many macro-step models, our formalism does not explicitly represent the reaction to signal absence. This does not influence the expressivity of the model, as reaction to signal absence can be represented using non-deterministic choice. The composition through point-to-point links is not an essential restriction, as it is easy to define FIFO models that cover multicast.

The synchronous and asynchronous composition operators reflect the assumption that an emitted signal must not be left unread by the receiver. This hypothesis reflects in an operational fashion the rendez-vous-like synchronized product composition from macro-step formalisms.

Composing the μSTSs $\Sigma_i, i = \overline{1,n}$ using the $\|$ operator intuitively corresponds to implementing $|_{i=1}^{n}\Sigma_i$ as a GALS system where all the communication lines have been replaced with asynchronous FIFOs. The components are still clocked, but individual clocks are independent, and the components are only synchronized by the FIFO causality rules. In the GALS implementation the clock of one component can be triggered concurrently with another clock or an assignment of another component. The GALS implementation can function in a multi-rate fashion, as no constraint relates the occurrence of clock transition in different components.

Compared to classical macro-step approaches, our model brings a level of detail which is essential in deciding the correctness of actual implementations. Composing Σ_5 and Σ_6 results in a blocking system:

$$\Sigma_5: \quad \tau_1 \circlearrowleft \; 0 \xrightarrow{!a} 1 \xrightarrow{?b} 2 \xrightarrow{!c} 3 \circlearrowright \tau_1$$

$$\Sigma_6: \quad \tau_2 \circlearrowleft \; 0 \xrightarrow{?c} 1 \xrightarrow{!b} 2 \circlearrowright \tau_2$$

However, this problem cannot be observed in macro-step settings, where the system does not block and can even fire the transition of label abc. Indeed, the microstep model is better suited for analysis akin to causality checks performed in synchronous languages like Esterel. In fact, we shall see in section 4.4.4, that non-blocking correctness and semantics preservation are truly related.

4.4 Correct desynchronization criteria

Following the goal fixed in the introduction, we now define criteria that characterize a large class of synchronous components for which small, simple wrappers produce deterministic, efficient, and semantics-preserving GALS implementations.

4.4.1 Microstep weak endochrony

Microstep weak endochrony (or, simply, weak endochrony) is the property guaranteeing that a given synchronous component (μSTS) knows how to read its inputs, so that no asynchronous wrapper is needed. Weak endochrony requires that all internal choice of the component is visible as a choice over the value (and not presence/absence status) of a directed variable (either input or

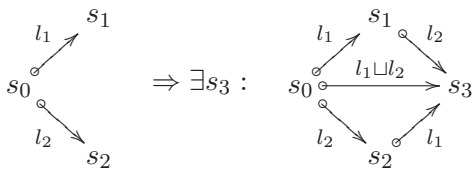
output). Thus, the behavior of the system becomes predictable in *any asynchronous environment*, because choices can be observed.

With this requirement, the implementation space delimited by weak endochrony is nonetheless very large: Concurrent behaviors are not affected by the previous rule, so that independent system parts can evolve at different speeds. Weak endochrony does not require I/O determinism. Instead, a weakly endochronous component must inform the environment about non-deterministic decisions (the variable used to do so behaves like an oracle that is visible from outside).

Definition 4.4.1 (weak endochrony) *We say that the μSTS $\Sigma = (S, \hat{s}, V, \tau, \circ\!\!\rightarrow)$ is weakly endochronous if it satisfies the following axioms:*

μWE1 (determinism): $s \overset{l}{\circ\!\!\rightarrow} s_i, i = 1,2 \Rightarrow s_1 = s_2$ *(from now on, we shall denote with $s.\varphi$ the unique state of Σ having the property $s \overset{\varphi}{\circ\!\!\rightarrow} s.l$, and the notation is extended to traces).*

μWE2 (independence): *if the labels l_1 and l_2 are disjoint and if $l_1, l_2 \neq \tau$, then:*



μWE3 (clock properties): *assume that $s_0 \overset{<\tau>}{\circ\!\!\rightarrow} s_1$ and $\varphi \in Traces_\Sigma(s_0)$ with $\tau \notin supp(\varphi)$. Then:*

1. $\varphi \in Traces_\Sigma(s_1)$
2. *if* $\varphi < \tau > \in Traces_\Sigma(s_0)$, *then* $\varphi < \tau > \in Traces_\Sigma(s_1)$ *and* $s_0.\varphi < \tau > = s_1.\varphi < \tau >$
3. *if* $\varphi\psi < \tau > \in Traces_\Sigma(s_1)$, *then there exists* $\psi' \leq \psi$ *such that* $\varphi\psi' < \tau > \in Traces_\Sigma(s_0)$.
4. *if* $\varphi < \tau >, \theta < \tau > \in Traces_\Sigma(s_0)$ *and* $\varphi \bowtie \theta$, *then* $\varphi(\theta \setminus \varphi) < \tau > \in Traces_\Sigma(s_0)$

μWE4 (choice): *if* $\varphi_i < v = x_i > \in Traces_\Sigma(s)$, $i = 1,2$ *and* $\varphi_1 \bowtie \varphi_2$, *then* $\varphi_1 < v = x_2 > \in Traces_\Sigma(s)$.

Similar in intuition and in function to its macrostep counterpart [143], weak endochrony is nevertheless specific to our more concrete causal, microstep framework. Thus, while choice can only occur at the level of atomic variable assignments, concurrency (more precisely confluence) must also deal with full reactions and clock transitions (through axioms μWE2 and μWE3, and the consequences of lemma 4.4.3). Axiom μWE4 insures that a choice between two concurrent execution paths does not hide a “real” choice between non-concurrent assignments.

Lemma 4.4.2 (independence) *Let $\Sigma = (S, \hat{s}, V, \tau, \circ\!\!\rightarrow)$ be a weakly endochronous μSTS, let $s \in S$, and let $\varphi_1, \varphi_2 \in Traces_\Sigma(s)$ with $supp(\varphi_1) \cap supp(\varphi_2) \subseteq \{\tau\}$. Then:*

1. *If $\tau \notin supp(\varphi_i), i = 1, 2$, then $s.\varphi_1\varphi_2$ and $s.\varphi_2\varphi_1$ are defined and equal.*
2. *If φ_i complete, $i = 1, 2$, then $s.\varphi_1\varphi_2$ and $s.\varphi_2\varphi_1$ are defined and equal.*

Proof:
part 1: From $supp(\varphi_1) \cap supp(\varphi_2) \subseteq \{\tau\}$ and $\tau \notin supp(\varphi_i), i = 1, 2$, we obtain $supp(\varphi_1) \cap supp(\varphi_2) = \emptyset$. Then the labels $\varphi_1[k]$ and $\varphi_2[l]$ are disjoint, for all k and l. Based on this remark, the result is easily obtained by induction over $length(\varphi_1) + length(\varphi_2)$, the induction step using axiom μWE2.
part 2: We shall give here the proof for the case where φ_1 and φ_2 comprise each one step. This will prove that independent steps commute, and this result can then be easily applied to prove that any two complete traces commute. Assume then $\varphi_1 = Step_0(\varphi_1) < \tau >$ and $\varphi_2 = Step_0(\varphi_2) < \tau >$. Let $s' = s.Step_0(\varphi_1)$.

According to the first part of this lemma, $s.Step_0(\varphi_1)Step_0(\varphi_2)$ and $s.Step_0(\varphi_2)Step_0(\varphi_1)$ exist and are equal.

By applying axiom μWE3.4, $s.Step_0(\varphi_1)Step_0(\varphi_2) < \tau >$ exists. Then, by applying axiom μWE3.2 in s', we obtain that $s'.< \tau > Step_0(\varphi_2) < \tau >$ exists and is equal to $s'.Step_0(\varphi_2) < \tau >$. By using the definition of s', this implies

$$s.Step_0(\varphi_1) < \tau > Step_0(\varphi_2) < \tau > = s.Step_0(\varphi_1)Step_0(\varphi_2) < \tau >$$

Similarly:

$$s.Step_0(\varphi_2) < \tau > Step_0(\varphi_1) < \tau > = s.Step_0(\varphi_2)Step_0(\varphi_1) < \tau >$$

Given that the second terms of the two equalities are equal, the proof is completed. □

Lemma 4.4.3 (confluence) *Let $\Sigma = (S, \hat{s}, V, \tau, \circ\!\!\rightarrow)$ be a weakly endochronous μSTS, let $s \in S$, and let $\varphi_i \in Traces_\Sigma(s), i = 1, 2$ such that $\varphi_1 \bowtie \varphi_2$. Then:*

1. *If $\tau \notin supp(\varphi_i), i = 1, 2$, then $s.\varphi_1(\varphi_2 \setminus \varphi_1)$ and $s.\varphi_2(\varphi_1 \setminus \varphi_2)$ are defined and equal.*
2. *If φ_i complete, $i = 1, 2$, then $s.\varphi_1(\varphi_2 \setminus \varphi_1)$ and $s.\varphi_2(\varphi_1 \setminus \varphi_2)$ are defined and equal.*

Proof:
part 1:
Case a: When each of the two traces are reduced to one label $\varphi_1 = l_1$, $\varphi_2 = l_2$. From $\varphi_1 \bowtie \varphi_2$ we have $l_1 \bowtie l_2$. Then, from axiom GCTS2, we can decompose l_2 into $l_2 \setminus l_1$ and $l_2 \sqcap l_1$. By applying axiom μWE2 to $l_2 \setminus l_1$ and l_1 in state s, we obtain that $s.l_1(l_2 \setminus l_1)$ and $s.(l_2 \sqcup l_1)$ exist and are equal. Similarly, $s.l_2(l_1 \setminus l_2)$ exists, and is equal to $s.(l_2 \sqcup l_1)$, which implies the needed result.
Case b: when only φ_2 is reduced to a single transition. We deduce that the desired result holds by induction over $length(\varphi_1)$, and by applying case (a).

Case c: the general case. Induction over $length(\varphi_2)$ allows us to prove the first point of the lemma in the general case.

part 2: We shall give here the proof for the case where φ_1 and φ_2 comprise one step each. This will prove that non-contradictory steps can be merged into confluent derivations. The general case is proved by iterating this result. Assume then $\varphi_1 = Step_0(\varphi_1) < \tau >$ and $\varphi_2 = Step_0(\varphi_2) < \tau >$. Let $s' = s.Step_0(\varphi_1)$.

Using the first point of the lemma, $s.Step_0(\varphi_1)(Step_0(\varphi_2) \setminus Step_0(\varphi_1))$ and $s.Step_0(\varphi_2)(Step_0(\varphi_1) \setminus Step_0(\varphi_2))$ exist and are equal. From axiom μWE3.4, $s.Step_0(\varphi_1)(Step_0(\varphi_2) \setminus Step_0(\varphi_1)) < \tau >$ exists, and then, by applying axiom μWE3.2 in state s' we obtain that

$$s.Step_0(\varphi_1) < \tau > (Step_0(\varphi_2) \setminus Step_0(\varphi_1)) < \tau > = s.Step_0(\varphi_1)(Step_0(\varphi_2) \setminus Step_0(\varphi_1)) < \tau >$$

Similarly,

$$s.Step_0(\varphi_2) < \tau > (Step_0(\varphi_1) \setminus Step_0(\varphi_2)) < \tau > = s.Step_0(\varphi_2)(Step_0(\varphi_1) \setminus Step_0(\varphi_2)) < \tau >$$

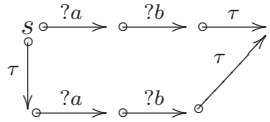
which implies $s.\varphi_1(\varphi_2 \setminus \varphi_1) = s.\varphi_2(\varphi_1 \setminus \varphi_2)$. □

Note that the proofs of lemma 4.4.2(1) and lemma 4.4.3(1) are only based on the axioms μWE3 and μWE4.

Lemma 4.4.4 (completion) *Let $\Sigma = (S, \hat{s}, V, \tau, T)$ be a weakly endochronous μSTS, and consider a state $s \in S$ and two traces $\varphi_1, \varphi_2 \in Traces_\Sigma(s)$. If φ_2 is complete and $\varphi_1 \leq \varphi_2$, then there exists φ_3 complete such that $s.\varphi_1\varphi_3 = s.\varphi_2$ and $\varphi_1\varphi_3 \sim \varphi_2$. In addition, if φ_1 is complete then we can take $\varphi_3 = \varphi_2 \setminus \varphi_1$.*

Proof: The case where φ_1 is complete is a mere corollary of lemma 4.4.3(2). The case where $\tau \notin \varphi_1$ is proved using axiom μWE3.4. The general case is a simple combination of the two previous cases. □

Note that we do not require confluence for arbitrary (incomplete) traces. The intuition behind this restriction is that the atomicity of reactions must be preserved, and therefore the clock transitions cannot follow the simple commutation rule of axiom μWE2. In the following weakly endochronous μSTS, for instance (initial state s), $s.< ?a >$ and $s.< \tau >< ?a >$ are different:



Also note how lemma 4.4.2 gives the classical independence (full commutation) results, for the case where $supp(\varphi_1)$ and $supp(\varphi_2)$ share no directed variable. However, the finer microstep notion allows us to consider systems like Σ_3 where the classical macrostep independence does

not apply (in state s_0, the macrostep transitions ab and ar do not commute, yet the system is I/O deterministic).

The confluence properties of an endochronous system are even stronger, as stated by the following:

Theorem 4.4.5 (determinism) *Let $\Sigma = (S, \hat{s}, V, \tau, T)$ be a weakly endochronous μSTS, $s \in S$, and let φ_1, φ_2 be traces of $Traces_\Sigma(s)$ such that $\varphi_1 \sim \varphi_2$. Then*

1. *if $\tau \notin supp(\varphi_i)$, $i = 1, 2$, then $s.\varphi_1 = s.\varphi_2$*
2. *if φ_1, φ_2 are complete, then $s.\varphi_1 = s.\varphi_2$.*

Proof: Point 1 is a corollary of lemma 4.4.3(1). Point 2 is a simple corollary of lemma 4.4.4. □

Note that the last lemma (point 1) tells us that we can non-ambiguously label the states reachable from a given state in one instant by the signals emitted or received to reach it.

In fact, these strong confluence properties allow us to put any trace of of a weakly endochronous system in *normal form*, in which every transition is maximal and the number of reactions minimal. The main result is:

Theorem 4.4.6 (maximal steps/normal form) *Let $\Sigma = (S, \hat{s}, V, \tau, T)$ be a weakly endochronous μSTS, $s \in S$, and $\varphi \in Traces_\Sigma(s)$, complete. Then, there exists $\overline{\varphi} \in Traces_\Sigma(s)$, complete, with $\overline{\varphi} \sim \varphi$ and such that $< Step_0(\overline{\varphi}) >$ is maximal (for label inclusion).*

Proof: Let L be the set of all labels l of non-clock transitions starting in s such that $l \leq \varphi$. Then, for all $l_1, l_2 \in L$ we have $l_1 \bowtie l_2$. By using the same reasoning as in the proof of lemma 4.4.3(1), we obtain $l_1 \sqcup l_2 \in L$, for all $l_1, l_2 \in L$. The maximal transition $\overline{\varphi}[0]$ is the union of all the labels in L.

This process can be iterated to construct maximal non-clock, non-void transitions $\overline{\varphi}[j]$, $j \geq 0$, until for a given $j_0 + 1$ no such transition can be built. The process is finite, for each variable $v \in V$ can be assigned at most once by $\overline{\varphi}_1 = \overline{\varphi}[0 \ldots j_0]$.

From the maximality of $\overline{\varphi}_1$ and from lemma 4.4.4, we deduce that $\overline{\varphi} = \overline{\varphi}_1 < \tau > (\varphi \setminus \overline{\varphi}_1)$ is a trace of $Traces_\Sigma(s)$ with $\varphi \sim \overline{\varphi}$. The maximality of $< Step_0(\overline{\varphi}) >$ is easily proved by *reduction ad absurdum*, which completes our proof. □

We conclude the presentation of weak endochrony by stating the very important compositionality result that allows to incrementally build complex weakly endochronous systems.

Theorem 4.4.7 (compositionality) *Let $\Sigma_i, i = \overline{1, n}$ be composable weakly endochronous μSTSs. Then, $|_{i=1}^{n} \Sigma_i$ is weakly endochronous.*

Proof: Direct application of the previous results, by taking into account the definition of the synchronous composition. □

Weak endochrony is illustrated by the μSTSs Σ_2, Σ_3, and $\Sigma_3 \mid \Sigma_2$ of section 4.3.1, and by all the examples of the sections 4.3.2 and 4.3.3. The μSTS Σ_1 is not weakly endochronous because the non-deterministic choice in state s_1 makes Σ_1 unpredictable, so that other components, like

Σ_4, cannot adjust their behavior to preserve the synchronous semantics. The transformation of Σ_1 in Σ_1' illustrates the type of instrumentation required to transform a general μSTS into a weakly endochronous one.

4.4.2 Comparison with macrostep Weak Endochrony

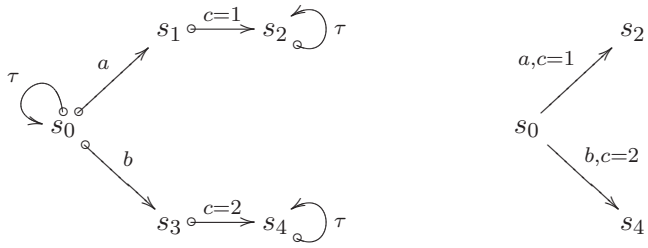
The fundamental difference between macrostep Weak Endochrony and this microstep version is that the former can make decisions involving the value of several signals received during a reaction. In our microstep framework, each decision is based on the value of only one input signal.

It is easy to associate a macrostep synchronous representation – an LSTS in the spirit of [143] – to any STS. More exactly, given $\Sigma = (S, \hat{s}, V, \tau, T)$, we associate the LSTS $[\Sigma] = (S, \hat{s}, T')$, where:

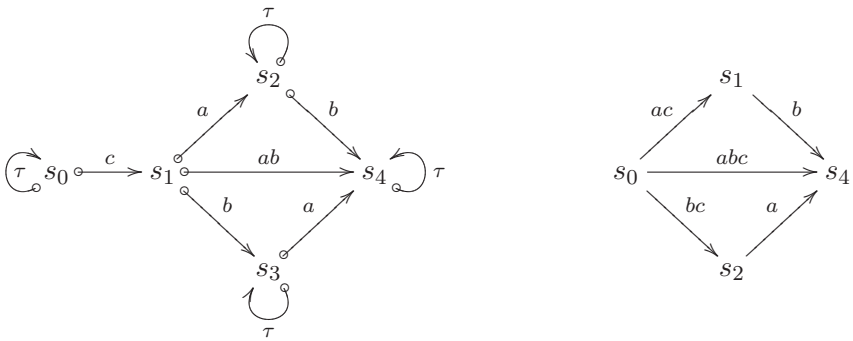
$$s \xrightarrow[[\Sigma]]{l} s' \Leftrightarrow \exists \varphi : \begin{cases} s \overset{\varphi}{\Longrightarrow} s' \\ \varphi = Step_0(\varphi) < \tau > \\ l =< Step_0(\varphi) > \end{cases}$$

In other words, the macrostep version considers only transitions from a synchronizing state to another synchronizing state, the other states being invisible at this level of abstraction.

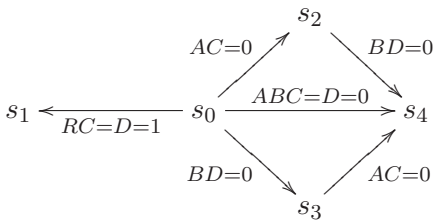
Unfortunately, the relation between microstep and macrostep weak endochrony is not simple. Given a μSTS Σ, such as the one below (at left) the fact that $[\Sigma]$ (below, at right) is weakly endochronous does not imply that Σ is microstep weakly endochronous. In our case, it is not.



At the same time, a microstep weakly endochronous system Σ is not necessarily macrostep weakly endochronous, as the following example shows:



One extra problem is that there exist macrostep weakly endochronous systems that have no microstep weakly endochronous encoding. One of them is the following:



It appears that for each macrostep weakly endochronous system there exists a microstep one over the same variables and with the same asycnhronous traces. As explained, earlier, this is due to the fact that macrostep weak endochrony can rely on tests involving several variables at a time, which is impossible in our microstep framework.

4.4.3 Comparison with related models

Weak endochrony belongs to a family of properties whose goal is to preserve concurrency while ensuring the correct operation of a system in an untimed asynchronous environment. We refer here to the work of Keller [116]. In this paper, Keller shows that 3 properties – *determinism*, *commutativity*, and *persistence* – ensure global confluence in a very general form of asynchronous transition system. The determinism requirement is quite common, but commutativity and persistence are the key point of the approach. They roughly correspond to axioms μWE2 and μWE3, ensuring that independent labels in a given state are concurrent and non-interfering, and remain available while not taken.

Weak endochrony follows the same principles, but in a much more specific setting:

– Our communication lines can transmit data, not mere arrival notifications. This allows us to refine our correctness criteria to take into account *choice* in the system-environment synchronization protocol (axiom μWE4).

– Weak endochrony deals with *synchronous systems*. The most natural way of ensuring persistency of transitions that are not taken is to ensure an intra-instance persistency, concerning microstep transitions (μWE2), and a macrostep persistency, covering full reactions (μWE3 states that clock transitions cannot disable other transitions). It is interesting here to recall that the macrostep weak endochrony of [143] needed only a macrostep persistency property. But here we need the microstep aspect, as well. Macrostep persistency can be seen as covering non-interfering transactions instead of elementary communications.

– Weak endochrony defines a normal, most compact form for system behaviors, something not provided by the general confluence results of Keller. This allows reasoning on convergence speed. For instance, if two synchronous reactions starting in a state are non-contradictory, then convergence between them can be attained in at most one reaction.

– Finally, our systems are input/output systems, which are predictable, but not deterministic as such (they are deterministic only if we forget about the direction of signals).

These supplementary aspects determine the complexity of the theory and the difficulty of the proofs. A Major difference with Keller's work is that he is interested in the confluence of a single system (which corresponds, in our setting, to lemma 4.4.4). Our work aims at finding conditions under which the semantics of a system of components does not change when we replace a strongly synchronized composition mechanism with a purely asynchronous one.

The work of Keller provides the link with two approaches used in asynchronous circuit design: speed independence and delay insensitivity. Speed independence [136, 115, 116] (which usually implies the hypothesis of semimodularity) ensures that the behavior of a circuit does not depend on the speed of its basic computing elements. Delay insensitivity [173, 132] ensures that the behavior of a circuit does not depend on the delays of its internal or external communication lines. These two properties are important because they support the definition of circuits whose functionality remains unchanged when the fabrication process changes.

Like weak endochrony, speed independence and delay insensitivity support specializations of Keller's fundamental theorem (as noted, for instance, in [116]), but the hypothesis on the systems and communication lines are different from those of weak endochrony.

A second important difference with Keller's work is that our results are based on the assumption that an underlying communication infrastructure provides a lossless message-passing mechanism. Under this assumption, weak endochrony implies a very permissive persistency property. By comparison, speed independence and delay insensitivity ensure, among other things, that signals are not lost (in a sense, they cover at the same time the correctness of the message-passing protocol under given hypothesis [4], and the persistency property).

Wires in speed independent or delay insensitive circuits can only transmit events, not values: An event consist in the wire changing its value from 0 to 1, or from 1 to 0. Thus, value choices (as found in weakly endochronous systems) cannot be directly expressed in Keller's formalism, as the only possible choices are among different events, occurring on different wires. Microstep weak endochrony is not meant to express such choices, which depend on temporal assumptions on the environment (no input is produced by the environment until the system is ready to read it). With an appropriate introduction of clock transitions, weak endochrony should be able to directly represent delay-insensitive systems with no choice (*cf.* axiom R'_3 in [173]).

More work is needed to understand the precise relation between weak endochrony, on one side, and speed independence and delay insensitivity, on the other, particularly by defining a notion of *circuit realization* for weakly endochronous systems, along the lines of [115].

Our work bears some relations with that of Yun and Dill on burst-mode circuits [179]. Their goal is to deal with multiple-signal interactions, instead of single signal events. The approach is oriented towards circuit synthesis, and strict operation conditions are required, which basically exclude true concurrency.

4. such as the fact that forks are isochronic

4.4.4 Correctness results

Weak endochrony is compositional. However weak endochrony of all components does not guarantee the correctness (non-blocking) of the global synchronous specification, nor the correctness (semantics preservation) of the GALS implementation model. This can be easily checked on the systems formed by composing Σ_1' and Σ_2 — defined in Sections 4.3.1 and 4.3.2.

The most important result of this chapter is the following theorem, which states that the correctness of the synchronous composition implies the correctness of the GALS implementation.

In fact, the strong confluence and determinism properties of the weakly endochronous systems will allow to prove an even stronger result, that also insures state determinism in addition to observational behavior equivalence:

Criterion 2 (correct desynchronization for weakly endochronous systems) *Let $\Sigma_i, i = \overline{1,n}$ be composable μSTSs. Then, we shall say that the GALS implementation $\|_{i=1}^{n}\Sigma_i$ is correct* w.r.t. *the synchronous specification $|_{i=1}^{n}\Sigma_i$ if for all synchronizing state s of $|_{i=1}^{n}\Sigma_i$ and for all trace $\varphi \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ there exist $\overline{\varphi} \in Traces_{|_{i=1}^{n}\Sigma_i}(s)$ complete and $\tilde{\varphi} \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ such that $\varphi \preceq \tilde{\varphi}$, $\tilde{\varphi} \sim \iota(\overline{\varphi})$, and $\iota(s).\tilde{\varphi} = \iota(s).\iota(\overline{\varphi})$.*

Quite interestingly, Criterion 2 implies Criterion 1 (the former has extra requirements).

Theorem 4.4.8 (correctness) *Let $\Sigma_i, i = \overline{1,n}$ be composable weakly endochronous μSTSs. If $|_{i=1}^{n}\Sigma_i$ is non-blocking, then $\|_{i=1}^{n}\Sigma_i$ is correct* w.r.t. *$|_{i=1}^{n}\Sigma_i$ in the sense of criterion 2.*

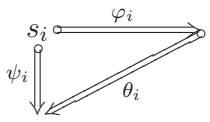
Two technical lemmas are needed to prove the theorem.

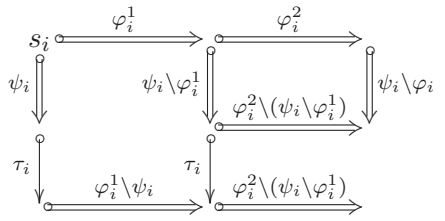
Lemma 4.4.9 (completion, GALS) *Let $\Sigma_1, \ldots, \Sigma_n$ be composable weakly endochronous μSTSs, let s be a synchronizing state of $|_{i=1}^{n}\Sigma_i$, and let $\psi \in Traces_{|_{i=1}^{n}\Sigma_i}(s)$, complete, and $\varphi \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s))$ such that $\varphi \leq \iota(\psi)$. Then, there exists $\theta \in Traces_{\|_{i=1}^{n}\Sigma_i}(\iota(s).\varphi)$ such that $\varphi\theta \sim \iota(\psi)$ and $\iota(s).\varphi\theta = \iota(s).\iota(\psi)$*

Proof: We can assume, without losing generality, that all the labels of ψ are atomic (assign exactly one variable). By projecting φ and ψ on the components Σ_i, we obtain:

$$\pi_i^{\alpha}(\varphi), \pi_i^{\sigma}(\psi) \in Traces_{\Sigma_i}(\pi_i^{\sigma}(s)) \text{ with } \begin{cases} \pi_i^{\sigma}(\psi) \text{ complete} \\ \pi_i^{\alpha}(\varphi) \leq \pi_i^{\sigma}(\psi) \end{cases}$$

We denote with $\varphi_i = \pi_i^{\alpha}(\varphi)$, $\psi_i = \pi_i^{\sigma}(\psi)$, $s_i = \pi_i^{\sigma}(s)$. By applying lemma 4.4.4(2), we find a complete trace θ_i such that the following holds in Σ_i:





where $\begin{cases} (\varphi_i^1 \setminus \psi_i)(\varphi_i^2 \setminus (\psi_1 \setminus \varphi_i^1)) = \varphi_i \setminus \psi_i \\ \varphi_i^1 \varphi_i^2 = \varphi_i \end{cases}$

Figure 4.2: Diagram with the transitions used in the proof of lemma 4.4.10

Now recall that the construction process used by lemma 4.4.4 (based on the constructions of lemma 4.4.3) insures that in each θ_i the atomic communication operations (non-clock labels) are ordered in the same fashion as they are in ψ_i. More precisely, let r_i be the rank in ψ_i of the non-clock label that has rank i in θ_i. Whenever $r_i > r_j$, we have $i > j$. The same relation is preserved by the projection of ψ onto ψ_i. Then, the ordering of the operations of the θ_i in ψ can be used to interleave the labels of the traces θ_i into θ, and our lemma is proved. □

Lemma 4.4.10 (technical) *Let $\Sigma_1, \ldots, \Sigma_n$ be composable weakly endochronous μSTSs, let s be a synchronizing state of $|_{i=1}^n \Sigma_i$, and let $\varphi, \psi < \tau_1 \ldots \tau_n > \in Traces_{\|_{i=1}^n \Sigma_i}(\iota(s))$ such that $\varphi \bowtie \psi$ and $\forall i : \tau_i \notin supp(\psi)$. Then:*

$$\varphi(\psi \setminus \varphi), \psi < \tau_1 \ldots \tau_n > (\varphi \setminus \psi) \in Traces_{\|_{i=1}^n \Sigma_i}(\iota(s))$$

Proof: By projecting φ and ψ on the components Σ_i, we obtain:

$$\pi_i^\alpha(\varphi), \pi_i^\alpha(\psi) < \tau_i > \in Traces_{\Sigma_i}(\pi_i^\sigma(s)) \text{ with } \begin{cases} \pi_i^\alpha(\varphi) \bowtie \pi_i^\alpha(\psi) \\ \tau_i \notin supp(\pi_i^\alpha(\psi)) \end{cases}$$

We denote with $\varphi_i = \pi_i^\alpha(\varphi)$, $\psi_i = \pi_i^\alpha(\psi)$, $s_i = \pi_i^\sigma(s)$. Also let $\varphi_i = \varphi_i^1 \varphi_i^2$ with φ_i^1 complete and $\tau_i \notin supp(\varphi_i^2)$. All these elements are pictured in fig. 4.2, which also contains the other transitions that will be constructed during this proof.

By applying lemma 4.4.3(2), $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1) < \tau_i >$ and $s_i.\psi_i < \tau_i > (\varphi_i^1 \setminus \psi_i)$ exist and are equal.

By applying lemma 4.4.3(1) in state $s_i.\varphi_i^1$ and for the traces φ_i^2 and $\psi_i \setminus \varphi_i^1$, we conclude that $s_i.\varphi_i^1\varphi_i^2(\psi_i \setminus \varphi_i)$ and $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1)(\varphi_i^2 \setminus (\psi_i \setminus \varphi_i^1))$ exist and are equal.

The existence of $s_i.\varphi_i^1\varphi_i^2(\psi_i \setminus \varphi_i)$ means that $\varphi_i(\psi_i \setminus \varphi_i) \in Traces_{\Sigma_i}(s_i)$.

The existence of $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1)(\varphi_i^2 \setminus (\psi_i \setminus \varphi_i^1))$ means that $\theta_i = \varphi_i^2 \setminus (\psi_i \setminus \varphi_i^1)$ is a trace of Σ_i starting in $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1)$. Since $< \tau_i >$ is a trace starting in the same state, from axiom μWE1

we obtain that θ_i is a trace of Σ_i starting in state $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1) < \tau_i >$, and by the identity of $s_i.\varphi_i^1(\psi_i \setminus \varphi_i^1) < \tau_i >$ and $s_i.\psi_i < \tau_i > (\varphi_i^1 \setminus \psi_i)$ we have $\psi_i < \tau_i > (\varphi_i \setminus \psi_i) \in Traces_{\Sigma_i}(s_i)$.

We proved that for all $1 \leq i \leq n$ we have $\varphi_i(\psi_i \setminus \varphi_i), \psi_i < \tau_i > (\varphi_i \setminus \psi_i) \in Traces_{\Sigma_i}(s_i)$, meaning that

$$\forall i : \pi_i^\alpha(\varphi(\psi \setminus \varphi)), \pi_i^\alpha(\psi < \tau_1 \ldots \tau_n > (\varphi \setminus \psi)) \in Traces_{\Sigma_i}(\pi_i^\sigma(s))$$

On each channel, the projection of any of $\varphi(\psi \setminus \varphi)$ or $\psi < \tau_1 \ldots \tau_n > (\varphi \setminus \psi)$ is either a prefix of the projection of φ, or a prefix of the projection of ψ, which are themselves consistent. Therefore, traces $\varphi(\psi \setminus \varphi)$ and $\psi < \tau_1 \ldots \tau_n > (\varphi \setminus \psi)$ are also consistent. According to lemma 4.2.10, this implies that $\varphi(\psi \setminus \varphi), \psi < \tau_1 \ldots \tau_n > (\varphi \setminus \psi)$ is in $Traces_{\|_{i=1}^n \Sigma_i}(\iota(s))$. □

Thanks to these two technical lemmas, theorem 4.4.8 can now be proved.

Proof of theorem 4.4.8: Let s be a synchronizing state of $|_{i=1}^n \Sigma_i$ and let $\varphi \in Traces_{\|_{i=1}^n \Sigma_i}(\iota(s))$. We prove the existence of $\overline{\varphi}$ and $\tilde{\varphi}$ by induction over $\mid \varphi \mid$ (the number of variable assignments in φ). We can assume, without losing generality, that every label of φ assigns exactly one variable, either clock or directed. The reduction to this case is straightforward.

If $\mid \varphi \mid = 0$, then $\tilde{\varphi} = \varphi$ and $\overline{\varphi} =< \tau >$ clearly satisfy the conditions of criterion 2.

Consider now φ with $\mid \varphi \mid \geq 1$. If $\varphi[0] =< \tau >$, then we have:

$$\iota(s) \circ \xrightarrow{\varphi[0]} \iota(s) \circ \xRightarrow{\varphi[1 \ldots length(\varphi)-1]}$$

The induction hypothesis can be applied on $\varphi[1 \ldots length(\varphi) - 1]$ to determine $\tilde{\varphi}$ and $\overline{\varphi}$.

From now on, we assume that $\varphi[0]$ is not a clock transition.

To prove that $\tilde{\varphi}$ and $\overline{\varphi}$ exist, we shall first construct $\psi \in Traces_{|_{i=1}^n \Sigma_i}(s)$ such that $\tau \notin supp(\psi)$, $\psi < \tau > \in Traces_{|_{i=1}^n \Sigma_i}(s)$, $\psi[0] = \varphi[0]$, and $\iota(\psi) \bowtie \varphi$.

Construction of ψ: Consider the projections of φ onto components $\pi_j^\alpha(\varphi)$, $1 \leq j \leq n$. Since $\varphi[0]$ is not the empty transition, nor a clock transition, then there exists at least an j such that $\pi_j^\alpha(\varphi)[0]$ is not a clock transition, nor a void transition. Note that $\pi_j^\alpha(\varphi)[0]$ is fireable in state $\pi_j^\sigma(s)$.

Then, we start the iterative construction of ψ by setting the iteration counter i to 0 and setting $\psi_0 = \varphi[0]$. The iteration step:

continuation test: If $\psi_i < \tau > \in Traces_{|_{i=1}^n \Sigma_i}(s)$, then we completed our construction.

construction step: If $\psi_i < \tau > \notin Traces_{|_{i=1}^n \Sigma_i}(s)$, and since $|_{i=1}^n \Sigma_i$ is non-blocking, there exists a label $< v = x >$ such that $\psi_i < v = x > \in Traces_{|_{i=1}^n \Sigma_i}(s)$. If $\psi_i < v = x > \bowtie \varphi$, then consider $\psi_{i+1} = \psi_i < v = x >$ and go to the continuation test.

If not, then $\varphi = \varphi_0 < v = y > \varphi_1$ with $x \neq y$ and $v \notin supp(\varphi_0)$. However, by applying axiom μWE4, we have $\psi_i < v = y > \in Traces_{|_{i=1}^n \Sigma_i}(s)$. By considering $\psi_{i+1} = \varphi_i < v = y >$, we also have $\psi_{i+1} \bowtie \varphi$, and we go to the continuation test.

The previous algorithm is finite, bounded by the number of variables in $|_{i=1}^n \Sigma_i$. In the end we put the last ψ_i in ψ.

Construction of $\tilde{\varphi}$ and $\overline{\varphi}$: Let $s_1 = s.\psi < \tau >$. According to lemma 4.4.10 we have

$$\varphi(\iota(\psi) \setminus \varphi), \iota(\psi) < \tau_1, \ldots, \tau_n > \in Traces_{\|_{i=1}^{n} \Sigma_i}(\iota(s))$$

Since $\varphi[0] = \iota(\psi)[0]$, we have $\mid \varphi \setminus \psi \mid < \mid \varphi \mid$. Then, we can apply the induction hypothesis in the synchronizing state s_1, and obtain $\overline{\theta} \in Traces_{|_{i=1}^{n} \Sigma_i}(s_1)$, complete, and $\tilde{\theta} \in Traces_{\|_{i=1}^{n} \Sigma_i}(\iota(s_1))$ such that:

$$\begin{cases} \varphi \setminus \iota(\psi) \preceq \tilde{\theta} \\ \iota(\overline{\theta}) \sim \tilde{\theta} \\ \iota(s_1).\iota(\overline{\theta}) = \iota(s_1).\tilde{\theta} \end{cases}$$

Let $\overline{\varphi} = \phi < \tau > \overline{\theta}$. Given that $\varphi \leq \iota(\psi) < \tau_1 \ldots \tau_n > (\varphi \setminus \iota(\psi))$ and that $\varphi \setminus \iota(\psi) \leq \iota(\overline{\theta})$, we deduce $\varphi \leq \iota(\overline{\varphi})$. Since $\overline{\varphi} \in Traces_{|_{i=1}^{n} \Sigma_i}(s)$ is complete, lemma 4.4.9 can be applied to build $\varphi' \in Traces_{\|_{i=1}^{n} \Sigma_i}(\iota(s).\varphi)$ such that $\varphi\varphi' \sim \iota(\overline{\varphi})$ and $\iota(s).\varphi\varphi' = \iota(s).\iota(\overline{\varphi})$. By considering $\tilde{\varphi} = \varphi\varphi'$, the proof is completed. □

Theorem 4.4.8 implies that for large classes of components for which simple wrappers exist, the correctness of the GALS implementation is implied by the correctness of the global synchronous specification. Thus, no extra signalization is needed to ensure semantics preservation (and no costly synthesis algorithms). The GALS implementation is correct by construction.

4.5 Conclusion. Future work

We introduced a new model for the representation of asynchronous implementations of synchronous specifications. The model covers implementations where a notion of global synchronization is preserved by means of signaling, and GALS implementations, where global synchronization is relaxed. The model takes into account computation and communication causality, and allows us to reason about semantics-preservation and absence of deadlocks in the GALS deployment of synchronous specifications. As the model captures the internal concurrency of the synchronous specification, our correctness criteria support implementations that are less constrained and more efficient than existing ones.

The results of section 4.4 suggest that our model offers a good abstraction level for reasoning about desynchronization. In particular, the level of detail is essential in revealing the intricate relation between (1) causal dependencies, concurrency and conflicts in the micro-step semantics of a synchronous specification and (2) the correctness (semantics preservation) of its GALS implementation.

4.5.1 Future work

Thanks to this new model, we are exploring the development of GALS circuits made of synchronous IPs. Our work aims at using asynchronous logic wrappers to encapsulate the components

of a modular synchronous circuit into delay insensitive components. Our model seems well-suited to analyze designs involving both synchronous and asynchronous circuit specifications. Preliminary results are presented in [74], but we are only at the beginning of our work.

We are also considering symbolic analysis techniques that would allow us to translate the theory detailed in this chapter to high-level synchronous languages like Signal or Esterel, instead of simple finite state automata. The objective is to derive efficient algorithms transforming general high-level specifications into weakly endochronous ones. Preliminary results in this direction are presented in [169].

A third research direction concerns the (still not sufficiently clear) relations between classical, macro-step synchronous models and more operational models like microstep synchronous transition systems (μSTS), or the ones covering the implementations of synchronous programming languages, especially when desynchronization is involved. For instance, it is important to understand how the notions of correct desynchronization and endochrony can be transposed into a constructive framework such as the one of Esterel [35].

Chapter 5

The Non-Standard Semantics of Hybrid Systems

Résumé : *La modélisation à l'aide de systèmes hybrides est devenue une pratique courante dans de nombreuses industries. Pour preuve, les outils de modélisation et de simulation hybride (Matlab/Simulink étant le plus connu d 'entre eux) sont utilisés par un très grand nombre d'ingénieurs. D'autres formalismes connaissent également un essort rapide, en particulier le langage Modelica, qui permet, en utilisant des systèmes d'équations algebro-différentielles, de definir des composants avec des variables dont le rôle, entrée ou sortie, n'est pas figé, mais au contraire inféré á la compilation, lorsque le composant est instancié dans un environnement connu.*

Ces formalismes permettent de modéliser avec une grande facilité des systèmes complexes, mélangeant des modèles de systèmes physiques et de logiciel. Toutefois, certaines difficultés demeurent non-résolues : la concurrence créée par la simultanéité de passages par zéro peut être difficile à gérer et est source d'ambiguités. La sémantique des parties discrètes d'un modèle hybride n'est pas toujours simple à comprendre, en particulier quand des cascades de passages par zéro se produisent. La question de la génération de code de simulation est elle même une question délicate. Ainsi, la partition d'un programme hybride en deux parties, l'une hybride (devant être exécutée par un solveur déquations différentielles) et l'autre discrète (servant à contrôler le solveur d'équations différentielles), est un problème qui n'est pas totalement résolu.

Ce chapitre détaille nos travaux portant sur la définition d'un mini-langage flôts de données hybride, reprennant les principes fondamentaux des langages flôts de données synchrones, ainsi que sa sémantique formelle. Nous avons repris une construction des nombres réels non-standard pour définir cette sémantique hybride, qui nous permet de parler avec rigueur de variations infinitésimales et d'incréments de temps infinitésimaux. Le résultat est une sémantique constructive (mais non effective) de notre mini-langage hybride, dans laquelle un comportement est une suite trans-finie de transitions représentant soit l'exécution d'un pas discret du programme, soit une évolution continue infinitésimale. Cette sémantique a l'avantage qu'elle permet de parler avec

précision des questions de causalité et de concurrence entre passages par zéro. Elle permet également de mieux comprendre les programmes Zénon, en particulier ceux qui définissent des modes glissants.

5.1 Introduction

Over the last two decades, hybrid systems modelers have become the corner stone of complex embedded system development, especially for computer controlled systems. Simulink [1] has become the de facto standard for physical system modeling and simulation. Noticeably, by building on the success of Simulink, The Mathworks was able to dominate several sectors of the market for embedded systems design. This in itself demonstrates the importance of such tools. In this chapter we focus on general modelers, aimed at modeling and simulating any type of hybrid system and we refer the reader to [60] for a more general overview of tools for hybrid systems analysis. Besides Simulink and its state-based extension Stateflow, several other hybrid systems modelers have been developed. Scicos [2] is freely available software developed by Ramine Nikoukhah at INRIA [58, 137]. Modelica [3] is a non-proprietary, object-oriented, equation based language to conveniently model complex multi-physics systems. In Modelica, equations have no pre-defined causality. Hybrid systems modelers raise a number of difficult issues:

1. Zero-crossings, which trigger mode changes, can involve a combination of complex operations whose scheduling may be delicate.
2. How discrete is the semantics of the discrete part of a hybrid systems modeler? Can we obtain a simulation engine using a purely discrete time language compiler (e.g. a synchronous language engine) managing a bare ODE solver? Note, that, quite often, discrete and continuous behaviours are not cleanly separated in hybrid systems modelers.
3. Since simulations use a single, global, solver, the choice and tuning of the integration method affects the entire system. This may lead to undesirable interactions between sub-systems that seemingly should not interact.
4. What are the consequences for the compilation of Modelica's "acausal" approach?

In this chapter we focus on the first three issues. The case of Modelica and the handling of Differential Algebraic Equations (DAE) is not covered here.

Issues raised by zero-crossings The following examples illustrate some of the subtleties of zero-crossings. In them, the resetting mechanisms involve a tuple of zero-crossings. For instance, the statement "$\mathbf{reset}\,[1,-1]\,\mathbf{every\ up}[x,-x]$" specifies that the signals x and $-x$ are monitored for

1. http://www.mathworks.com/products/simulink/
2. http://www-rocq.inria.fr/scicos/
3. http://www.modelica.org/

upward crossings of zero (from ≤ 0 to > 0), and further that the signal is rest to 1 when a zero-crossing occurs on x, and to -1 when a zero-crossing occurs on $-x$, with priority to the former if both events occur simultaneously.

$$\begin{aligned} \dot{y} = \;& 0 \;\mathbf{init}\; -1 \;\mathbf{reset}\; [1,-1] \quad \mathbf{every}\; \mathbf{up}[x,-x] \\ \dot{x} = \;& 0 \;\mathbf{init}\; -1 \;\mathbf{reset}\; [-1,1,1] \;\mathbf{every}\; \mathbf{up}[y,-y,z] \\ \dot{z} = \;& 1 \;\mathbf{init}\; -1 \end{aligned} \tag{5.1}$$

In (5.1), during the interval $[0,1)$, x and y remain steady (their slope is 0 with initial value to -1) while z increases at constant speed 1. Right after $t = 1$, z has a zero-crossing, which causes x to be reset to 1, which in turn causes a cascaded zero-crossing of x, which causes the value of y to be reset to 1, this causes a second cascaded zero-crossing on x, which then causes a second reset of the value of y to 1, and so on, unboundedly. All these cascaded zero-crossings occur while time remains blocked at $t = 1$. An attempt at illustrating this is depicted on Figure 5.1. In this drawing, $\varepsilon > 0$ is a "very small" step size, in that finitely many ε's still sum up to ≈ 0. Example (5.1) is

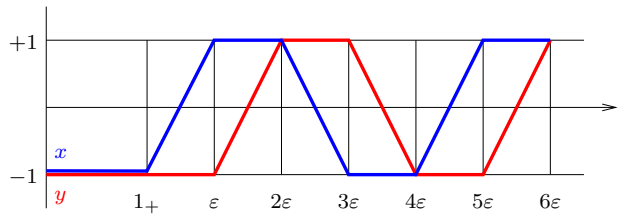


Figure 5.1: Example (5.1); ε is infinitesimal; symbol 1_+ indicates that the considered event occurs right after 1.

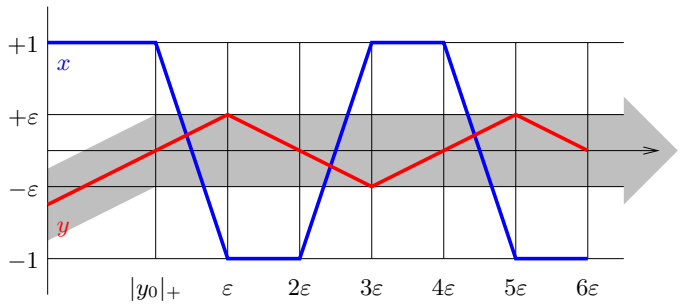


Figure 5.2: Example (5.2) with $y_0 < 0$; ε is infinitesimal; symbol $|y_0|_+$ indicates that the considered event occurs right after $t = |y_0|$.

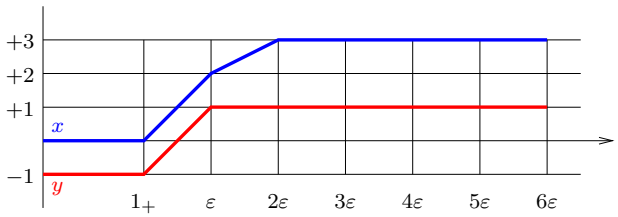


Figure 5.3: Example (5.3); z is not shown; ε is infinitesimal; symbol 1_+ indicates that the considered event occurs right after 1.

certainly pathological. In contrast,

$$\begin{aligned} \dot{x} &= 0 \ \mathbf{init} \ -\operatorname{sgn}(y_0) \ \mathbf{reset} \ [-1, 1] \ \mathbf{every} \ \mathbf{up}[y, -y] \\ \dot{y} &= x \ \mathbf{init} \ y_0 \end{aligned} \tag{5.2}$$

is the simplest case of *sliding mode control* [93]. Suppose $y_0 < 0$, and hence $x_0 > 0$. Then, y increases at constant speed until its first zero-crossing, just after time $t = |y_0|$. From then on, y chatters infinitesimally around 0 as its speed alternates between -1 and $+1$ with infinitesimal steps, as shown in Figure 5.2. This simple example captures the behaviour of systems like ABS in automobile brakes. An adequate interpretation of the behaviour of y is *averaging* over time, thus resulting in the mean dynamics $\mathbf{y}$, where:

$$\dot{\mathbf{y}} = \begin{cases} -\operatorname{sgn}(y_0), & \text{for the interval } [0, |y_0|\,) \\ 0 & \text{for } [\,|y_0|, \infty), \end{cases}$$

see the thick shaded dynamics in Figure 5.2.

For our last example, operator $\mathbf{last}\,(x)$, where x is a signal, delivers at instant t the left-limit $\lim_{s \nearrow t} x_s$:

$$\begin{aligned} \dot{x} &= 0 \ \mathbf{init} \ 0 \ \mathbf{reset} \ [\mathbf{last}\,(x) + 1, \mathbf{last}\,(x) + 2] \\ &\qquad\qquad \mathbf{every} \ \mathbf{up}[y, z] \\ \dot{z} &= 1 \ \mathbf{init} \ -1 \\ \dot{y} &= 0 \ \mathbf{init} \ -1 \ \mathbf{reset} \ [1] \ \mathbf{every} \ \mathbf{up}[z] \end{aligned} \tag{5.3}$$

Signal z has a zero-crossing right after $t = 1$, which causes y to have a cascaded zero-crossing. In Figure 5.3 we show the behaviour of x that results if we consider the cascaded zero-crossings of z and y as successive "micro-steps": x has two successive jumps, of 2 and then 1. One could, however, consider that the two zero-crossings occur simultaneously and then the zero-crossing of y preempts that of z (since y is listed first), which yields a single jump of 1 for x. Which semantics is best?

In Section 5.7 we discuss a more physical example where two balls collide and show how the three examples are simulated by Simulink in Section 5.8. These examples raise a number of issues:

- Can we propose a semantic domain for these examples?
- Can we use it
  - to identify (5.1) as pathological, but not (5.2)?
  - to decide on the semantics of (5.3)?
- More generally, can we develop a semantic domain to serve as a mathematical basis for the management of (possibly cascaded) zero-crossings?

We insist that engines of hybrid systems modelers cannot themselves perform sophisticated singular perturbation analyses involving averaging techniques [108]. We thus seek techniques based on abstract analyses that compilers can support.

Our contribution In [25] we advocated the use of *non-standard analysis* as a semantics domain for hybrid systems on the basis that it provides a semantics "as if it were step-based" but without fixing an effective step size for the solvers; hence any scheme for the solvers is supported. In this chapter we address issues 1)–3) of the introduction.

The chapter is organized as follows. Some background on non-standard analysis is provided in section 5.2. Our mathematical formalism for hybrid systems specification is introduced in section 5.3, we call it SIMPLEHYBRID. Section 5.4 is the core of the chapter; a denotational semantics is given based on non-standard analysis; we use it in a non trivial way to study examples (5.1), (5.2) and (5.3). The constructive semantics (see [36, 29] for this notion) of SIMPLEHYBRID is provided in Section 5.5; it provides a firm basis for the scheduling of actions at execution time. In Section 5.6 we provide a structuring of SIMPLEHYBRID systems showing that an execution engine can be obtained using an existing synchronous language engine that activates a generic ODE solver at particular times. Related work is analysed in section 5.9.

5.2 Non-standard analysis

Non-standard analysis was proposed by Abraham Robinson in the 1960s to allow explicit manipulations of "infinitesimals" [161, 87]. Robinson's approach is axiomatic, in that he proposes enriching the basic ZF (Zermelo-Fraenkel) framework with three more axioms.

To our surprise, the idea of using non-standard analysis for hybrid systems is indeed not new. Iwasaki et al. [111] first proposed using non-standard analysis to discuss the nature of time in hybrid systems. Bliudze and Krob [40, 39] used non-standard analysis as a mathematical support for defining a system theory for hybrid systems. The formalization they propose closely mimics that of Turing machines.

The introduction to non-standard analysis in [39] is very pleasant and we take the liberty to borrow and adapt it. This presentation was originally due to Lindstrøm, see [126]. Its interest is that it does not require any fancy axiomatic material but only makes use of the axiom of choice —

actually a weaker form of it.

The goal is to augment $\mathbb{R} \cup \{\pm\infty\}$ by adding, to each x in this set, a bunch of elements that are "infinitesimally close" to it, call ${}^\star\mathbb{R}$ the resulting set. Another requirement is that all operations and relations defined on $\mathbb{R}$ should extend to ${}^\star\mathbb{R}$. A first idea is to represent such additional numbers as convergent sequences of reals.[4] For example, the sequences $u_n = 1/n$, $v_n = 1/\sqrt{n}$, and $w_n = 1/n^2$ yield elements infinitesimally close to the real number < zero, observe that they can be ordered: $0 < w_n < u_n < v_n$. In fact, this can be made systematic as we will now explain.

5.2.1 Construction of non-standard domains

For I an arbitrary set, a *filter* $\mathcal{F}$ over I is a family of subsets of I such that:

1. the empty set does not belong to $\mathcal{F}$,
2. $P, Q \in \mathcal{F}$ implies $P \cap Q \in \mathcal{F}$, and
3. $P \in \mathcal{F}$ and $P \subset Q \subseteq I$ implies $Q \in \mathcal{F}$.

Consequently, $\mathcal{F}$ cannot contain both a set P and its complement P^c. A filter that contains at least one of the two for any subset $P \subseteq I$ is called an *ultra-filter*. At this point we recall Zorn's lemma, known to be equivalent to the axiom of choice:

Lemma 5.2.1 (Zorn's lemma) *Any partially ordered set $(X, \leq)$ such that any chain in X possesses an upper bound has a maximal element.*

It is easily seen that a filter $\mathcal{F}$ over I is an ultra-filter if and only if it is maximal with respect to set inclusion. By Zorn's lemma, any filter $\mathcal{F}$ over I can be extended to an ultra-filter over I. Now, if I is infinite, the family of sets $\mathcal{F} = \{P \subseteq I \mid P^c \text{ is finite}\}$ is a *free* filter, meaning it contains no finite set. It can thus be extended to a free ultra-filter over I:

Lemma 5.2.2 *Any infinite set has a free ultra-filter.*

Every free ultra-filter $\mathcal{F}$ over I uniquely defines, by setting $\mu(P) = 1$ if $P \in \mathcal{F}$ and otherwise 0, a finitely additive measure[5] $\mu : 2^I \mapsto \{0, 1\}$, which satisfies

$$\mu(I) = 1 \text{ and, if } P \text{ is finite, then } \mu(P) = 0.$$

Now, fix an infinite set I and a finitely additive measure μ over I as above. Let $\mathbb{X}$ be a set and consider the Cartesian product $\mathbb{X}^I = (x_i)_{i \in I}$. Say $(x_i) \approx (x_i')$ iff $\mu\{i \in I \mid x_i \neq x_i'\} = 0$. Relation $\approx$ is an equivalence relation whose equivalence classes are denoted by $[x_i]$ and we define

$${}^\star\mathbb{X} = \mathbb{X}^I / \approx \tag{5.4}$$

4. Indeed, the proposed construction bears some resemblance with the construction of $\mathbb{R}$ as the set of equivalence classes of Cauchy sequences in $\mathbb{Q}$ modulo the equivalence relation $(u_n) \approx (v_n)$ iff $\lim_{n\to\infty}(u_n - v_n) = 0$.

5. Observe that, as a consequence, μ cannot be sigma-additive (in contrast to probability measures or Radon measures) in that it is *not* true that $\mu(\bigcup_n A_n) = \sum_n \mu(A_n)$ holds for an infinite denumerable sequence A_n of pairwise disjoint subsets of $\mathbb{N}$.

$\mathbb{X}$ is naturally embedded into ${}^\star\mathbb{X}$ by mapping every $x \in \mathbb{X}$ to the constant tuple such that $x_i = x$ for every $i \in I$. Any algebraic structure over $\mathbb{X}$ (group, ring, field) carries over to ${}^\star\mathbb{X}$ by almost pointwise extension. In particular, if $[x_i] \neq 0$, meaning that $\mu\{i \mid x_i = 0\} = 0$ we can define its inverse $[x_i]^{-1}$ by taking $y_i = x_i^{-1}$ if $x_i \neq 0$ and $y_i = 0$ otherwise. This construction yields $\mu\{i \mid y_i x_i = 1\} = 1$, whence $[y_i][x_i] = 1$ in ${}^\star\mathbb{X}$. The existence of an inverse for any non-zero element of a ring is indeed stated by the following first order formula: $\forall x(x = 0 \vee \exists y(xy = 1))$. More generally:

Lemma 5.2.3 (Transfer Principle) *Every first order formula is true over ${}^\star\mathbb{X}$ iff it is true over $\mathbb{X}$.*

5.2.2 Non-standard reals and integers

We just apply the above general construction to $\mathbb{X} = \mathbb{R}$ and $I = \mathbb{N}$ and denote the result by ${}^\star\mathbb{R}$, which is then a field according to the transfer principle. By the same principle, ${}^\star\mathbb{R}$ is totally ordered by $[u_n] \leq [v_n]$ iff $\mu\{n \mid v_n > u_n\} = 0$. For u, an arbitrary sequence of real numbers, let $\lim(u) \subseteq \overline{\mathbb{R}} =_{\text{def}} \mathbb{R} \cup \{-\infty, +\infty\}$ denote the (possibly empty) set of all limit points of sequence u: for $x \in \lim(u)$, let $v_k = u_{n_k}$ be a subsequence of u converging to x. If $\lim(u) \neq \emptyset$, there exists exactly one limit point $x \in \lim(u)$ such that $\mu\{n_k\} = 1$, and any other limit point yields a μ-measure 0 for the corresponding subsequence.[6] Call x the *standard part* of $[x_n]$ and we write $x = st([x_n])$. Infinite $x \in {}^\star\mathbb{R}$ have no standard part in $\mathbb{R}$.

It is also of interest to apply the general construction (5.4) to $\mathbb{X} = I = \mathbb{N}$, which results in the set ${}^\star\mathbb{N}$ of *non-standard integers*. ${}^\star\mathbb{N}$ differs from $\mathbb{N}$ by the addition of *infinite integers*, which are equivalence classes of sequences of integers whose essential limit is $+\infty$.

5.2.3 Integrals and differential equations

Any sequence (g_n) of functions $g_n : \mathbb{R} \mapsto \mathbb{R}$ pointwise defines a function $[g_n] : {}^\star\mathbb{R} \mapsto {}^\star\mathbb{R}$ by setting

$$[g_n]([x_n]) = [g_n(x_n)] \tag{5.5}$$

A function ${}^\star\mathbb{R} \to {}^\star\mathbb{R}$ which can be obtained in this way is called *internal*. Properties of and operations on ordinary functions extend pointwise to internal functions of ${}^\star\mathbb{R} \to {}^\star\mathbb{R}$. For $g : \mathbb{R} \to \mathbb{R}$, its *non-standard version* is the internal function ${}^\star g = [g, g, g, \ldots]$. The same notions apply to sets. An internal set $A = [A_n]$ is called *hyperfinite* if $\mu\{n \mid A_n \text{ finite}\} = 1$; the *cardinal* $|A|$ of A is defined as $[|A_n|]$.

6. So far this was a bit of hand waving. To prove this, let $x = \sup\{x \in \mathbb{R} \mid [x] \leq [x_n]\}$, where $[x]$ denotes the constant sequence equal to x. Since $[x_n]$ is finite, x exists and we only need to show that $[x_n] - [x]$ is infinitesimal. If not, then there exists $y \in \mathbb{R}, y > 0$ such that either $[y] < [x_n] - [x]$ or $[y] < [x] - [x_n]$, a contradiction. The unicity of x is clear.

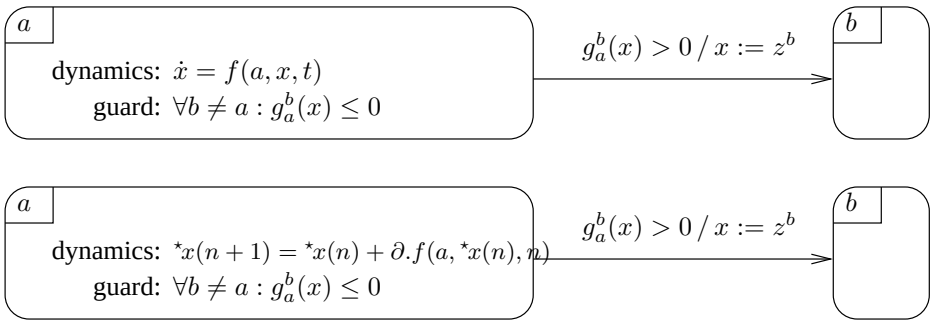


Figure 5.4: Hybrid system with mode switching, showing one transition. Top: standard, continuous time form. Bottom: non-standard form. We write for short ${}^\star x(n)$ instead of ${}^\star x(t_n)$ and ${}^\star x(t)$ is the piecewise constant, right continuous interpolation of ${}^\star x(t_n)$.

Now, consider an infinite number $N \in {}^\star\mathbb{N}$ and the set

$$T = \{\, 0, \tfrac{1}{N}, \tfrac{2}{N}, \tfrac{3}{N}, \ldots \tfrac{N-1}{N}, 1 \,\} \tag{5.6}$$

By definition, if $N = [N_n]$, then $T = [T_n]$ with

$$T_n = \{\, 0, \tfrac{1}{N_n}, \tfrac{2}{N_n}, \tfrac{3}{N_n}, \ldots \tfrac{N_n-1}{N_n}, 1 \,\}$$

hence $|T| = [|T_n|] = [N_n + 1] = N + 1$. Next, consider an internal function $g = [g_n]$ and a hyperfinite set $A = [A_n]$. We can then define the *sum* of g over A by

$$\textstyle\sum_{a\in A}\ g(a) =_{\text{def}} \left[\sum_{a\in A_n}\ g_n(a)\right]$$

If t is as above and $f : \mathbb{R} \to \mathbb{R}$ is a standard function, we get

$$\textstyle\sum_{t\in T} \frac{1}{N} {}^\star f(t) = \left[\sum_{t\in T_n} \frac{1}{N_n} f(t_n)\right] \tag{5.7}$$

Now, f continuous implies $\sum_{t\in T_n} \frac{1}{N_n} f(t_n) \to \int_0^1 f(t)dt$, so,

$$\textstyle\int_0^1 f(t)dt = st\big(\sum_{t\in T}\ \frac{1}{N} {}^\star f(t)\big) \tag{5.8}$$

Under the same assumptions, for any $t \in [0, 1]$,

$$\textstyle\int_0^t f(u)du = st\big(\sum_{u\in T, u\leq t}\ \frac{1}{N} {}^\star f(t)\big) \tag{5.9}$$

Now, consider the ODE with initial condition

$$\dot{x} = f(x, t), \;\; x(0) = x_0 \tag{5.10}$$

and assume it possesses a solution $[0,1] \ni t \mapsto x(t)$ such that function $t \mapsto f(x(t),t)$ is continuous. Rewriting (5.10) in integral form $x(t) = x_0 + \int_0^t f(x(u),u)du$ and using (5.9) yields

$$x(t) = st\big(\, x_0 + \textstyle\sum_{u\in T, u\leq t} \ \frac{1}{N} {}^\star f(x(u),u)\,\big) \tag{5.11}$$

Substitute in (5.11) $\partial = 1/N$ which is > 0 and infinitesimal, so that $T = \{t_n = n\partial \mid n = 0,\ldots,N\}$. Then, the expression in parentheses at the right hand side of (5.11) is the piecewise-constant right-continuous function ${}^\star x(t), t \in [0,1]$ such that, for $n = 1,\ldots,N$:

$$\begin{array}{rcl} {}^\star x(t_n) & = & {}^\star x(t_{n-1}) + \partial \times {}^\star f({}^\star x(t_{n-1}), t_{n-1}) \\ {}^\star x(t_0) & = & x_0 \end{array} \tag{5.12}$$

Hence, the solution x of ODE (5.10) on the one hand, and ${}^\star x$ as computed by algorithm (5.12) on the other, are related by $x = st({}^\star x)$. In other words, formula (5.12) can be seen as a *non-standard operational semantics* for ODE (5.10). In particular, formula (5.11) has the remarkable consequence that non-standard semantics are all equivalent whatever the particular choice for the infinitesimal step ∂ is.

We can push the above argument further by considering the (standard) hybrid system with mode switching depicted on Figure 5.4, top. In this figure, we show one transition of a system having a finite set $A \ni a, b, etc$ of *modes*. While in mode $a \in A$ the dynamics of the system are given by $\dot{x} = f(a,x,t)$. Mode switching is triggered when the first *zero-crossing* occurs: $g_a^b(x) > 0$, which causes a switch to mode b and the reset of x to the current value of some signal z^b. Now, suppose that this hybrid system possesses a solution $x(t), t \in \mathbb{R}_+$ such that: (i) $t \mapsto f(a,x(t),t)$ is continuous while the system stays within mode a, and (ii) the sequence of zero-crossings is either finite or diverging. Then the same reasoning as above can be used to rederive $x = st({}^\star x)$, for x and ${}^\star x$ solutions of the system shown at top and bottom, respectively, of Figure 5.4. The above analysis is summarized by the following ***Standardisation Principle***:

Principle 5.2.4 *Non-standard dynamical system of Figure 5.4, bottom, can always be considered, for any non-standard functions $f, g_a^b : {}^\star\mathbb{R} \mapsto {}^\star\mathbb{R}$. It possesses a well defined non-standard semantics ${}^\star x$. If, furthermore:*

1) functions f, g_a^b are internal, and

2) the hybrid system at the top of Figure 5.4 possesses a unique solution x such that

a) $t \mapsto f(a,x(t),t)$ is continuous within each mode, and

b) the sequence of zero-crossings is finite or diverging,

then $x = st({}^\star x)$ holds, regardless of the choice of infinitesimal step ∂. Thus, the non-standard operational semantics is intrinsic in that it does not depend on a particular ∂.

Note the "lazyness" of the argument justifying the non-standard semantics. It says: if the (standard) system possesses a "nice" solution, then this solution is found by the non-standard semantics. This

argument does not tell you whether or not the standard system possesses such a nice solution. In some sense, checking this is left for run time trial. The key point is that there is no need to check for any condition prior to considering the non-standard semantics, as it always has its own meaning. Non-standard semantics can be used in an assumption-agnostic way.

5.2.4 Semantic domain for hybrid systems

Using non-standard analysis has the following advantages:

1. Time set $\mathbb{T}$ is both *dense* in $\mathbb{R}$ and *discrete* in that each instant in $\mathbb{T}$ possesses a unique previous and next instant.
2. Since $\mathbb{T}$ is discrete, we can specify dynamical systems over $\mathbb{T}$ in full generality, without the need for referring to any kind of smoothness condition—e.g., as in (5.12).
3. Did the problem with the smoothness condition miraculously disappear? Not quite so. But it is postponed to the very end, at run time, thanks to Standardisation Principle 5.2.4: if the hybrid system under consideration has a unique solution in the usual mathematical sense, then the standardisation of our operational semantics computes it.

5.3 The SimpleHybrid Formalism

In this section we develop a tiny "mathematical language" for hybrid systems, we call it SIMPLEHYBRID. By this we mean a formalism that has the essential features of a language (a small set of primitive entities and statements, plus a composition operator), but that is designed primarily to facilitate mathematical manipulation. Primitive statements of SIMPLEHYBRID are equations of the following form:

$$\begin{aligned}
Eq_1: &\quad y = f([x]) \\
Eq_2: &\quad y = \mathbf{last}\,(x) \\
Eq_3: &\quad y = \dot{x} \\
Eq_4: &\quad \zeta = \mathbf{up}(z) \\
Eq_5: &\quad \dot{y} = x\ \mathbf{init}\ y_0\ \mathbf{reset}\ u \\
Eq_6: &\quad u = [v]\ \mathbf{every}\ [\zeta]\ \mathbf{init}\ u_0 \\
Eq_7: &\quad y = \mathbf{pre}\,(x)\ \mathbf{init}\ y_0
\end{aligned} \tag{5.13}$$

Note that, in a concrete programming language, equations Eq_1–Eq_4 and Eq_7 would appear as expressions. The above choice of primitives is, however, equally powerful. It is close to a Static Single Assignment (SSA) form with intermediate values stored in variables and is used to simplify the mathematical developments.

In (5.13) symbols u, x, y, v, z denote *variables*, with respective domains D_u, D_x, etc., taken from an underlying set $\mathcal{X}$ of variables and $[x] = [x_1, \ldots, x_n]$ is a tuple of variables. Symbol ζ denotes variables of *zero-crossing* taken from an underlying set $\mathcal{T} \subset \mathcal{X}$ of clock variables (generically denoted by the symbol τ). Clock variables take their values from the set of all *clocks*, where a clock is any subset of $\mathbb{R}_+$. Symbols y_0 and u_0 denote values. Finally, dotted variables $\dot{x}$ and $\dot{y}$ indicate derivatives. Equations Eq_1–Eq_7 define dynamical systems, or, equivalently, sets of behaviours with time index set $\mathbb{R}_+ = [0, +\infty)$. For example, Eq_3 means $\forall t \in \mathbb{R}_+ : y_t = \dot{x}_t$. Hybrid systems are specified via sets of equations of the form Eq_1–Eq_7, taken conjunctively. In the following we give an informal explanation of the above primitives, without making explicit the necessary continuity and smoothness assumptions for them to make sense. The corresponding mathematical semantics will be given in the next section.

We identify any clock τ with the boolean predicate it defines (the same convention also applies to zero-crossings):

$$\tau_t \quad = \quad \text{if } t \in \tau \text{ then } \mathrm{T} \text{ else } \mathrm{F} \tag{5.14}$$

For $X \subseteq \mathcal{X}$ finite, a *state* over X is an element $s \in D_X$ where $D_X = \prod_{x \in X} D_x$ and a *behaviour* over X is an element $\sigma \in \mathbb{R}_+ \to D_X$. For $x \in X$, let $\sigma(x) \in \mathbb{R}_+ \to D_x$ be the x-coordinate of σ, we call it a *signal*. By abuse of notation, and since no confusion will result, we write x_t instead of $\sigma \to \sigma(t)(x)$ and ζ_t instead of $\sigma \to \sigma(t)(\zeta)$. We now briefly review the primitives listed in (5.13).

Eq_1: means that $y_t = f(x_t^1, \ldots, x_t^n)$ holds for all t, where f is a total function over its domain and tuple $[x] = [x^1, \ldots, x^n]$;

Eq_2: means $y_t = x_{t_-} =_{\text{def}} \lim_{s \nearrow t} x_s$, i.e., y_t is the *left-limit* of x_s when s approaches t from below.

Eq_3: means $\forall t \in \mathbb{R}_+ : y_t = \dot{x}_t$.

Eq_4: defines the clock ζ such that, using convention (5.14):

$$\zeta_t \quad = \quad [z_{t_-} \leq 0] \wedge [z_t > 0]$$

Thus ζ selects the instants t at which z_t crosses zero from below, we call such a clock a *zero-crossing.* We will need to consider tuples $[\zeta_1 \ldots \zeta_k]$ of zero-crossings, denoted by the symbol $[\zeta]$.

Eq_5: For y, x two signals, y_0 a value, and u a *discrete* signal (see below), Eq_5 states that ODE $\dot{y}_t = x_t$ holds with initial condition y_0 and this ODE is reset to the value given by u at each instant of the discrete clock of u.

Eq_6: For u a signal, $u_0 \in \mathbb{R}^n$ a value, and $[\zeta] = [\zeta_1 \ldots \zeta_k]$ and $[v] = [v_1 \ldots v_k]$ two matching [7] tuples of zero-crossings and signals, Eq_6 states that u has clock $\zeta = \bigcup_{i=1}^{k} \zeta_i$ and, for every $t \in \bigcup_{j=1}^{i} \zeta_j$, $u_t = v_{i,t}$ holds, and $u_t = u_0$ for $t < t_1$, the first instant of ζ.

7. Say that two tuples $[u_1 \ldots u_k]$ and $[v_1 \ldots v_l]$ are *matching* if they possess identical numbers of components: i.e. $k = l$.

So far we have introduced the needed statements to define systems of ODE with mode changes and reset conditions. The additional statement Eq_7 allows embedding discrete time systems. We first need to clarify what "discrete time" means.

Signals are typed discrete or continuous

For each signal x, we assume a clock τ_x such that x is guaranteed constant on the complement of τ_x. We call τ_x the *clock of* x. A signal is typed *discrete* if either it has been declared as such, or if its clock is some zero-crossing. Otherwise it is *continuous.* For example, Eq_4 defines a discrete clock and signal u output by Eq_5 is discrete (note that it is not required by Eq_5 that input v is discrete).

Remark 5.3.1 *Mathematically, a clock is discrete if its restriction to any bounded interval of* $\mathbb{R}_+$ *is finite, a property that cannot be statically checked in general. The rationale for defining "discrete" as stated above is twofold: (i) it is a syntactic criterion and thus it can be statically checked; (ii) it generally matches the mathematical definition of a discrete clock.*

For instance, if $f : \mathbb{R}_+ \mapsto \mathbb{R}$ *is continuous, then* $zero(f) =_{\text{def}} \{t \in \mathbb{R}_+ \mid f(t) = 0\}$ *is a closed subset of* $\mathbb{R}_+$*. If, furthermore, all instants belonging to* $zero(f)$ *are isolated (i.e., are pairwise separated by a non-empty interval), then* $zero(f)$ *is either a finite set or a diverging sequence; in both cases it is discrete in the mathematical sense. Functions* f *from which zero-crossings are constructed would typically possess such properties.*

Of course, property (ii) is not guaranteed in all cases; for tricky signals, sets of zero-crossings may very well be Zeno or even a Cantor set (see example (5.2)). On the other hand, statically checking that a clock is discrete in the pure mathematical sense is simply not possible.

Some operators only apply to discrete signals. They define discrete signals by specifying their value at each instant of their clock. This by itself is not enough since it leaves the signal undefined before the first instant of its clock. An explicit initial value must therefore be given.

Eq_7**:** assumes x *discrete* and defines y as the delayed version of x by setting $\tau_y = \tau_x$ and setting the nth new value for y equal to the $(n-1)$th one of x; an initial condition y_0 is provided.

As previously stated, hybrid systems are specified in SIMPLEHYBRID via sets of equations of the form Eq_1–Eq_7, taken conjunctively. As an illustration, composing ODE Eq_5 with statement $x = f(y, v)$ of the form Eq_1, and reset Eq_6, yields the ODE

$$\dot{y} = f(y, v) \textbf{ init } y_0 \textbf{ reset } [v] \textbf{ every } [\zeta] \tag{5.15}$$

which means that ODE $\dot{y}_t = f(x_t)$ holds with initial condition y_0 and that this ODE is reset to a value given by z_i each time zero-crossing ζ_i occurs (where $\zeta_i = \textbf{up}(z_i)$).

5.4 Non-standard semantics

Throughout this section we fix a basic infinitesimal base step $\partial \approx 0$. Without loss of generality, we can assume that $\partial = [\varepsilon_n]$ for some decreasing sequence ε_n of reals converging to 0, see Section 5.2.2. Following [40], as our universal time base we replace $\mathbb{R}_+$ by the non-standard set

$$\mathbb{T} \quad = \quad \{t_n = n\partial \mid n \in {}^\star\mathbb{N}\}$$

For $t \in \mathbb{T}$, define

$$\begin{aligned} {}^\bullet t &= \max\{s \mid s \in \mathbb{T}, s < t\} \\ t^\bullet &= \min\{s \mid s \in \mathbb{T}, s > t\} \end{aligned} \tag{5.16}$$

We thus have ${}^\bullet t_n = t_{n-1}$ and $t_n^\bullet = t_{n+1}$. The key fact about $\mathbb{T}$ is that for every $u \in \mathbb{R}_+$ there exists a unique $t \in \mathbb{T}$ such that ${}^\bullet t < u \leq t$ and $t - u$ is infinitesimal. Thus $\mathbb{T}$ is, at the same time, dense in $\mathbb{R}_+$, and can still be handled as if it were discrete and totally ordered.

5.4.1 The semantics

A *hybrid system* is a tuple $S = (X, T, \Sigma)$, where $X \subseteq \mathcal{X}$ and $T \subseteq \mathcal{T}$ are finite and Σ is a set of behaviours over $X \cup T$. For $Y \supseteq X \cup T$, we can lift Σ to Y, written $\Sigma^{\uparrow Y}$, by taking all behaviours over Y whose projection over $X \cup T$ are in Σ. Then, for $S_i = (X_i, T_i, \Sigma_i), i = 1, 2$, we define the *parallel composition*

$$S_1 \parallel S_2 = \left(X, T, {\Sigma_1}^{\uparrow X \cup T} \cap {\Sigma_2}^{\uparrow X \cup T}\right), \tag{5.17}$$

where $X = X_1 \cup X_2$ and $T = T_1 \cup T_2$.

For a system $S = (X, \Sigma)$, specified as the parallel composition of a finite set of statements of one of the forms Eq_1–Eq_7, let $Clocks(S)$ be the (finite) set of all discrete clock variables involved in the specification of S. Then a *clock configuration* for S is a map

$$\kappa : \ Clocks(S) \mapsto \{\mathrm{F}, \mathrm{T}\}, \tag{5.18}$$

that assigns a truth value to each discrete clock variable of S. Clock configurations are used to indicate the presence or absence of each discrete clock of S at a given instant t. A clock configuration κ for S is called *reachable* if there exists a behavior σ and an instant t such that $\sigma(t)(T) = \kappa(T)$ for every $T \in Clocks(S)$.

The non-standard semantics of SIMPLEHYBRID is given in the second column of Table 5.4. Note the semantics of $\zeta = \mathbf{up}(z)$, which corresponds to a "weak preemption" in that the change in the sign of z at instant t results in the emission of a zero-crossing at the next instant $t^\bullet$.

The important fact about this semantics is that, unlike for a fixed step size (standard) semantics, it does not suffer from overshoot problems for zero-crossings or even zenoness, or any need of mentioning continuity properties, since steps are infinitesimal but "discrete". Yet the semantics is still statically defined, as was desired.

5.4.2 Back to the examples

Observe that Figures 5.1–5.3 plot the non-standard semantics of SIMPLEHYBRID systems (5.1–5.3) according to the second column of Table 5.4. We now discuss these examples in detail.

Example (5.1) The mysteries regarding example (5.1) are now clarified: the first zero-crossing occurs at time $t = 1 + \partial$ (corresponding to 1_+ of Figure 5.1). Then, setting $\varepsilon = 2\partial$, zero-crossings occur repeatedly with a period of 4ε, forever, thus filling the time line until $+\infty$. The non-standard domain of time $\mathbb{T} = \{n\partial \mid n \in {}^\star\mathbb{N}\}$ allows for having several successive zero-crossings each of zero duration, with time still eventually diverging, since we can always find n infinitely large enough so that $n\partial > t$ for $t \in \mathbb{R}_+$. Now, the key feature of example (5.1) is that, despite being well defined within a non-standard analysis framework, there is no possible standardisation.

Example (5.2) In contrast, consider example (5.2). We claim that standardisation $\mathbf{y} = st(y)$ exists and has the *averaged* dynamics given just before (5.3). To show this, we use a variation of the argument developed in analysing formulas (5.6)–(5.8), see Sections 5.2.2 and 5.2.3. Let (x, y) be the non-standard semantics of (5.2), i.e., given by Figure 5.2. Again, let $\varepsilon = 2\partial$ and ε_n be the sequence of positive (standard) reals converging to 0 such that $\varepsilon = [\varepsilon_n]$. Consider the following sequence of (standard) dynamical systems y^n

$$\begin{aligned} \dot{x}^n = \;& 0 \;\mathbf{init}\; -\operatorname{sgn}(y_0) \\ & \mathbf{reset}\,[-1,1]\,\mathbf{every}\,\mathbf{up}[y^n - \varepsilon_n, -y^n + \varepsilon_n] \\ \dot{y}^n = \;& x^n \;\mathbf{init}\; y_0 \end{aligned} \tag{5.19}$$

The behaviour of y^n can again be seen on Figure 5.2, with, however, ε_n substituted for ε. For any $k \in {}^\star\mathbb{N}$, we have $k\varepsilon = [k\varepsilon_n]$ and thus, since x alternates between -1 and $+1$ at multiples of ε (see Figure 5.2), it follows that $x(k\varepsilon) = [x^n(k\varepsilon_n)]$, expressing that $x = [x^n]$, see (5.5). The same reasoning shows that $y = [y^n]$. On the other hand, using elementary arguments from standard analysis, (5.19) defines a sequence of functions $y^n_t, t \geq 0$ that converges uniformly to $\mathbf{y}$ defined just before (5.3) when $n \nearrow +\infty$ — this part of the argument cannot be invoked for example (5.1). The above analysis shows that $\mathbf{y} = st(y)$ where y is given by (5.2) and $\mathbf{y}$ is given just before (5.3).

Example (5.3) It is to some extent an intermediate case. While not "as pathological" as example (5.1), its non-standard semantics is still not standardisable because of the double jump at $t = 1$. If we insist that only systems having a standardisable semantics are accepted, then this program should be rejected. If, however, we still want to accept it, then its effective semantics must be based on extra, somehow arbitrary, principles. Two alternative approaches can be considered:

1. The first one consists in staying with the non-standard semantics when collapsing ∂ to zero, thus *super-dense time* following [124, 125] is used. In particular, all zero-crossings remain with the same scheduling. This requires being able to statically check that there are only finitely many cascaded zero-crossings at any given instant.
2. The other possibility is to adopt the policy of synchronous languages: the successive non-standard micro-steps at $t = 1$ are collapsed to be simultaneous and then the zero-crossing of y has priority, giving rise to a single jump of size 1 for x at $t = 1$. Super-dense time is then not needed, but we must be able to statically check that no signal exhibits more than one zero-crossing at any given instant.

We will see at the end of Section 5.5 that such static checks are made possible by our constructive semantics. To summarize: (i) any SIMPLEHYBRID system possesses a non-standard semantics; (ii) the latter may or may not be standardisable, but it does capture subtle or even pathological cases; (iii) sufficient, statically checkable conditions for being standardisable will be provided by the constructive semantics.

Simultaneous zero-crossings In [156], R. Nikoukhah advocates rejecting simultaneous zero-crossings, unless it can be statically checked that they should occur. He advocates that, in case simultaneous zero-crossings do incidentally occur, they should be interleaved non-deterministically. Numerical solvers (e.g., Sundials CVODE) do detect simultaneous zero-crossing, and so discarding some or processing them sequentially would be a source of non-determinism. We prefer a synchronous interpretation in which the programmer decides what happens when zero-crossings occur simultaneously. Writing, for example, a handler

$$[0, 1, -1] \ \mathbf{every} \ [\mathbf{up}(x) \,\&\, \mathbf{up}(y), \mathbf{up}(y), \mathbf{up}(x)] \,,$$

would mean that the value 0 is returned when both x and y cross zero, 1 when it is only $\mathbf{up}(y)$, and -1 when it is only $\mathbf{up}(x)$. Note that this synchronous interpretation conforms with the behaviour of Simulink.

5.5 Constructive semantics

As for any synchronous language, the *constructive semantics* [36] formalizes how a reaction should be executed, that is, how the different actions should be scheduled at a given instant, given a program's underlying causality constraints. Different approaches have been proposed.

G. Berry [36] advocated using a Scott domain with an extra value "undefined", to be interpreted as "not executed yet"; the domain of values is made a flat partial order by setting "undefined $<$ any other value". Undefined should not be confused with the special status *absent*, which is characteric of synchronous languages and belongs to the domain of values. Using this extra status *undefined*, Esterel reactions are encoded as sets of equations in this Scott domain and the minimal fixpoint is

sought, by iterating from the configuration where all variables and signals are undefined. If in the fixpoint all variables are uniquely defined, then the program is deterministic and can be executed. An earlier approach was proposed by F. Boussinot [43] based on micro-step automata, which are automata describing the allowed schedules and decomposing a reaction into micro-steps of atomic operations. These two approaches were indeed developed and shown equivalent for Signal [29]. Here we develop a Scott semantics.

Scheduling constraints Let $\perp$ be a special value not belonging to any domain D_x, to be interpreted as "not evaluated yet".[8] Define, for any $x \in \mathcal{X}$, $D_x^\perp = D_x \cup \{\perp\}$. Write $x = \top$ to mean that $x \neq \perp$. Let $\lhd$ be the *scheduling constraint* that relates any two variables u and v, that have, respectively, domains $D_u^\perp$ and $D_v^\perp$:

$$u \lhd v \quad =_{\text{def}} \quad [u = \top] \ \vee \ [v = \perp] \tag{5.20}$$

i.e., $u \lhd v$ means $[v = \top] \Rightarrow [u = \top]$, which formalizes that "v cannot be evaluated strictly before u". In particular, for any clock τ,

$$\forall t \in \tau \Rightarrow x_t \lhd y_t \quad =_{\text{def}} \quad [x_t = \top] \vee [y_t = \perp] \vee [\tau_t = \text{F}]$$

where τ_t is defined in (5.14). Observe that statement $v = f(u)$, where f is a function, abstracts as $u \lhd v$ since v can be substituted by its evaluation $f(u)$ everywhere. Relation $\lhd$ captures causality constraints within a system of equations.

The constructive semantics is obtained by abstracting, in the non-standard semantics (second column of Table 5.4), any statement of the form $y_t = exp$ where expression exp involves variables x_s, u_s, τ_s for $s = {}^\bullet t, t, t^\bullet$, by the scheduling constraints $x_s \lhd y_t$, $u_s \lhd y_t$, or $\tau_s \lhd y_t$, respectively. For example, $y_t = f(x_t)$ is abstracted as $x_t \lhd y_t$.

Observe that the semantics of $\zeta = \mathbf{up}(z)$ corresponds to a "weak preemption" in that the change in the sign of z at instant t results in emitting a zero-crossing at the next instant $t^\bullet$. Hence, no clock occurs on any consequent part of a zero-time causality constraint. Therefore, preconditions such as "$t \in \tau \Rightarrow$" in the mid column of Table 5.4 do not impair the validity of the above mentioned abstractions.

Pre- and post-variables In writing the constructive semantics, we would like to abstract away dummy time index t. To this end, for each variable $x \in X$ of the considered system S, we augment X with the two auxiliary variables ${}^\bullet x$ and $x^\bullet$, such that ${}^\bullet x_t = x_{\bullet t}$ and $x_t^\bullet = x_{t\bullet}$ hold for every t. Using these auxiliary variables and clock variables, time index t can be abstracted away in the constructive semantics. Using the above notations, the constructive semantics is given in the last column of Table 5.4.

8. This notation deviates from the historically established use of symbol $\perp$ in synchronous languages to denote absence. Absence of a signal in a reaction is a well defined status that is the result of evaluating the considered reaction. "Absence" and "not evaluated yet" should therefore not be confused.

It is indeed tempting to extend SIMPLEHYBRID with the statement $x \lhd y$, seen as a statement belonging to the generic family in row 1. Doing this makes it possible to express the causality analysis of a SIMPLEHYBRID program, and even additional scheduling constraints that the programmer may want to enforce, in the language SIMPLEHYBRID itself. This trick is not new; it was already used for the Signal synchronous language [32].

Causality circuits Using the above abstraction, for each given clock configuration of S, the transitive closure of relation $\lhd$ is a *pre-order* on X — by abuse of notation, we also call it $\lhd$. If S is such that $\lhd$ is a *partial order* for any reachable clock configuration (see (5.18) and below), then no causality circuit occurs in S and the different variables can be evaluated according to any order compatible with $\lhd$. Since no clock occurs on any consequent part of a zero-time causality constraint, the only possible cause of circuits in relation $\lhd$ is via sets of statements of the form Eq_1. We thus formally justify here the rule that no delay-free, derivative-free, data flow circuit should exist in the considered program.

Single assignment condition Say that system S obeys the single assignment condition if no variable of S sits on the left hand side of two or more equations. The following holds:

Lemma 5.5.1 *If S possesses no causality circuit and obeys the single assignment condition, then it is deterministic and the partial order $\lhd$ at each clock configuration specifies all correct schedulings for the execution of S.*

Interactions of sub-systems The constructive semantics conveys the necessary information to identify when several ODE equations of type Eq_5 must be jointly submitted to the same solver because they are coupled in all directions — coupling may involve the ODEs and/or their associated zero-crossings and resets. Example (5.2) is one such example. Of more interest is the ability to identify the *lack* of interaction, for example when two sub-systems only communicate in a unidirectional way. This addresses issue 3) raised in the introduction.

Cascaded zero-crossings The constructive semantics of example (5.1) involves the following scheduling constraints, where ζ_u denotes the zero-crossing induced by signal u: ${}^\bullet z \lhd \zeta_z \lhd x$, ${}^\bullet x \lhd (\zeta_x, \zeta_{-x}) \lhd y$, and ${}^\bullet y \lhd (\zeta_y, \zeta_{-y}) \lhd x$, thus statically showing that infinitely many cascaded zero-crossings for x and y can occur.

On the other hand, the constructive semantics of example (5.2) involves the single scheduling constraint ${}^\bullet y \lhd (\zeta_y, \zeta_{-y}) \lhd x$, thus statically showing that at most one zero-crossing for y can occur at any given time.

Finally, the constructive semantics of example (5.3) involves scheduling constraints ${}^\bullet z \lhd \zeta_z \lhd (x, y)$ and ${}^\bullet y \lhd \zeta_y \lhd x$, thus showing a risk of at most two cascaded zero-crossings for x.

To conclude, our constructive semantics is powerful enough to support the static checkings required for cascaded zero-crossings, see the discussion of examples (5.1–5.3).

5.6 Off-the-shelf compilers

In this section we explain how to derive a SIMPLEHYBRID compiler by reusing a legacy synchronous language engine in combination with a legacy ODE solver. The synchronous language engine will regard ODE solutions between two successive zero-crossings as just another (big!) step, regardless of the fact that this step is managed by an external entity, namely the solver. In the chapter we only explain the principles, the detailed development of such a tool will be presented elsewhere. The key idea is to structure SIMPLEHYBRID systems in a specific way. Decompose every SIMPLEHYBRID system S as

$$S \quad = \quad S_{\mathrm{ODE}} \parallel S_{\mathrm{noODE}}, \text{ where} \tag{5.21}$$

- Subsystem S_{ODE} collects all equations in S of the form Eq_1–Eq_5;
- Subsystem S_{noODE} collects all equations in S of the form Eq_1, Eq_2, Eq_6 and Eq_7;
- Since equations of types Eq_1 and Eq_2 appear in both subsystems and we want to preserve the single-assignment condition of S, if it does hold, we must guard these equations by clock conditions. Thus, such equations are assigned to S_{noODE} at instants of zero-crossing, and otherwise they are assigned to S_{ODE}.

One way to achieve the last type of separation would be to extend the expressiveness of SIMPLEHYBRID by allowing for equations *guarded by clocks*. For instance, Eq_1 would become "$\mathbf{on}\ \tau\ :\ y\ =\ f([x])$". This would be feasible but it would increase the complexity of the formalism while adding little value in the study of fundamental issues. Thus, we prefer to keep SIMPLEHYBRID as it is.

So we instead propose a simple convention to acheive the necessary expressiveness. For S, a system, let ζ_S be the union of all zero-crossings involved in S — there are only finitely many of them, thus ζ_S is a discrete clock too. While assigning an equation of type Eq_1 to S_{noODE}, we guard it by ζ_S as the following equation of type Eq_6:

$$\underbrace{y = f([x])}_{\text{original equation}} \quad \underbrace{\mathbf{every}\ \zeta_S}_{\text{added guard}} \tag{5.22}$$

We also do the same for equations of type Eq_2. On the other hand, the same Eq_1 and Eq_2 equations are assigned to S_{ODE} and *it is understood that these equations are preempted by corresponding guarded equations* (5.22) *at instants of* ζ_S. This trick avoids the duplication of equations at any given instant and allows decomposition (5.21) to preserve the single-assignment condition if it was satisfied by S.

Now, S_{noODE} is nothing but a synchronous program (it can be encoded in Lustre). On the other hand, S_{ODE} is exactly what a state-of-the-art ODE solver (such as Sundials [9]) can compute, namely solving ODEs with given initial conditions and resetting values, and halting when some specified variables are subject to zero-crossings.

9. https://computation.llnl.gov/casc/sundials/main.html

A prototype tool has been developed based on these principles using the Sundials solver. In Section 5.8 we report some experiments on the examples of the introduction, showing their behavior in Simulink and with the tool.

5.7 Hitting balls example

We consider the case of two balls hitting each other along a wall as shown on Figure 5.5. The figure shows the initial condition $d_1 < d_2 = w_2 = 0$ and $w_1 > 0$, meaning that ball 2 sits steady on contact of the wall, whereas ball 1 is going to hit it. To simplify, these are ideal balls of zero diameter. For convenience, the system is activated at initial time $t = -d_1/w_1$, so that the first hit occurs right after $t = 0$ (formally, at time $t = \partial$).

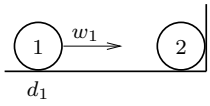


Figure 5.5: The hitting balls example: initial condition.

Corresponding equations are:

$$\begin{array}{rcl} \dot{x}_1 & = & v_1 \textbf{ init } d_1 \\ \dot{x}_2 & = & v_2 \textbf{ init } d_2 \\ \dot{v}_1 & = & 0 \textbf{ init } w_1 \textbf{ reset last } (v_2) \\ & & \qquad \textbf{every up}[x_1 - x_2] \\ \dot{v}_2 & = & 0 \textbf{ init } w_2 \textbf{ reset } [\textbf{last } (v_1) , -\textbf{last } (v_2)] \\ & & \qquad \textbf{every up}[x_1 - x_2, x_2] \end{array} \tag{5.23}$$

The non-standard semantics yields:

1. at $t = \partial$, $x_1 = \partial.w_1 > 0$, which causes $x_1 - x_2$ to have a zero-crossing.
2. As a result, at $t = 2\partial$ the two balls exchange their velocities: $v_1 = 0$ and $v_2 = w_1$.
3. At $t = 3\partial$, $x_1 = 2\partial.w_1$ and $x_2 = \partial.w_1$, which causes x_2 to have a zero-crossing.
4. Hence at $t = 4\partial$, $x_1 = x_2 = 2\partial.w_1$, $v_1 = 0$ and $v_2 = -w_1$.
5. At $t = 5\partial$, $x_1 = 2\partial.w_1$ and $x_2 = \partial.w_1$, which causes $x_1 - x_2$ to have a zero-crossing.
6. Hence at at $t = 6\partial$, $x_1 = 2\partial.w_1$, $x_2 = 0$, $v_1 = -w_1$ and $v_2 = 0$.

Then, ball 1 moves to $-\infty$ and no more zero-crossing occurs. Observe that this non-standard semantics is not standardisable. For this example, the super-dense time interpretation 1) at the end of Section 5.4.2 is preferred.

5.8 Experimental results

We have modelled examples (5.1) to (5.3) in both Simulink (version 7.7.0.471, R2008b) and a prototype based on the Sundials (version 2.4.0) CVODE library [105].

5.8.1 Using Simulink

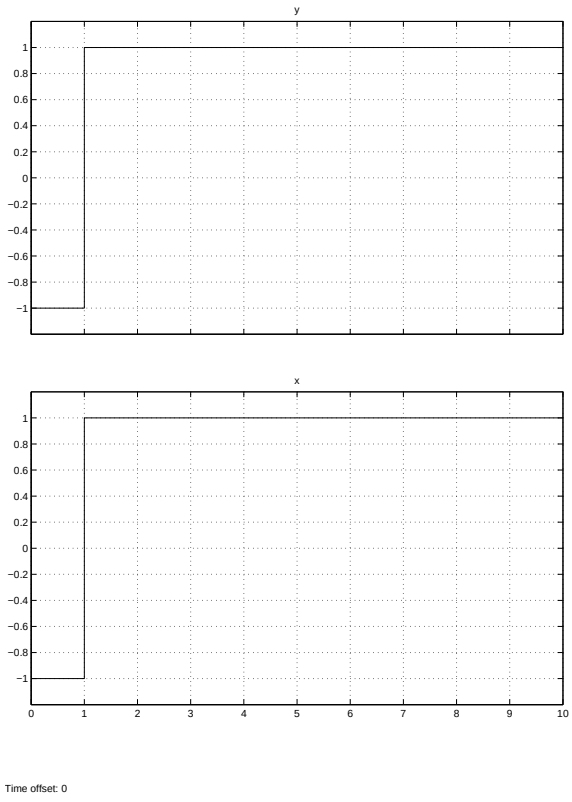


Figure 5.6: Example (5.1) in Simulink

Example (5.1) The Simulink model corresponds to the following set of equations [10]:

$$
\begin{aligned}
y &= 1/s(iy, updown(lx), 0)\\
x, lx &= 1/s(ix, up(z_x), 0)\\
z &= 1/s(-1, 1)\\
i_y &= switchup(lx, 1, switchup(-lx, -1, -1))\\
i_x &= switchup(y, -1, switchup(-y, 1, switchup(z, 1, -1)))\\
z_x &= switchup(y, 1, switchup(-y, 1, switchup(z, 1, -1)))
\end{aligned}
$$

We write $1/s(init, zero, input)$ for the integration of a signal $input$ with initial value $init$ and reset given by a zero-crossing condition $zero$. The integration operator may return a second output (the so-called state port which corresponds to the left limit of signal x). We write it lx in the equations above. $updown(r)$ means that a zero-crossing is detected when r crosses zero in any direction; $up(r)$ when r crosses zero from a negative to a positive value. $switchup(x, e_1, e_2)$ returns the value of e_1 when x crosses zero, the value of e_2 otherwise [11]. The zero-crossing handler $[-1, 1, 1]$ `every` $[y, -y, z]$ is encoded with two equations: the equation ix defines the initial value for x whereas z_x is true when y or $-y$ or z cross zero. This is encoded by the integer expression

$$switchup(y, 1, switchup(-y, 1, switchup(z, 1, -1)))$$

A simulation is given in Figure 5.6. It shows that Simulink does not introduce a delay in the effect of a zero-crossing. The numerical solver runs and stops at time $t = 1$ with the first zero-crossing $up(z)$. This implies several zero-crossings:

- $z_x = 1$ as y depends on lx and thus did not cross zero.
- As a consequence, x is reset with the value of ix which equals 1.
- At time $t + \epsilon$, y is reset to $i_y = 1$.
- Then, no more zero-crossing occurs and the two signal x and y remain constant.

This effect is a direct consequence of the priority between zero-crossing made explicit by the programmer.

Example (5.2) The Simulink model is depicted in Figure 5.7 with the corresponding outputs for x and y. In normal mode, the simulation fails because of too many zero-crossing at instant $t = 1$ [12].

10. For lack of space, we do not show the corresponding block-diagram. We show a systematic encoding here with no simpliciation of the code.

11. In Simulink, $switchup(x, e_1, e_2)$ is implemented with a switch operator and a hit-crossing operator applied to x.

12. Simulink stops with the error: At time 1.000000000019998, simulation hits (1000) consecutive zero crossings. Consecutive zero crossings will slow down the simulation or cause the simulation to hang. To continue the simulation, you may 1) Try using Adaptive zero-crossing detection algorithm or 2) Disable the zero crossing of the blocks shown in the following table.

We output the result of a simulation using adaptive zero-crossing detection. We take $y_0 = -1$.

$$
\begin{aligned}
x &= 1/s(ix, updown(y), 0)\\
y &= 1/s(y_0, x)\\
ix &= switchup(y, -1, switchup(-y, 1, -y_0))
\end{aligned}
$$

Results in Figure 5.7 show that y stick to zero (more or less a threshold parameter used by the adaptive algorithm) whereas x alternate from 1 to -1. Note that because of the use of the adaptive algorithm, the signal x may stay some time at 1 or -1 before changing.

Example (5.3) The corresponding Simulink model is given below as a set of equations. This time, the first integration block returns both x and the state port lx.

$$
\begin{aligned}
x, lx &= 1/s(ix, up(zx), 0)\\
z &= 1/s(-1, 1)\\
y &= 1/s(iy, up(z), 0)\\
ix &= switchup(y, lx + 1, switchup(z, lx + 2, 0))\\
zx &= switchup(y, 1, switchup(z, 1, -1))\\
iy &= switchup(z, 1, -1)
\end{aligned}
$$

A run of the example is given in Figure 5.8. It shows that Simulink does not introduce a delay in the effect of a zero-crossing:

- The numerical solver runs an stops at time $t = 1$ with the first zero-crossing $up(z)$.
- This implies a second zero-crossing for the equation of y with value 1. As a consequence, y crosses zero (going from -1 to 1). Time dit not progress, i.e., the zero-crossing $up(y)$ is synchronous with $up(z)$.
- As a consequence, the equation for x is reset with value $lx + 1$, that is, 1 since $up(y)$ is treated before $up(z)$ in the handler $switchup(y, lx + 1, switchup(z, lx + 2, 0))$.

This confirms that a cascade of zero-crossing is instantaneous in Simulink. That is, Simulink takes the following interpretation for a zero-crossing:

$$\mathbf{up}(x)_t = [x_{\bullet t} \leq 0] \wedge [x_t > 0]$$

which means that the effect of a zero-crossing is instantaneous. This hilight the main benefit of non-standard analysis as a model for reasonning on hybrid systems and the treatment of zero-crossing. Choosing the Simulink interpretation would only change the definition of $\mathbf{up}(.)$ in the non-standard semantics.

phase	time	x	y	z
I	0.000000e+00	-1.000000e+00	-1.000000e+00	-1.000000e+00
C	1.000000e-01	-1.000000e+00	-1.000000e+00	-9.000000e-01
C	2.000000e-01	-1.000000e+00	-1.000000e+00	-8.000000e-01
C	3.000000e-01	-1.000000e+00	-1.000000e+00	-7.000000e-01
C	4.000000e-01	-1.000000e+00	-1.000000e+00	-6.000000e-01
C	5.000000e-01	-1.000000e+00	-1.000000e+00	-5.000000e-01
C	6.000000e-01	-1.000000e+00	-1.000000e+00	-4.000000e-01
C	7.000000e-01	-1.000000e+00	-1.000000e+00	-3.000000e-01
C	8.000000e-01	-1.000000e+00	-1.000000e+00	-2.000000e-01
C	9.000000e-01	-1.000000e+00	-1.000000e+00	-1.000000e-01
C	1.000000e+00	-1.000000e+00	-1.000000e+00	-2.235451e-14
C′	1.000000e+00	-1.000000e+00	-1.000000e+00	7.786350e-14
Z	1.000000e+00	up(z)		
D	1.000000e+00	1.000000e+00	-1.000000e+00	7.786350e-14
Z	1.000000e+00	up(x)		
D	1.000000e+00	1.000000e+00	1.000000e+00	7.786350e-14
Z	1.000000e+00	up(y)		
D	1.000000e+00	-1.000000e+00	1.000000e+00	7.786350e-14
Z	1.000000e+00	up(-x)		
D	1.000000e+00	-1.000000e+00	-1.000000e+00	7.786350e-14
Z	1.000000e+00	up(-y)		
D	1.000000e+00	1.000000e+00	-1.000000e+00	7.786350e-14

Table 5.1: Log of example (5.1) (prototype tool)

5.8.2 Using the Sundials-based Prototype

We have developed a prototype implementation of our language in Ocaml, which comprises a generic interface to the CVODE library (using serial vectors), and an implementation of the algorithm that alternates between continuous phases and discrete phases in response to zero-crossings. Each example was manually translated into a single Ocaml function that is called by Sundials during continuous phases, and by the algorithm directly during discrete phases.

Example (5.1) The results of running the prototype tool on example (5.1) are shown in Table 5.1. The first row ('I') shows the initial state values, it is followed by a series of executions of the CVODE solver ('C') during which the states evolve according to their derivatives, and then just after 1.0, a zero-crossing is detected ('Z'). The values of the continuous states at the time of the zero-crossing ('C′'), become *last* values during the subsequent discrete phase ('D'). The first zero-crossing occurs for $\mathbf{up}(z)$. It triggers an unbounded cascade of discrete phases, after each of which another (single and non-simultaneous) zero-crossing is detected. The sequence $\mathbf{up}(x)$, $\mathbf{up}(y)$, $\mathbf{up}(-x)$, $\mathbf{up}(-y)$ is repeated indefinitely without the continuous solver ever being reinvoked.

Example (5.2) The results for example (5.2) are shown in Table 5.2. The value of y exceeds zero and triggers the zero-crossing $\mathbf{up}(y)$ just after $t = 1.0$. Then, the value of x is changed from 1.0 to -1.0 during the discrete phase, but as there are no further zero-crossings the continuous solver is called again. Another zero-crossing, $\mathbf{up}(-y)$, is discovered almost immediately and

phase	time	x	y
I	0.0000000000000000e+00	1.000000e+00	-1.000000e+00
C	1.0000000000000000e-01	1.000000e+00	-9.000000e-01
C	2.0000000000000000e-01	1.000000e+00	-8.000000e-01
C	3.0000000000000000e-01	1.000000e+00	-7.000000e-01
C	4.0000000000000000e-01	1.000000e+00	-6.000000e-01
C	5.0000000000000000e-01	1.000000e+00	-5.000000e-01
C	6.0000000000000000e-01	1.000000e+00	-4.000000e-01
C	7.0000000000000000e-01	1.000000e+00	-3.000000e-01
C	7.9999999999999999e-01	1.000000e+00	-2.000000e-01
C	8.9999999999999999e-01	1.000000e+00	-1.000000e-01
C	9.9999999999999999e-01	1.000000e+00	-4.464441e-14
C′	1.0000000000000100e+00	1.000000e+00	5.557360e-14
Z	1.0000000000000100e+00	up(y)	
D	1.0000000000000100e+00	-1.000000e+00	5.557360e-14
C′	1.0000000000000175e+00	-1.000000e+00	-1.974954e-14
Z	1.0000000000000175e+00	up(-y)	
D	1.0000000000000175e+00	1.000000e+00	-1.974954e-14
C′	1.0000000000000195e+00	1.000000e+00	9.288416e-18
Z	1.0000000000000195e+00	up(y)	
D	1.0000000000000195e+00	-1.000000e+00	9.288416e-18
C′	1.0000000000000215e+00	-1.000000e+00	-1.972025e-14
Z	1.0000000000000215e+00	up(-y)	
D	1.0000000000000215e+00	1.000000e+00	-1.972025e-14
C′	1.0000000000000234e+00	1.000000e+00	3.853879e-17
Z	1.0000000000000234e+00	up(y)	
D	1.0000000000000234e+00	-1.000000e+00	3.853879e-17
C′	1.0000000000000254e+00	-1.000000e+00	-1.969104e-14
Z	1.0000000000000254e+00	up(-y)	
D	1.0000000000000254e+00	1.000000e+00	-1.969104e-14
C′	1.0000000000000274e+00	1.000000e+00	6.770241e-17

Table 5.2: Log of example (5.2) (prototype tool)

another discrete phase is triggered during which x is changed back to 1.0. This process is repeated indefinitely; time is advanced in small increments by the continuous solver, and the value of x is alternated between 1.0 and -1.0 by intervening discrete phases. The observed behaviour thus approximates the ideal behaviour; a small overshoot, which is proportional to step size chosen by the continuous solver, effectively simulates the ε of the non-standard semantics. Note that the time column is given with a greater precision than in the other examples. Without the extra significant figures, it appears as if the simulation iterates without bound at $t = 1.0$. As it is, time barely advances just as is in Simulink when the adaptive zero-crossing detection algorithm is not used.

Example (5.3) The results for example (5.3) are shown in Table 5.3. Both x and y are constant throughout the initial continuous phases, but z increases steadily from -1.0. The first zero-crossing, $\mathbf{up}(z)$, is triggered just after z crosses 0.0. The ensuing discrete phase sees x incremented by 2.0 and y set to 1.0. The latter update triggers the zero-crossing $\mathbf{up}(y)$, which causes another discrete phase to be executed at the same instant of time. During this second discrete phase, x is incremented by 1.0. The simulation then continues with an unbounded number of continuous phases. Note that, during a discrete phase, the effects of changes to variables on zero-crossing expressions

phase	time	x	y	z
I	0.000000e+00	0.000000e+00	-1.000000e+00	-1.000000e+00
C	1.000000e-01	0.000000e+00	-1.000000e+00	-9.000000e-01
C	2.000000e-01	0.000000e+00	-1.000000e+00	-8.000000e-01
C	3.000000e-01	0.000000e+00	-1.000000e+00	-7.000000e-01
C	4.000000e-01	0.000000e+00	-1.000000e+00	-6.000000e-01
C	5.000000e-01	0.000000e+00	-1.000000e+00	-5.000000e-01
C	6.000000e-01	0.000000e+00	-1.000000e+00	-4.000000e-01
C	7.000000e-01	0.000000e+00	-1.000000e+00	-3.000000e-01
C	8.000000e-01	0.000000e+00	-1.000000e+00	-2.000000e-01
C	9.000000e-01	0.000000e+00	-1.000000e+00	-1.000000e-01
C	1.000000e+00	0.000000e+00	-1.000000e+00	-1.500536e-16
C′	1.000000e+00	0.000000e+00	-1.000000e+00	1.000680e-13
Z	1.000000e+00	up(z)		
D	1.000000e+00	2.000000e+00	1.000000e+00	1.000680e-13
Z	1.000000e+00	up(y)		
D	1.000000e+00	3.000000e+00	1.000000e+00	1.000680e-13
C	1.100000e+00	3.000000e+00	1.000000e+00	1.000000e-01
C	1.200000e+00	3.000000e+00	1.000000e+00	2.000000e-01

Table 5.3: Log of example (5.3) (prototype tool)

are not detected immediately, rather any new zero-crossings are detected after the discrete phase, i.e. after variables have been reset as necessary, when the last values of zero-crossing expressions are compared with their new values. There is thus no question of priority in this example: $\mathbf{up}(x)$ occurs strictly before $\mathbf{up}(y)$, even though no simulation time elapses between them.

5.9 Related work

Studies on hybrid systems modelers from a semantics point of view are not so numerous. We discuss the few we consider relevant for comparison. First of all, we recall previous work [27]. In fact, the agenda presented in that paper closely resembles the one we develop here. Except that, in [27] the tool of non-standard analysis was not used. Consequently, [27] suffers from some hand waving, as careful readers will notice.

Perhaps the attempt most similar to ours is the work of the Ptolemy group, by E. Lee and H. Zheng [124, 125], which studies the handling of discontinuities in hybrid systems modelers. They apply the model of *tagged signals* [123]. Events are tagged with an extended time index taken from the set $\mathbb{R}_+ \times \mathbb{N}$ with its associated lexicographic order. This set is referred to by the authors as *super-dense* time. This type of multi-dimensional time set was considered earlier for discrete time systems models in the area of synchronous languages [32, 33]. Our approach avoids using super-dense time because the non-standard index set $\mathbb{T}$ is both discrete and dense. The existence of a previous instant ${}^\bullet t$ and a next instant $t^\bullet$ was used in Table 5.4, replacing the multi-dimensional instants $(t, 0)$ and $(t, 1)$ of [124, 125]. On another aspect, the approach [124, 125] is made complicated by issues of smoothness, Lipschitzness, existence and uniqueness of solutions, Zenoness, etc (see section 6 of [124] on “Ideal Sover Semantics” and section 7 of [125] on “Continuous Time Models”). These issues do not simply dissappear in our approach, instead

they are more or less postponed to run time. Finally, we do not see how our firest two examples could be analyzed within the framework of [124, 125].

The work described by P. Mosterman and his co-workers at The Mathworks [134] is also very interesting. It attempts to establish the Simulink modeler on a solid semantic basis. The contribution of the paper is to show how (a restricted class of) variable step solvers can be given a functional *stream* semantics [62]. To achieve this, the class of solvers is first restricted to those relying on *explicit schemes*, as *implicit* ones cannot be put in explicit functional form. While this indeed provides a hybrid systems modeler with a stream semantics, the semantics is extremely complex since it the discretization method is made explicit — in particular, changing the method changes the semantics. This approach precludes using implicit schemes, although they are valuable from the point of view of numerical analysis.

In [156], R. Nikoukhah discusses cascaded zero-crossings. He advocates rejecting the "synchronous" interpretation of them, see the interpretation 2) of Example (5.3) in Section 5.5. He favors instead a micro-step style of interpretation, where cascaded zero-crossings interleave non-deterministically. We prefer a synchronous interpretation in which the programmer makes explicit what to do when two zero-crossings occur. Then non-determinism arises solely in numerical solvers, and not from the semantics of a program. Because the effect of $\mathbf{up}(e)$ is delayed by one cycle of $\mathbb{T}$, a cascade of zero-crossing can last for several successive instants of $\mathbb{T}$. Note that the synchronous interpretation coincides with that of Simulink (see discussion in Section 5.8) where zero-crossings have an immediate effect.

5.10 Conclusion

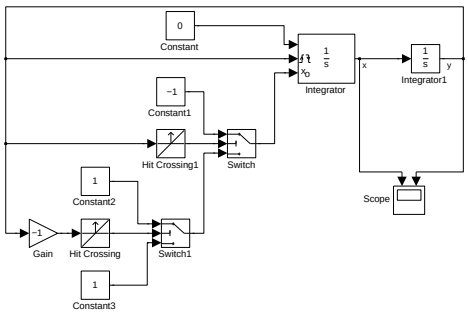
We have proposed a novel approach to the semantics of hybrid systems modelers. In doing so, we wanted:

1) To leave the choice of integration method unconstrained;
2) To ensure that hybrid systems are a conservative extension of discrete time systems;
3) To provide semantic support for:
   a) Statically analyzing and scheduling the actions triggered by cascaded zero-crossings;
   b) Separating discrete and continuous behaviours, and treating them both by combining existing ODE solvers with existing synchronous language compilers;
   c) Rejecting programs with causality circuits;
   d) Allowing for the use of several local solvers instead of a single, global one, with the objective of limiting side effects between non-interacting sub-systems, due to step size adjustments.

Achieving these objectives was made possible thanks to the use of non-standard analysis as a semantic domain. We believe that non-standard semantics is not a fancy thing for math addicts.

It is rather a very natural way of viewing continuous time and hybrid systems from the syntactic side, as is usually preferred by computer scientists. Our study of cascaded zero-crossings benefits greatly from the semantics. The non-standard semantics allowed for the rapid development of a prototype by combining an off-the-shelf ODE solver with an of-the-shelf synchronous language compiler.

ACKNOWLEDGEMENT *The authors are indebted to Ramine Nikoukhah and Sébastien Furic for detailed discussions regarding Modelica, and to Daniel Krob and Simon Bliudze for comments on their work.*



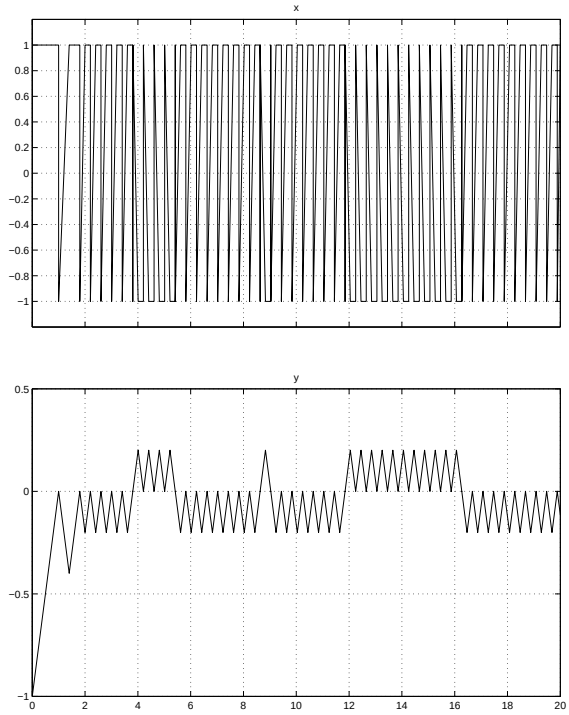


Figure 5.7: Example (5.2) in Simulink

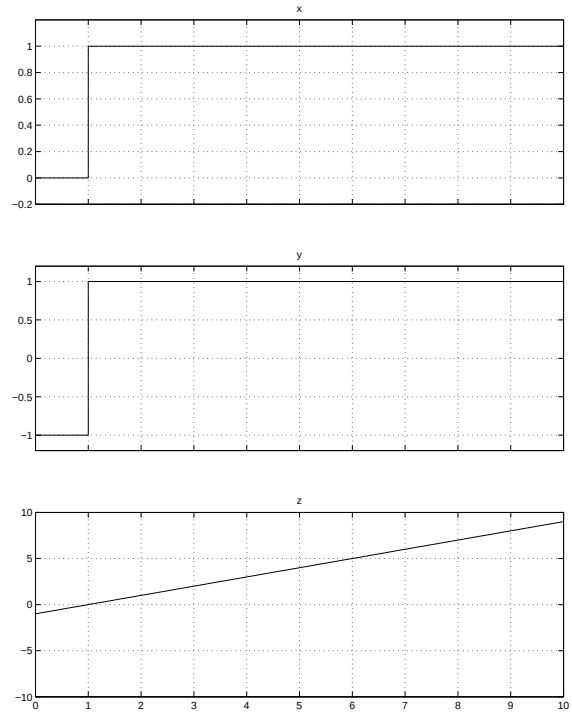


Figure 5.8: Example (5.3) in Simulink

statement	**non-standard semantics**	**constructive semantics**
$Eq_1 \quad y = f([x])$	$y_t = f([x_t])$	$[x] \lhd y$
$Eq_2 \quad y = \mathbf{last}\,(x)$	$y_t = x_{\bullet t}$	${}^\bullet x \lhd y$
$Eq_3 \quad y = \dot{x}$	$y_t = \frac{1}{\partial}(x_t - x_{\bullet t})$	$x \lhd y$
$Eq_4 \quad \zeta = \mathbf{up}(z)$	$\zeta_{t^\bullet} = [z_{\bullet t} \leq 0] \wedge [z_t > 0]$ ζ is discrete	$z \lhd \zeta^\bullet$ ζ is discrete
$Eq_5 \quad \dot{y} = x$ $\mathbf{init}\ y_0$ $\mathbf{reset}\ u$	τ_u discrete $t \in \tau \setminus \tau_u \Rightarrow y_t = y_{\bullet t} + \partial \times x_{\bullet t}$ $t \in \tau_u \Rightarrow y_t = u_t$	$\tau_u \lhd \quad y$ ${}^\bullet x \lhd \quad y\ \mathbf{init}\ y_0 \quad \lhd$ $y\ \mathbf{reset}\ u \quad \lhd y$
$Eq_6 \quad u = [v]$ $\mathbf{every}\ [\zeta]$ $\mathbf{init}\ u_0$	discrete $\tau_u = \bigcup_i \zeta_i$ $t < \min(\bigcup_i \zeta_i) \Rightarrow u_t = u_0$ $t \in \zeta_i \setminus (\bigcup_{j<i} \zeta_j) \Rightarrow u_t = v_{i,t}$	discrete $\tau_u = \bigcup_i \zeta_i$ $[\zeta] \lhd u$ $[v] \lhd u \quad \mathbf{every}\ [\zeta]$ $\mathbf{init}\ u_0 \quad \lhd u$
$Eq_7 \quad y = \mathbf{pre}\,(x)$ $\mathbf{init}\ y_0$	$\tau_y = \tau_x$ discrete $t < \min(\tau_y) \Rightarrow y_t = y_0$ $t \in \tau_y \Rightarrow y_t = x_{\bullet t}$	$\tau_y = \tau_x$ discrete $\tau_x \lhd y$ ${}^\bullet x \lhd y\ \mathbf{init}\ y_0 \lhd y$
$Eq_8 \quad S_1 \parallel S_2$ $S_1 = (X_1, \Sigma_1)$ $S_2 = (X_2, \Sigma_2)$	$\left(X, {\Sigma_1}^{\uparrow X} \cap {\Sigma_2}^{\uparrow X}\right),\ X = X_1 \cup X_2$	$[\![S_1]\!] \parallel [\![S_2]\!]$

Table 5.4: Non-standard semantics (mid column) and constructive semantics (right column) of SIMPLEHYBRID. Time t is universally quantified.

Part III

Synthesis and Control of Concurrent Systems

Chapter 6

Distributing finite automata through Petri net synthesis

Résumé : *La programmation des systèmes communicants est une activité délicate, demandant du programmeur des compétences spécifiques aux systèmes répartis et aux modèles de la concurrence. Il est donc souhaitable de pouvoir offrir aux concepteurs un modèle de programmation qui fasse abstraction de la communication. Ceci suppose bien entendu que des méthodes de synthèse de protocoles communicants puissent être utilisées et que celles-ci produisent des protocoles corrects, efficaces et economes en messages échangés.*

Les techniques de synthèse de réseaux de Petri permettent de transformer un modèle dans lequel la concurrence entre événements n'est pas explicite, en un modèle concurrent. Ces techniques sont intéressantes pour des applications aussi variées que la synthèse de circuits matériels asynchrones ou la construction de modèles à partir de traces d'exécution.

Ce chapitre, repris de [9], présente une spécialisation d'un algorithme de synthèse de réseaux de Petri généraux pour ne produire que des réseaux pouvant être facilement transformés en systèmes communicants : les réseaux distribuables. les réseaux distribuables forment une classe syntaxique de réseaux dont les places et transitions sont réparties sur un ensemble de sites, et qui ne permettent pas d'exprimer des propriétés de conflit entre deux sites distincts. L'algorithme de synthèse en temps polynomial repose sur la résolution dans les nombres rationnels de systèmes d'inéquations linéaires, les contraintes de distribution apparaissant comme des conditions de signe de certaines variables.

Une fois obtenu un réseau distribuable (si il en existe), il est aisé de transformer le réseau distribuable en processus communicants. L'intérêt de cet algorithme est que le problème de synthèse d'un système communicant est décomposé en trois étapes, la premère étant la transformation de la spécification de manière à ce qu'elle admette comme solution un réseau distribuable. Une fois cette première étape franchie, le calcul d'un réseau distribuable et sa transformation en processus

communicants est automatique. Cet algorithme a été mis en œuvre dans l'outil Synet [1].

6.1 Introduction

The *synthesis problem* for Petri nets is the question whether a given automaton (or a given language) on alphabet E is isomorphic to the state graph (or equal to the set of behaviours) of a Petri net to be discovered, with set of events identical to E. We will address this question for finite automata defined on an enriched alphabet (E, λ) where $\lambda : E \to \Lambda$ maps each event $e \in E$ to a *location* $\lambda(e) \in \Lambda$. Considering these automata as specifications of distributed systems, we search for realizing them by *distributable* nets [107], such that two transitions with different locations are never in conflict. A distributable net translates easily to an equivalent family of finite communicating automata which, plugged in at separate locations in an asynchronous network, provide the desired implementation. We give in this chapter the description of an algorithm for distributed net synthesis and report on a few case studies where it supplies an assistance for the engineering of distributed protocols. Our goal is to attract engineers to a technique which may yield unexpected but effective solutions to practical distribution problems, relying in a totally hidden way on linear algebra that often beats intuition. A few case studies in distributed net synthesis would suffice to make the point, but we prefer to give a thorough presentation of the algorithm, yet unpublished, in order to allow users try different implementations. The material of the chapter is mainly assembled from [9], preliminary studies ([12],[50]) and research reports ([10],[49]).

The problem of synthesizing nets equivalent to a given finite automaton was addressed first by Ehrenfeucht and Rozenberg, who showed in [88] and [89] how deciding on the feasability of this problem for elementary nets, using the crucial concept of *regions*. A (boolean) region in an automaton is a subset of states such that this set is entered or exited uniformly by all transitions with a common label (exactly as if it were the set of reachable cases of a net that hold a fixed condition or place). Regions may be interpreted as and give rise to *atomic* nets with a single place, filled by incoming transitions and emptied by outgoing transitions. By considering the finitely many different subsets of regions of a finite automaton, and gluing atomic nets on transitions, a finite number of nets may be derived in this way from a finite automaton, but it may well occur that none of them has a case graph isomorphic to the given automaton. Those automata which are isomorphic to reachable case graphs of elementary nets are characterized by two *separation axioms*, one expressing that any two different states are separated by some region containing exactly one of them, and the other expressing that for any event e, each state disabling e can be separated from all states enabling e by some region exited by e. When these axioms are satisfied, the automaton is isomorphic to the reachable case graph of the elementary net assembled from the whole set of regions (viewed as atomic subnets). The same holds of any smaller net assembled from a subset of regions large enough to witness the satisfaction of both separation axioms [84].

The concept of regions was extended next by Mukund [135], who defined regions modelling

1. http://www.irisa.fr/s4/tools/synet/

extensions of places in state graphs of general Petri nets with the step firing rule. Mukund's extended regions may again be interpreted as one-place Petri nets, with flow arcs now weighted by non negative integers. Using these (integral) regions, Mukund established an abstract correspondence (namely a co-reflection) between separated *step* automata and Petri nets, where separated step automata satisfy an adapted version of Ehrenfeucht and Rozenberg's separation axioms. The separation axioms served also (albeit with slightly different regions) to characterize up to isomorphism automata which may be realized by pure Petri nets with the sequential firing rule [34], and automata *with concurrency relation* which may be realized by general Petri nets [86]. The afore mentioned results do not directly entail the decidability of the Petri net realization problem for finite automata, since the set of integral regions of a finite automaton is infinite: contrary to the set of boolean regions, this set cannot be inspected exhaustively in a finite amount of time for stating validity or invalidity of the separation axioms. It was shown in [12] that the synthesis problem for *pure* Petri nets is nevertheless decidable, because the set of integral regions of a finite automaton forms a free module and one may actually compute a finite set of generators of this module. The decision was extended later on to general Petri nets, possibly accomodating *self-loops* [10]. In both cases, deciding on the synthesis problem for Petri nets (or on the net realization problem for finite automata) takes time polynomial in the size of automata.

In contrast, the synthesis problem for elementary nets is an NP-complete problem [13]. The jump of complexity may be explained as follows. Integral regions, considered as maps from states to non-negative integers, are stable under addition of maps, hence they are suited for linear reasoning. On the contrary, regions, considered as maps from states to $\mathbb{Z}/2\mathbb{Z}$, are not stable under addition of maps, hence they are suited only for (non-linear) combinatorial reasoning. This did not prevent practically efficient algorithms based on binary decision diagrams and graph traversal to be constructed and used for deriving elementary nets from large automata [72, 70]. The tool PETRIFY in which these algorithms are integrated shows convincing applications of net synthesis to asynchronous circuits [69]. Elementary net synthesis finds there a privileged field of application, where it brings in a new technology.

The potential fields of application of general Petri net synthesis have not been investigated to a comparable degree. Following the path shown in [158], applications may be found in supervisory control of discrete event systems, but the most critical issue of distributed and asynchronous control has not yet been considered at depth. More positively, the work presented in ([49], [50]) brings the elements of an emerging technology for distributing finite reactive automata. Successful experiments have been conducted on simple communication protocols using the tool SYNET which has the nice feature to accomodate distribution constraints on the nets it is able to synthesize from automata.

The rest of the chapter presents the principles of the synthesis of general Petri nets (section 2), a proof that the synthesis problem can be solved in polynomial time for finite automata (section 3), an adaptation of the synthesis algorithm to distributable nets (section 4), a translation of distributable nets to asynchronously communicating automata (section 5), two case studies in distributing reactive automata using SYNET (section 6), and a few conclusions (section 7).

6.2 The Petri Net Synthesis Problem

The purpose of the section is to give an axiomatic characterization of the subfamily of finite automata which are isomorphic to reachable state graphs of unlabelled Petri nets. This characterization is based on Ehrenfeucht and Rozenberg's separation axioms for elementary transition systems, adapted to general Petri nets through an adequate extension of the concept of regions. The extended regions we propose are similar in spirit to those considered by Mukund [135] and by Droste and Shortt [86].

Before stating the definition of regions, let us fix terminology and notations. We assume a finite set E of *events*. A (finite) *automaton* over E is an initialized transition system $A = (S, E, T, s_0)$ with a (finite) nonempty set of *states* S, a (possibly empty) set of *transitions* $T \subseteq S \times E \times S$, and an *initial state* $s_0 \in S$. For convenience, $s \stackrel{e}{\rightarrow} s'$ is an equivalent of $(s, e, s') \in T$, and $s \stackrel{e}{\rightarrow}$ and $s \stackrel{e}{\not\rightarrow}$ are respective abbreviations for $\exists s' \in S\ (s \stackrel{e}{\rightarrow} s')$ and $\forall s' \in S\ \neg(s \stackrel{e}{\rightarrow} s')$. An automaton A is *deterministic* if for any state $s \in S$ and for any event $e \in E$, $(s \stackrel{e}{\rightarrow} s') \wedge (s \stackrel{e}{\rightarrow} s'') \Rightarrow s' = s''$. An automaton A is *co-deterministic* if for any state $s \in S$ and for any event $e \in E$, $(s' \stackrel{e}{\rightarrow} s) \wedge (s'' \stackrel{e}{\rightarrow} s) \Rightarrow s' = s''$. On the one hand, the automata we consider are not always deterministic or co-deterministic, on the other hand, they are always *reachable*, i.e. such that $S = \{s \,|\, s_0 \stackrel{*}{\longrightarrow} s\}$ where $\stackrel{*}{\longrightarrow}$ is the reflexive and inductive closure of the unlabelled transition relation $\rightarrow = \cup \{ \stackrel{e}{\rightarrow} \,|\, e \in E\}$. An automaton is *event-reduced* if it is reachable and every event $e \in E$ occurs on at least one transition $(s, e, s') \in T$. We recall hereafter the definition of marked Petri nets and their sequential state graphs (see, e.g., [157]).

Definition 6.2.1 (Petri nets) *A (finite) Petri net is a triple $N = (P, E, F)$ where P and E are (finite) disjoint sets of* places *and* events*, and F is a function, $F : (P \times E) \cup (E \times P) \rightarrow \mathbb{N}$. The net is* pure *if $F(p, e) = 0$ or $F(e, p) = 0$ for every place p and for every event e; it is* impure *otherwise. A* marking *of N is a map $M : P \rightarrow \mathbb{N}$. An event e has* concession *at M if $M(p) \geq F(p, e)$ for every place $p \in P$. An event e which has concession at M may be* fired*, resulting in a transition $M\ [\ e >\ M'$ where M' is the marking such that $M'(p) = M(p) - F(p, e) + F(e, p)$ for every place $p \in P$.*

Definition 6.2.2 (Marked nets and state graphs) *A marked Petri net is a quadruple $\mathcal{N} = (P, E, F, M_0)$ where M_0 is a marking of the underlying net (P, E, F), called the* initial marking*. The marked net is* place simple *if it is never the case that $M_0(p) = M_0(p')$ for different places p and p' such that $[\, F(p, e) = F(p', e) \wedge F(e, p) = F(e, p')\,]$ for every event $e \in E$. The* reachable markings *of $\mathcal{N}$ are the markings $M : P \rightarrow \mathbb{N}$ such that $M_0\ [\ *>\ M$, where $[\ *>$ is the reflexive and transitive closure of the unlabelled transition relation of the underlying net (P, E, F). The sequential state graph of $\mathcal{N}$ is the automaton $\mathcal{N}^* = (RM(\mathcal{N}), E, T, M_0)$ where $RM(\mathcal{N})$ is the set of reachable markings of $\mathcal{N}$ and $T = \{M \stackrel{e}{\rightarrow} M' \,|\, M, M' \in RM(\mathcal{N}) \wedge M\ [\ e >\ M'\}$.*

From this definition, the sequential state graph of a marked Petri net is a deterministic, co-deterministic

and reachable automaton, but it is generally not finite. Nets considered in the sequel are place simple but not necessarily pure.

6.2.1 Regions

We come now to the definition of regions, which does not depend upon the finiteness of automata; still, we are mostly interested in regions of finite automata, for the axiomatic characterization given in Section 6.2.2 is valid only for finite automata.

Definition 6.2.3 (Regions) *A region of the automaton* $A = (S, E, T, s_0)$ *is a tuple* $(\sigma, {}^\bullet\eta, \eta^\bullet)$ *where* $\sigma : S \to \mathbb{N}$, ${}^\bullet\eta$ *and* $\eta^\bullet : E \to \mathbb{N}$ *are maps such that:*
(a) $s \xrightarrow{e} \quad \Rightarrow \quad \sigma(s) \geq {}^\bullet\eta(e)$ *and*
(b) $s \xrightarrow{e} s' \quad \Rightarrow \quad \sigma(s') = \sigma(s) - {}^\bullet\eta(e) + \eta^\bullet(e)$
A region $(\sigma, {}^\bullet\eta, \eta^\bullet)$ *is* pure *if* ${}^\bullet\eta(e) = 0$ *or* $\eta^\bullet(e) = 0$ *for every event* $e \in E$.

It is readily observed that every place p of a net $\mathcal{N} = (P, E, F, M_0)$ determines an associated region $(\sigma, {}^\bullet\eta, \eta^\bullet)$ of the sequential state graph $\mathcal{N}^*$, such that $\sigma(M) = M(p)$ for every reachable marking $M \in RM(\mathcal{N})$, and ${}^\bullet\eta(e) = F(p, e)$ and $\eta^\bullet(e) = F(e, p)$ for every event e.

Conversely, every region $p = (\sigma, {}^\bullet\eta, \eta^\bullet)$ of A determines an atomic net $\mathcal{N}' = (\{p\}, E, F', M_0')$, such that $F'(p, e) = {}^\bullet\eta(e)$, $F'(e, p) = \eta^\bullet(e)$, and $M_0'(p) = \sigma(s_0)$. In case when $A = \mathcal{N}^*$ and region p derives from a homonymic place p of $\mathcal{N}$, the atomic net $\mathcal{N}'$ is actually isomorphic to the atomic subnet of $\mathcal{N}$ with the unique place p.

Before investigating the structure of the set of regions of an automaton, let us introduce notations and terminology based on an analogy with electricity specific to *pure regions*. In this particular case, the map $\eta = \eta^\bullet - {}^\bullet\eta$ provides the same information as the pair of maps $({}^\bullet\eta, \eta^\bullet)$. The map η measures the variations of σ along the paths of the automaton as a function of the events that occur, since

$$s \xrightarrow{e} s' \Rightarrow \sigma(s') - \sigma(s) = \eta(e) \tag{6.1}$$

on account of Condition *(b)* in Def. 6.2.3. Thus, if the automaton is reachable, the map σ is totally determined from $\sigma(s_0)$ and η; and the following condition is satisfied for every event $e \in E$:

$$[s_1 \xrightarrow{e} s_1' \quad \wedge \quad s_2 \xrightarrow{e} s_2'] \quad \Rightarrow \quad \sigma(s_1') - \sigma(s_1) = \sigma(s_2') - \sigma(s_2) \tag{6.2}$$

Given a region $(\sigma, {}^\bullet\eta, \eta^\bullet)$ of an automaton A, one may look at A as the model of an electric circuit, where each transition $s \xrightarrow{e} s'$ represents a component of type e, with nominal *tension* $\eta(e)$, plugged in between nodes s and s'. The map σ may thus be seen as a distribution of *potential*. Conversely, a map $\sigma : S \to \mathbb{N}$ satisfying Condition 6.2 defines a distribution of potential that can be realized by connecting components with adequate tensions $\eta(e)$, induced from differences of potential $\sigma(s') - \sigma(s)$ uniformly attached to transitions $s \xrightarrow{e} s'$. It will be assumed from now on that A is an event-reduced automaton, hence the map $\eta = \eta^\bullet - {}^\bullet\eta$ may be derived from σ for every region of A. For clarity, let us turn this into a formal definition.

Definition 6.2.4 *Let $A = (S, E, T, s_0)$ be an event-reduced automaton. A map $\sigma : S \to \mathbb{N}$ satisfying Condition 6.2 is called a* distribution (of potential) *over the nodes of A. The* derived *map $\eta : E \to \mathbb{Z}$ such that σ and η satisfy together Condition 6.1 is called a* distribution (of tension) *over the events of A.*

Leading further the analogy, we will show that if η derives from σ then $\sigma(s) = \sigma(s_0) + \int_{s_0}^{s} \eta$ for every state s (where integration follows any path from s_0 to s in A) and that $\int_C \eta = 0$ along every cycle C of A. For precision, let us state a few definitions.

Definition 6.2.5 *The* underlying graph *of the automaton $A = (S, E, T, s_0)$ is the oriented multigraph $G(A) = (S, U, \partial^0, \partial^1)$ with components as follows.*
– S, the set of nodes*, is the set of states of A.*
– U, the set of arcs*, is the disjoint union of two copies of T, $U = T^+ \cup T^-$ where $\cdot^+$ and $\cdot^-$ are respective injections from T to U. Arcs $u \in T^+$ are* forward *transitions. Arcs $u \in T^-$ are* backward *transitions.*
– The source map *$\partial^0 : U \to S$ (resp. the* target map *$\partial^1 : U \to S$) is defined with $\partial^0(t^+) = s$ and $\partial^0(t^-) = s'$ for $t = (s \xrightarrow{e} s') \in T$ (resp. with $\partial^1(t^+) = s'$ and $\partial^1(t^-) = s$).*

Definition 6.2.6 *A* path *in A or in $G(A)$ is a non-empty sequence of arcs $P = u_1 \dots u_n$ such that $\partial^1(u_i) = \partial^0(u_{i+1})$ for all $i < n$. The* source *and* target *of path P are the respective nodes $\partial^0(P) = \partial^0(u_1)$ and $\partial^1(P) = \partial^1(u_n)$. The* reverse *of path P is the path $P^{-1} = {u_n}^{-1} \dots {u_1}^{-1}$ where ${u_i}^{-1} = {t_i}^-$ if $u_i = {t_i}^+$ and ${u_i}^{-1} = {t_i}^+$ if $u_i = {t_i}^-$. Path P is* elementary *if $\partial^0(u_i) \neq \partial^0(u_j)$ and $\partial^1(u_i) \neq \partial^1(u_j)$ for all $i \neq j$. Path P is a* cycle *if $\partial^1(P) = \partial^0(P)$.*

In the sequel, path and cycle will always be used to mean elementary path and elementary cycle.

Now for any map $\eta : E \to \mathbb{Z}$, let $\int_P \eta = \int_P^+ \eta - \int_P^- \eta$ where $\int_P^+ \eta$ is defined as the sum of $\eta \circ \ell(t_i)$ for forward transitions ${t_i}^+$ on path P and $\int_P^- \eta$ is defined as the sum of $\eta \circ \ell(t_i)$ for backward transitions ${t_i}^-$ on path P; let $\int_C \eta$ be defined similarly for a cycle C. It should thus be clear from conditions 6.1 and 6.2 that whenever $\sigma : S \to \mathbb{N}$ and $\eta : E \to \mathbb{Z}$ are compatible distributions of potential and of tension, the following identities do hold for every path P and for every cycle C:

$$\int_P \eta = \sigma(\partial^1(P)) - \sigma(\partial^0(P)) \tag{6.3}$$

$$\int_C \eta = 0 \tag{6.4}$$

This suggests the following alternative to Def. 6.2.4.

Definition 6.2.7 *A map $\eta : E \to \mathbb{Z}$ satisfying equation 6.4 for every cycle C of the automaton A is a* distribution (of tension) *over the events of A.*

A simple reasoning shows that the two definitions of distributions of tension given in Def. 6.2.4 and Def. 6.2.7 are equivalent for *finite* event-reduced automata. Suppose that $\int_C \eta = 0$ for every cycle C. It is easily seen that two paths P and P' such that both $\partial^0(P) = \partial^0(P')$ and $\partial^1(P) = \partial^1(P')$ can always be cut into finitely many slices P_i and P'_i such that $P_i = P'_i$ or $P_i(P'_i)^{-1}$ is a cycle. It follows from the hypothesis on η that $\int_{P_i} \eta = \int_{P'_i} \eta$ for all i, whence $\int_P \eta = \int_{P'} \eta$. Therefore, there is nothing ambiguous if we let $\int_{s'}^{s} \eta$ be defined as $\int_P \eta$ for any path P from s' to s. It should now appear that η must derive from some distribution of potential: the adequate distributions $\sigma : S \to \mathbb{N}$ are obtained by integrating η according to

$$\sigma(s) = \sigma(s_0) + \int_{s_0}^{s} \eta \tag{6.5}$$

The resulting distributions σ are defined up to an additive constant, since any value of $\sigma(s_0)$ greater than or equal to the opposite of $\int_{s_0}^{s} \eta$ for all s (and in particular for s_0) may be chosen (recall that S has been assumed finite).

Now the opposite of $\int_{s_0}^{s'} \eta$ is $\int_{s'}^{s_0} \eta$ and $\int_{s'}^{s_0} \eta + \int_{s_0}^{s} \eta$ equals $\int_{s'}^{s} \eta$. We may therefore sum up as follows.

Proposition 6.2.8 *The* pure *regions $(\sigma, {}^\bullet\eta, \eta^\bullet)$ of a finite event-reduced automaton A are in bijection with the pairs (η, k) such that $k \in \mathbb{N}$ and η satisfies equation 6.4 for every cycle C. The bijection is given by $\eta = \eta^\bullet - {}^\bullet\eta$ and $k = \mathbf{min}\{\sigma(s) \mid s \in S\}$. The reciprocal bijection is given by*

$$\begin{aligned} {}^\bullet\eta(e) &= \mathbf{max}\{0, -\eta(e)\} \\ \eta^\bullet(e) &= \mathbf{max}\{0, \eta(e)\} \\ \sigma(s) &= \mathbf{max}\{\textstyle\int_{s'}^{s} \eta \mid s' \in S\} + k \end{aligned}$$

We will now extend this characterization from pure regions to *arbitrary regions* of A.

Proposition 6.2.9 *The regions $(\sigma, {}^\bullet\eta, \eta^\bullet)$ of a finite event-reduced automaton A are in bijection with the triples (η, k, δ) such that $k \in \mathbb{N}$, η satisfies equation 6.4 for every cycle C, and $\delta : E \to \mathbb{N}$ is a map such that*

$$\delta(e) \leq k + \mathbf{min}\{0, \eta(e)\} + \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \xrightarrow{e}\} - \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \in S\} \tag{6.6}$$

The bijection is given by $\eta = \eta^\bullet - {}^\bullet\eta$, $k = \mathbf{min}\{\sigma(s) \mid s \in S\}$, and $\delta(e) = {}^\bullet\eta(e)$ if $\eta(e) \geq 0$, $\delta(e) = \eta^\bullet(e)$ otherwise. The reciprocal bijection is given by

$$\begin{aligned} {}^\bullet\eta(e) &= \delta(e) + \mathbf{max}\{0, -\eta(e)\} \\ \eta^\bullet(e) &= \delta(e) + \mathbf{max}\{0, \eta(e)\} \\ \sigma(s) &= \mathbf{max}\{\textstyle\int_{s'}^{s} \eta \mid s' \in S\} + k \end{aligned}$$

Proof: Let $(\sigma, {}^\bullet\eta, \eta^\bullet)$ be an arbitrary region of A. For every event $e \in E$, define:

$$\eta(e) = \eta^\bullet(e) - {}^\bullet\eta(e) \quad {}^\circ\eta(e) = \mathbf{max}\{0, -\eta(e)\} \quad \eta^\circ(e) = \mathbf{max}\{0, \eta(e)\}$$

From Def. 6.2.3, $(\sigma, {}^\circ\eta, \eta^\circ)$ is a pure region of A, hence there exists $k \in \mathbb{N}$ such that, for all $s \in S$, $\sigma(s) = \mathbf{max}\{\int_{s'}^{s} \eta \mid s' \in S\} + k$.
Observe that $\eta(e) = \eta^\circ(e) - {}^\circ\eta(e)$. Therefore, $\eta^\bullet(e) - \eta^\circ(e) = {}^\bullet\eta(e) - {}^\circ\eta(e)$, and if $\delta(e)$ denotes this difference then $\delta(e) \geq 0$, for on the one hand, $\eta(e) \geq 0$ entails ${}^\circ\eta(e) = 0$ and $\delta(e) = {}^\bullet\eta(e) - {}^\circ\eta(e) = {}^\bullet\eta(e)$, and on the other hand, $\eta(e) < 0$ entails ${}^\circ\eta(e) = -\eta(e)$ and $\delta(e) = {}^\bullet\eta(e) + \eta(e) = {}^\bullet\eta(e) + (\eta^\bullet(e) - {}^\bullet\eta(e)) = \eta^\bullet(e)$.
From condition (a) in Def. 6.2.3, ${}^\bullet\eta(e) \leq \sigma(s)$ whenever $s \xrightarrow{e}$, or yet equivalently ${}^\circ\eta(e) + \delta(e) \leq \mathbf{max}\{\int_{s'}^{s} \eta \mid s' \in S\} + k$.
Now $\delta(e)$ satisfies relation 6.6, establishing half of the proposition, since

$$\mathbf{max}\{\int_{s'}^{s} \eta \mid s' \in S\} = \int_{s_0}^{s} \eta - \mathbf{min}\{\int_{s_0}^{s'} \eta \mid s' \in S\}$$

In order to establish the other half, consider η, k, and δ satisfying the conditions of the proposition, and let σ, ${}^\bullet\eta$ and $\eta^\bullet$ be the maps defined by the correspondence. We will show that $(\sigma, {}^\bullet\eta, \eta^\bullet)$ is a region of A. From Prop. 6.2.8, the map σ is a distribution of potential over the states of A, and η is the derived distribution of tension over the events of A. Since $\eta(e) = \eta^\bullet(e) - {}^\bullet\eta(e)$ for all e, condition (b) in Def. 6.2.3 is satisfied. Now, condition (a) in Def. 6.2.3 may be rewritten into the implication

$$s \xrightarrow{e} \quad \Rightarrow \quad \delta(e) - \mathbf{min}\{0, \eta(e)\} \leq k + \int_{s_0}^{s} \eta - \mathbf{min}\{\int_{s_0}^{s'} \eta \mid s' \in S\}$$

which follows directly from relation 6.6. ∎

The reader may observe that one shifts from pure regions to impure regions by adding simultaneously some $\delta(e)$ satisfying relation 6.6 to ${}^\bullet\eta(e)$ and to $\eta^\bullet(e)$ for each event e. Conversely, one returns from general regions to pure regions by subtracting simultaneously from ${}^\bullet\eta(e)$ and from $\eta^\bullet(e)$ the maximal $\delta(e)$ satisfying relation 6.6 for each event e.

Definition 6.2.10 *Let $R_{\eta,k,\delta}$ denote the region of A fixed by the correspondence given in Prop. 6.2.9. In case when $k = 0$ and δ takes for each e the maximal value allowed by relation 6.6, the region $R_{\eta,k,\delta}$ is said to be* canonical *and it is given the simpler notation R_η. Thus $R_\eta = (\sigma, {}^\bullet\eta, \eta^\bullet)$ is defined by*

$$\begin{array}{rcl} \sigma(s) & = & \sigma(s_0) + \int_{s_0}^{s} \eta \qquad \textit{where} \qquad \sigma(s_0) = \mathbf{max}\{\int_{s}^{s_0} \eta \mid s \in S\} \\ {}^\bullet\eta(e) & = & \sigma(s_0) + \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \xrightarrow{e}\} = \mathbf{min}\{\sigma(s) \mid s \xrightarrow{e}\} \\ \eta^\bullet(e) & = & \eta(e) + {}^\bullet\eta(e) \end{array}$$

Observe that R_η may be computed from η using time polynomial in $|S|$ and $|E|$, which are both bounded by $|T|+1$ (since A is reachable and event reduced). Canonical regions will play a major role in the sequel.

6.2.2 Representation Theorem

We come now to the logical laws explaining the structure of the sequential state graph of a net in terms of the regions that derive from its places.

Definition 6.2.11 (Separated automaton) *An automaton $A = (S, E, T, s_0)$ is* separated *if and only if the following axioms hold for all states $s, s' \in S$ and for every event $e \in E$:*

(SSA) $s \neq s' \Rightarrow \sigma(s) \neq \sigma(s')$ *for some region $R = (\sigma, {}^\bullet\eta, \eta^\bullet)$*
- R solves the states separation *problem at (s, s') -*

(ESSA) $s \overset{e}{\not\rightarrow} \Rightarrow \sigma(s) < {}^\bullet\eta(e)$ *for some region $R = (\sigma, {}^\bullet\eta, \eta^\bullet)$*
- R solves the event/state separation *problem at (s, e) -*

A subset of regions with enough elements to witness the satisfaction of both axioms is called an admissible *subset of regions.*

Let us explain the motivations under the definition of separated automata. If one observes the sequential state graph $\mathcal{N}^*$ of a marked Petri net $\mathcal{N}$, one may remark that it is a separated automaton: on the one hand, if markings M and M' are different, $M(p) \neq M'(p)$ for some place p, and the region that derives from p solves the states separation problem at (M, M'); on the other hand, if event e cannot be fired at M, $M(p) < F(p, e)$ for some place p, and the region that derives from p solves the event / state separation problem at (M, e). Thus, the regions of $\mathcal{N}^*$ that derive from the places of $\mathcal{N}$ form an admissible subset of regions, and axioms SSA and $ESSA$ are valid in every automaton isomorphic to the sequential state graph of a marked Petri net. Now reverting the analysis, consider a separated automaton A. Axiom SSA means that states of A may be represented injectively as markings of a net $\mathcal{N}$, whose places p are defined from regions $(\sigma, {}^\bullet\eta, \eta^\bullet)$ as one may expect: state s is mapped to marking M such that $M(p) = \sigma(s)$. In the considered representation, the contraposed version of axiom $ESSA$ means that an event e which has concession at marking M in $\mathcal{N}$ is necessarily enabled at s in A. As $\sigma(s)$ evolves through a transition $s \overset{e}{\rightarrow} s'$ in A in the same way as $M(p)$ evolves through a transition $M\,[\,e\rangle\,M'$ in $\mathcal{N}$, A and $\mathcal{N}^*$ cannot differ that much. We will show that the separation axioms characterize actually the variety of *finite event reduced* automata isomorphic to sequential state graphs of marked Petri nets. Recall that in an event-reduced automaton $A = (S, E, T, s_0)$, each event $e \in E$ labels at least one transition in T.

Lemma 6.2.12 *A separated automaton is deterministic and co-deterministic.*

Proof: Let $A = (S, E, T, s_0)$ be a separated automaton where $(s \overset{e}{\rightarrow} s')$ and $(s \overset{e}{\rightarrow} s'')$. Then for any region $(\sigma, {}^\bullet\eta, \eta^\bullet)$ of A, $\sigma(s') = \sigma(s) + \eta(e) = \sigma(s'')$ with $\eta = \eta^\bullet - {}^\bullet\eta$. Therefore, $s' = s''$

follows by axiom SSA. Co-determinism is shown in a similar way. ∎

Definition 6.2.13 (Net synthesized from a finite subset of regions)
Given an event reduced automaton $A = (S, E, T, s_0)$ and a finite subset P of regions of A, with typical region $p = (\sigma, {}^\bullet\eta, \eta^\bullet)$, let $A^*[P] = (P, E, F, M_0)$ be the marked Petri net such that $F(p, e) = {}^\bullet\eta(e)$, $F(e, p) = \eta^\bullet(e)$, and $M_0(p) = \sigma(s_0)$.

Theorem 6.2.14 (Characterization result) *A finite event reduced automaton is isomorphic to the sequential state graph of some marked Petri net if and only if it satisfies the separation axioms* **SSA** *and* **ESSA**. *A finite event reduced and separated automaton A is actually isomorphic to the sequential state graph of $A^*[P]$ for any finite admissible subset P of regions of A.*

Proof: Let $A = (S, E, T, s_0)$ be a finite event reduced automaton. If this automaton is separated, it follows from finiteness that one can extract a finite admissible subset P from its infinite set of regions. It suffices to show that A is then isomorphic to the sequential state graph of $\mathcal{N} = A^*[P]$. Let $\mathcal{N} = (P, E, F, M_0)$ then by definition $\mathcal{N}^* = (RM(\mathcal{N}), E, T, M_0)$ where $RM(\mathcal{N})$ is the set of reachable markings of $\mathcal{N}$ and $\forall M, M' \in RM(\mathcal{N})$, $(M \xrightarrow{e} M') \in T$ if and only if $M\,[\,e > M'$ in $\mathcal{N}$. Define $\sim \subseteq (S \times RM(\mathcal{N}))$ such that $s \sim M$ if and only if for all regions $p \in P$: $p = (\sigma, {}^\bullet\eta, \eta^\bullet) \Rightarrow M(p) = \sigma(s)$. From this definition, $\sim$ is a functional relation. From the axiom SSA and the assumption that P is an admissible subset of regions of A, $\sim$ is an injective relation. We will show that $\sim$ is an isomorphism of automata. Observing on the one hand that $s_0 \sim M_0$, and on the other hand that both A and $\mathcal{N}^*$ are deterministic and reachable, it suffices to establish the following transfer property: if $s \sim M$ then for any $e \in E$, $s \xrightarrow{e}$ in A if and only if $M\,[e >$ in $\mathcal{N}$, and then $s' \sim M'$ where $s \xrightarrow{e} s'$ and $M\,[e > M'$. So let $s \sim M$. Suppose $s \xrightarrow{e}$ in A, then for every place $p \in P$: $p = (\sigma, {}^\bullet\eta, \eta^\bullet) \Rightarrow \sigma(s) \geq {}^\bullet\eta(e)$ by definition of regions and hence $M(p) \geq F(p, e)$, showing that $M\,[e >$ in $\mathcal{N}$. Suppose $M\,[e >$ in $\mathcal{N}$ and assume for contradiction that $s \not\xrightarrow{e}$ in A. From the axiom $ESSA$ and the assumption that P is an admissible subset of regions of A, there exists in P some region $p = (\sigma, {}^\bullet\eta, \eta^\bullet)$ such that $\sigma(s) < {}^\bullet\eta(e)$, hence $M(p) < F(p, e)$ contradicting $M\,[e >$. Suppose finally that $s \xrightarrow{e} s'$ in A and $M\,[e > M'$ in $\mathcal{N}$. We should prove that for any place $p \in P$: $p = (\sigma, {}^\bullet\eta, \eta^\bullet) \Rightarrow M'(p) = \sigma(s')$. From the sequential firing rule, $M'(p) = M(p) - F(p, e) + F(e, p)$. From the definition of the net $\mathcal{N} = A^*[P]$, $F(p, e) = {}^\bullet\eta(e)$ and $F(e, p) = \eta^\bullet(e)$. From $s \sim M$, $M(p) = \sigma(s)$. From the definition of regions, $(s \xrightarrow{e} s') \Rightarrow \sigma(s) - {}^\bullet\eta(e) + \eta^\bullet(e) = \sigma(s')$. Altogether, $M'(p) = M(p) - F(p, e) + F(e, p) = \sigma(s) - {}^\bullet\eta(e) + \eta^\bullet(e) = \sigma(s')$, and $s' \sim M'$. ∎

Definition 6.2.15 *Given an automaton A, a subset $\mathcal{R}$ of regions of A is* logically complete *if all instances in A of the separation problems solved by regions of A are equally solved by regions in $\mathcal{R}$.*

Beware of that logically complete sets of regions are generally not admissible. To explain this, consider for instance the automaton $A = (S, E, T, s_0)$ defined with $S = \{s_0, s_1\}$, $E = \{e\}$, and $T = \{s_0 \xrightarrow{e} s_1, s_1 \xrightarrow{e} s_0\}$. This automaton is not isomorphic to the sequential state graph of any marked Petri net: states s_0 and s_1 cannot be separated by any region of A. Thus the set of all regions of A is not admissible, but it is logically complete by trivial application of the definition. Another, more conclusive, example of a logically complete set of regions is, in any automaton, the subset of all regions $(\sigma, {}^\bullet\eta, \eta^\bullet)$ such that ${}^\bullet\eta(e) \neq 0$ and $\eta^\bullet(e) \neq 0$ may hold simultaneously for *at most one* event e: whenever two states s and s' are separated by a region, they are separated as well by a pure region; whenever a region solves the event / state separation problem at (s, e), the problem may be solved as well by a region such that ${}^\bullet\eta(e') = 0$ or $\eta^\bullet(e') = 0$ for every event $e' \neq e$. The point is that, for deciding on separation, it suffices to search for admissible subsets of some logically complete set of regions. This is the common principle of all net synthesis algorithms known so far, differing one from another on the choice of the logically complete set of regions. This set must be finite, or at least finitely generated, in order to support effective algorithms. The algorithm defined in the next section uses the fact that the set of *canonical* regions is logically complete.

6.3 A Polynomial Time Synthesis Algorithm

We describe in this section an algorithm for the synthesis of Petri nets from finite automata taking time polynomial in the size of the automata, thus establishing the following:

Theorem 6.3.1 *Deciding whether a finite event-reduced automaton is isomorphic to the sequential state graph of some marked Petri net, and producing then a minimal Petri net realizing the automaton, takes time polynomial in the number of transitions of the automaton.*

The algorithm stems from the observation that the set of canonical regions is logically complete.

Proposition 6.3.2 *In any finite event-reduced automaton A, the set of canonical regions R_η is logically complete, and a subset of canonical regions $\{R_\eta \mid \eta \in \mathcal{H}\}$ is admissible if and only if:*

i) $\quad s \neq s' \quad \Rightarrow \quad \exists \eta \in \mathcal{H} \quad \int_{s_0}^{s} \eta \;\neq\; \int_{s_0}^{s'} \eta,$

ii) $\quad s' \overset{e}{\not\rightarrow} \quad \Rightarrow \quad \exists \eta \in \mathcal{H} \quad \int_{s_0}^{s'} \eta \;<\; \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \xrightarrow{e}\}.$

Proof: By definition, a region $(\sigma, {}^\bullet\eta, \eta^\bullet)$ solves the states separation problem at (s, s') if and only if $\sigma(s) = \sigma(s_0) + \int_{s_0}^{s} \eta \neq \sigma(s_0) + \int_{s_0}^{s'} \eta = \sigma(s')$ where $\eta = \eta^\bullet - {}^\bullet\eta$. The separation of s from s' by the canonical region R_η is expressed by an identical condition, and this condition holds if and only if $\int_{s_0}^{s} \eta \;\neq\; \int_{s_0}^{s'} \eta$.

By definition, a region $(\sigma, {}^\bullet\eta, \eta^\bullet)$ solves the event/state separation problem at (s', e) if and only if $\sigma(s') = \sigma(s_0) + \int_{s_0}^{s'} \eta < {}^\bullet\eta(e)$ where $\eta = \eta^\bullet - {}^\bullet\eta$. Now by Prop. 6.2.9, $(\sigma, {}^\bullet\eta, \eta^\bullet) = R_{\eta,k,\delta}$ for some $k \in \mathbb{N}$ and $\delta : E \to \mathbb{N}$ such that

${}^\bullet\eta(e) = \delta(e) + \mathbf{max}\{0, -\eta(e)\}$ and

${}^\bullet\eta(e) \leq k + \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \xrightarrow{e}\} - \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \in S\}$.
As $\sigma(s_0) = k - \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \in S\}$ it comes that ${}^\bullet\eta(e) \leq \sigma(s_0) + \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \xrightarrow{e}\}$.
To sum up, if the considered region $R_{\eta,k,\delta}$ separates s' from e, then necessarily $\int_{s_0}^{s'} \eta \ < \ \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \xrightarrow{e}\}$. By the definition of separation and Prop. 6.2.8, the canonical region R_η defined by $\eta = \eta^\bullet - {}^\bullet\eta$ separates s' from e if and only if $\sigma(s_0) + \int_{s_0}^{s'} \eta \ < \ \sigma(s_0) + \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \xrightarrow{e}\}$ and this relation holds if and only if $\int_{s_0}^{s'} \eta \ < \ \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \xrightarrow{e}\}$. ∎

The proof of Theo. 6.3.1 proceeds in two stages, described in separate subsections. We compute first a finite set of maps η generating all distributions of tension as linear combinations. We then use this finite presentation to check the separation conditions of Prop. 6.3.2. We are thus looking for an *admissible* family of distributions, containing enough elements to witness the satisfaction of both separation axioms. By Prop. 6.3.2, every admissible family of distributions $\mathcal{H}$ induces actually an admissible family of regions $\{R_\eta \mid \eta \in \mathcal{H}\}$ (where R_η is computed from η in polynomial time).

6.3.1 Computing Tensions

From now on, $A = (S, E, T, s_0)$ is a fixed automaton, finite and event-reduced.

Notation 6.3.3 *Let $\mathbb{Z}[E]$ denote the set of maps $u : E \to \mathbb{Z}$ equipped with addition of maps and multiplication of maps by integers. Each map $u : E \to \mathbb{Z}$ may thus be written as a linear combination $\sum_{e \in E} u(e) \times \bar{e}$, where $\bar{e} : E \to \mathbb{Z}$ is the map $\bar{e}(e') = 1$ if $e = e'$, 0 otherwise. For instance, $u = a - 2b$ is the map $u(a) = 1$, $u(b) = -2$, and $u(e) = 0$ for $e \notin \{a, b\}$. Alternatively, a map $u : E \to \mathbb{Z}$ may be seen as an E-vector with entries $u(e) \in \mathbb{Z}$. The scalar product of u and v in $\mathbb{Z}[E]$ is the integer $u \cdot v = \sum_{e \in E} u(e) \cdot v(e)$.*

Definition 6.3.4 *The* Parikh image *of a path P (of the automaton A) is the map $\psi(P) : E \to \mathbb{Z}$ such that $\psi(P)(e) = \int_P \bar{e}$. The Parikh image of a cycle C is the map $\psi(C) : E \to \mathbb{Z}$ such that $\psi(C)(e) = \int_C \bar{e}$.*

Example 6.3.5 *The Parikh image of the cycle C going through states s_0, s_3, s_5, s_2, s_4, s_0 in the automaton of Fig. 6.1 is $\psi(C) = -a + a' + a + b' + a' = 2a' + b'$*

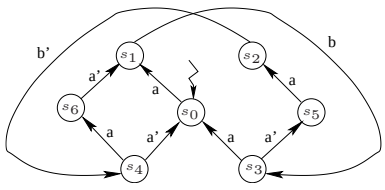


Figure 6.1: an automaton

Proposition 6.3.6 *A map $\eta \in \mathbb{Z}[E]$ defines a distribution of tension over the events of A if and only if $\eta \cdot \psi(C) = 0$ for every cycle C of A.*

Proof: From Def. 6.2.7, η represents a distribution of tension over the events of A if and only if $\int_C \eta = 0$ for every cycle C. For each $e \in E$, define $\eta_e : E \to \mathbb{Z}$ such that $\eta_e(e') = \eta(e)$ if $e' = e$, 0 otherwise. Now $\int_C \eta = \sum_{e \in E} (\int_C \eta_e) = \sum_{e \in E} (\eta(e) \times \psi(C)(e)) = \eta \cdot \psi(C)$. ∎

Corollary 6.3.7 *The maps $\eta \in \mathbb{Z}[E]$ that define distributions of tension over the events of A are the solutions of a linear system $M \cdot \eta = \mathbf{0}$ where M is an integral matrix whose rows are all Parikh images of (elementary) cycles of A.*

As the number of (elementary) cycles of A is finite, the matrix M of the above corollary is well defined. At this stage, the classical algorithm of von zur Gathen and Sieveking (see [163]) may be employed to compute (using time polynomial in the size of M) a finite set of solutions $\{\eta_1, \ldots, \eta_k\}$ of $M \cdot \eta = \mathbf{0}$ such that every solution of this system writes in a unique way as a linear combination $z_1\eta_1 + \ldots + z_k\eta_k$ with integer coefficients $z_i \in \mathbb{Z}$ (and any such combination yields a solution). Distributions of tension form therefore a *freely generated* integer module (see [131] for a definition). Note that $k \leq |E|$ (since generators η_i are linearly independent) and hence $k \leq |T|$ (since $|E| \leq |T|$ by the hypothesis of event reducedness).

We have not paid much attention to algorithmic complexity in the above. As the number of (elementary) cycles of a finite automaton is not polynomial in the number of transitions, the situation is problematic: one cannot set any bound polynomial in $|T|$ on the number of rows of M (whose number of columns is bounded by $|T|$ as A is event-reduced). Fortunately, a standard result of applied graph theory (see *e.g.* [66] or [96]) tells us that one needs not consider all cycles: it suffices to take as rows of M the Parikh images of $|T|-|S|+1$ *fundamental* cycles of A (see Def. 6.3.8 below).

Let us recall briefly this result. An elementary cycle of A, $C = u_1 \ldots u_n$, may be represented by a map $C : T \to \{-1, 0, 1\}$ as follows: if $C = t^+t^-$ or $C = t^-t^+$ for some transition t, then let $C(t) = 0$ for all $t \in T$, else, for all $t \in T$, let $C(t) = +1$ if $u_i = t^+$ for some i, $C(t) = -1$ if $u_i = t^-$ for some i, and $C(t) = 0$ in any other case. The result is the following: every *spanning tree* of A (see Def. 6.3.8 below) determines a family of $\beta = |T| - |S| + 1$ cycles of A, let $\{C_1, \ldots, C_\beta\}$, such that any cycle of A writes as a linear combination $C = \sum_{j=1}^{\beta} x_j \times C_j$ with coefficients $x_j \in \{-1, 0, 1\}$. Thus, for any map $\eta : E \to \mathbb{Z}$:

$$\int_C \eta = \sum_{j=1}^{\beta} x_j \times \left(\int_{C_j} \eta \right)$$

As $\int_C \eta = \eta \cdot \psi(C)$ for every cycle C (see the proof of Prop. 6.3.6), it follows that η defines a distribution of tension over the events of A if and only if $M' \cdot \eta = \mathbf{0}$ where M' is the integral matrix

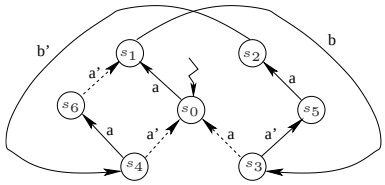


Figure 6.2: an automaton with one of its spanning trees (in solid lines)

whose rows are the Parikh images of the fundamental cycles $C_1 \ldots C_\beta$. Therefore, computing a set of generating distributions $\{\eta_1, \ldots, \eta_k\}$ takes time polynomial in the number of transitions $|T|$. It remains to give a precise definition of spanning trees and fundamental cycles, and to show an example.

Definition 6.3.8 *Given a finite reachable automaton $A = (S, E, T, s_0)$, a* spanning tree *is a reachable automaton $B = (S, E, T', s_0)$ with an identical set of states, such that $T' \subseteq T$ and all elementary cycles of B have form $C = t^+t^-$ or $C = t^-t^+$ (hence B is a tree). The $|T| - |S| + 1$ transitions $t \in T \setminus T'$ are called* chords. *For each chord t, there exists a unique cycle C_t such that $C_t(t) = 1$ and $C_t(t') = 0$ for every chord $t' \neq t$ (thus $C_t(t') \neq 0 \;\Rightarrow\; t' = t \vee t' \in T'$). The* fundamental cycles *of A are the cycles C_t defined from the chords $t \in T \setminus T'$.*

Note that *loops* $s \xrightarrow{e} s$ are special cycles, which appear as fundamental cycles under any choice of the spanning tree B. Note also that for each spanning tree B and for each state $s \neq s_0$ there exists a unique path from s_0 to s in B, denoted by P_s (hence, P_s is also a path from s_0 to s in A). Abusing the notations, let P_{s_0} denote the subgraph of A with the unique node s_0 and with the Parikh image $\psi(P_{s_0}) = \mathbf{0}$. Thus, for any state s and for any map $\eta \in \mathbb{Z}[E]$:

$$\int_{P_s} \eta \;=\; \sum_{e \in E} \left(\int_{P_s} \eta_e \right) = \sum_{e \in E} (\eta(e) \times \psi(P_s)(e)) = \eta \cdot \psi(P_s)$$

Example 6.3.9 *The Parikh images of the fundamental cycles defined by the spanning tree indicated in solid lines in Fig. 6.2 are $2a + b$ and $2a + 2a' + b + b'$. Thus, $\eta \in \mathbb{Z}[E]$ is a distribution of tension if and only if $2\eta(a) + \eta(b) = 0$ and $2\eta(a') + \eta(b') = 0$. The distributions of tension are therefore generated by $\eta_1 = a - 2b$ and $\eta_2 = a' - 2b'$.*

From now on, P_s is the path from s_0 to s in a fixed spanning tree, and $\{\eta_1, \ldots, \eta_k\}$ is a fixed set of generators for distributions of tension.

6.3.2 Solving the Separation Problems

Deciding upon separation of A comprises two subproblems, since two axioms must be satisfied. We consider first the states separation axiom SSA which is the easier one to check. This axiom is

Table 6.1

$\psi(P_{s_i})$	s_0	s_1	s_2	s_3	s_4	s_5	s_6
a	0	1	2	1	2	1	3
b	0	0	1	1	1	1	1
a'	0	0	1	0	1	1	1
b'	0	0	0	0	1	0	1

Table 6.2

$\eta_j \cdot \psi(P_{s_i})$	s_0	s_1	s_2	s_3	s_4	s_5	s_6
η_1	0	1	0	-1	0	-1	1
η_2	0	0	1	0	-1	1	-1

valid in A if each pair of distinct states s and s' is *separated* by a distribution of tension η such that $\int_{s_0}^{s} \eta \neq \int_{s_0}^{s'} \eta$. As $\int_{s_0}^{s} \eta = \eta \cdot \psi(P_s)$ and thus depends linearly on η, s and s' are separated if and only if they are separated by some distribution in the generating set $\{\eta_1, \ldots, \eta_k\}$. In order to check this, it suffices to construct a table as follows: rows are indexed by generators $\eta_i \in \{\eta_1, \ldots, \eta_k\}$, columns are indexed by states $s \in S$, and the contents of each entry (η_i, s) is the scalar product $\eta_i \cdot \psi(P_s)$. The axiom SSA is satisfied if and only if the columns of the table are all different (hence the distributions $\{\eta_1, \ldots, \eta_k\}$ form an admissible set w.r.t. state separation). In the converse case, the Petri net synthesis problem has no solution for A. As $|S| \leq |T|+1$ (for all states are reachable), $k \leq |T|$, and there exist $|S| \times (|S|-1)/2$ pairs of distinct states, checking states separation takes time polynomial in $|T|$.

Example 6.3.10 *The Parikh images $\psi(P_s)$ of the paths from s_0 to s in the spanning tree of the automaton shown in Fig. 6.2 are shown in Table 6.1. The scalar products $\eta_j \cdot \psi(P_{s_i})$, where $\eta_1 = a - 2b$ and $\eta_2 = a' - 2b'$, are shown in Table 6.2. As all columns are different, the states separation axiom SSA is satisfied. Thus, the family of distributions $\{\eta_1, \eta_2\}$ is admissible w.r.t. state separation.*

Let us consider now the event/state separation axiom $ESSA$. This axiom is valid in A if each pair (s', e) made of a state s' and an event e disabled at s' is *separated* by a distribution (of tension) η such that $\int_{s_0}^{s'} \eta \; < \; \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \stackrel{e}{\rightarrow}\}$.

Define $\beta_i\,(s', s) = \eta_i \cdot \psi(P_{s'}) - \eta_i \cdot \psi(P_s)$ for $i \in \{1, \ldots, k\}$ and $s \in S$, and set $\eta = \sum_{i=1}^{k} z_i \times \eta_i$. The question is to decide whether one can find $z_i \in \mathbb{Z}$ solving the system of linear homogeneous

inequalities

$$\left\{ \sum_{i=1}^{k} \beta_i\,(s', s) \times z_i < 0 \;\middle|\; s \xrightarrow{e} \right\} \tag{6.7}$$

The constants $\beta_i\,(s', s)$ may be computed in time polynomial in $|T|$. The number of the inequalities and the number of the unknown z_i are bounded by polynomials in $|T|$. Moreover, there are less than $|S| \times |E|$ instances of the event/state separation problem. Therefore, if (6.7) may be solved or proved unfeasible using time polynomial in the number of inequalities and in the number of the unknown then event/state separation may also be decided in polynomial time.

Now (6.7) is a *homogeneous* system of linear inequalities, hence it has an integral solution (in $\mathbb{Z}^k$) if and only if it has a rational solution (in$\mathbb{Q}^k$), and the integral solutions are the integer multiples of the rational solutions. Khachiyan's method of ellipsoids (see [163]) may therefore be used to decide on the feasability of (6.7) and to compute a solution, if it exists, in polynomial time.

By collecting the distributions $\eta = \sum_{i=1}^{k} z_i \times \eta_i$ that result from the solutions of (6.7) for all instances of $ESSA$, one obtains from $\{\eta_1, \ldots, \eta_k\}$ new distributions $\{\eta_{k+1}, \ldots, \eta_l\}$ which form an admissible set w.r.t. event/state separation.

Example 6.3.11 *In our running example, the homogeneous system of linear inequalities that express event/state separation at* (s_1, a) *is the following:*

s	$\sum_i \beta_i(s_1, s) \times z_i < 0$
s_0	$z_1 < 0$
s_3	$2z_1 < 0$
s_4	$z_1 + z_2 < 0$
s_5	$2z_1 - z_2 < 0$

This system, amounting to $z_1 < 0$ *and* $2z_1 < z_2 < -z_1$*, is solvable. A solution is for instance* $z_1 = -2$ *and* $z_2 = -3$*. This solution determines a distribution of tension* $\eta_3 = -2\eta_1 - 3\eta_2$ *given by the map* $\eta_3 = -2a + 4b - 3a' + 6b'$*. The canonical region* R_{η_3} *derived from* η_3 *according to Def. 6.2.10 separates* a *from* s_1 *as desired. The maps* $\{\eta_3, \eta_4, \eta_5\}$*, where* $\eta_4 = 2a - 4b + 3a' - 6b'$ *and* $\eta_5 = -a + 2b$*, form an admissible set w.r.t. event/state separation.*

Summary The separation axioms may be checked within time polynomial in the number of transitions, yielding an admissible set of distributions $\{\eta_1, \ldots, \eta_l\}$. An admissible set of canonical regions $P = \{R\eta_1, \ldots, R\eta_l\}$ derives according to Def. 6.2.10. A is then isomorphic to the sequential state graph of $A^*[P]$ (Def. 6.2.13). A minimal admissible subset P' may be extracted from P using time polynomial in $|T|$, providing a minimal net realization $A^*[P']$ of A. Theo. 6.3.1 is

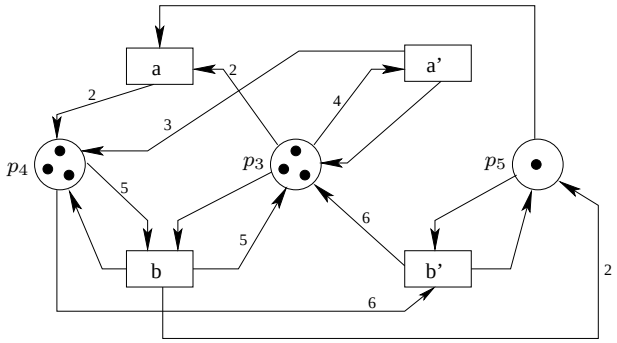


Figure 6.3: net synthesized from canonical regions

therefore proved.

Let us add one comment. When we call $A^*[P']$ a minimal realization of A, we mean that no place can be suppressed while keeping a sequential state graph isomorphic to A; we do *not* mean that $|P'|$ is the minimal number of places needed for realizing A.

Example 6.3.12 *In the automaton of Fig. 6.2, all instances of the state separation problem, solved either by η_1 or by η_2 as we saw, are solved as well by η_3. The canonical regions $\{R_{\eta_3}, R_{\eta_4}, R_{\eta_5}\}$ which derive from $\eta_3 = -2a + 4b - 3a' + 6b'$, $\eta_4 = 2a - 4b + 3a' - 6b'$, and $\eta_5 = -a + 2b$ form therefore an admissible set of regions. Applying Def. 6.2.10 to produce canonical regions $R_\eta = (\sigma, {}^\bullet\eta, \eta^\bullet)$ from distributions of tension η, and using relations $F(p,e) = {}^\bullet\eta(e)$, $F(e,p) = \eta^\bullet(e)$, and $M_0(p) = \sigma(s_0)$ to derine places from regions R_η, one obtains the places p_3, p_4, and p_5 of the net displayed in Fig. 6.3. The sequential state graph of this net is actually isomorphic to the automaton of Fig. 6.1.*

6.4 Adding Distribution Constraints

In order to show the principles of the application of net synthesis to the distribution of finite reactive automata, we introduce a special class of labelled Petri nets, called *distributable* nets. The label of an event indicates the location of its host in a network of automata which communicate by asynchronous message passing. The goal of distributable nets is to avoid distributed conflicts. This discipline is enforced by imposing common locations on conflicting events. The rest of the section deals with the synthesis of distributable nets from finite automata with fixed locations of events. This extended realization problem can still be solved in polynomial time. An adapted synthesis algorithm has been implemented in SYNET.

Definition 6.4.1 (Distributable net system) *A* distributable *net system with set of* locations Λ *is a quintuple $\mathcal{N} = (P, E, F, M_0, \lambda)$ where (P, E, F, M_0) is a marked Petri net and $\lambda : (P \cup E) \to \Lambda$*

is a placement *map such that* $F(p,e) \neq 0 \Rightarrow \lambda(p) = \lambda(e)$ *for every place* $p \in P$ *and for every event* $e \in E$.

Our definition of distributable nets differs notably from Hopkins's definition given in [107], since we impose from start on placement maps a constraint strong enough to ensure the existence of distributed implementations. We postpone the discussion about distributed implementations to section 6.5, and come now to the synthesis of distributable nets.

Definition 6.4.2 (Automata with set of locations) *An automaton with set of* locations Λ *is a quintuple* $A = (S, E, T, s_0, \kappa)$ *where* (S, E, T, s_0) *is an automaton and* $\kappa : E \to \Lambda$ *is a* placement *map.*

The synthesis problem for distributable nets (or the net realization problem for automata with locations) consists in deciding from an automaton (S, E, T, s_0, κ) with set of locations Λ, given as input, whether the underlying automaton (S, E, T, s_0) is isomorphic to the sequential state graph of a distributable net (P, E, F, M_0, λ), to be constructed, with an identical set of locations Λ and such that λ extends κ. We shall produce a decision algorithm for this extended realization problem by restricting the algorithm defined in section 6.3 to a special class of regions, called *localizable* regions, such that conflict occurs exclusively between events with the same location in the induced atomic nets.

Definition 6.4.3 (Localizable regions) *In an automaton with locations* $A = (S, E, T, s_0, \kappa)$*, a region* $(\sigma, {}^\bullet\eta, \eta^\bullet)$ *of* (S, E, T, s_0) *is* localizable *w.r.t.* κ *if* $(\forall e', e'' \in E)\ {}^\bullet\eta(e') \neq 0 \wedge {}^\bullet\eta(e'') \neq 0 \Rightarrow \kappa(e') = \kappa(e'')$.

From Def. 6.4.1 and Def. 6.4.3, each place of a distributable net $\mathcal{N}$ with placement map $\lambda : (P \cup E) \to \Lambda$ determines a region of the sequential state graph $\mathcal{N}^*$ which is localizable with respect to $\lambda\lceil E$. A place p and the region $(\sigma, {}^\bullet\eta, \eta^\bullet)$ which it determines are in fact linked by the relation $F(p,e) = {}^\bullet\eta(e)$, hence ${}^\bullet\eta(e) \neq 0$ entails $\lambda(e) = \lambda(p)$, and ${}^\bullet\eta(e') \neq 0 \wedge {}^\bullet\eta(e'') \neq 0 \Rightarrow \lambda(e') = \lambda(e'')$. Therefore, the sequential state graph of a distributable net has always an admissible subset of localizable regions. Conversely, if P is an admissible subset of regions of the automaton $A = (S, E, T, s_0)$ and all regions in P are localizable with respect to $\kappa : E \to \Lambda$, the net $A^*[P]$ synthesized from P can be lifted to a distributable net. From Def. 6.4.3, one may actually extend κ to a placement map $\lambda : (P \cup E) \to \Lambda$ conform to Def. 6.4.1 by setting $\lambda(e) = \kappa(e)$ for $e \in E$, $\lambda(p) = \kappa(e)$ for places $p \in P$ such that $p = (\sigma, {}^\bullet\eta, \eta^\bullet)$ and ${}^\bullet\eta(e) \neq 0$ for some e, and by choosing arbitrarily $\lambda(p)$ for the remaining places $p \in P$ which do not fit this condition. To sum up:

Proposition 6.4.4 *An automaton with locations* $A = (S, E, T, s_0, \kappa)$ *may be realized by a distributable net if and only if the underlying automaton* (S, E, T, s_0) *has an admissible subset* P *of localizable regions, in which case the net* $A^*[P]$ *synthesized from* P *may always be lifted to a distributable net.*

Therefore, in order to decide on the net realization problem for automata with locations, it suffices to decide on the restricted validity of the separation axioms with respect to localizable regions. Restricting validity of the separation axioms increases notably the complexity of the decision. It may therefore be wise, before deriving a distributable net from an automaton with locations, to derive first a Petri net from the underlying automaton. If this is not possible, the synthesis of the distributable net will certainly fail!

Henceforth, $A = (S, E, T, s_0)$ is a fixed automaton, finite, reachable and event reduced, $\kappa : E \to \Lambda$ is a surjective placement map with codomain $\Lambda = \{1, \ldots, m\}$, and $\{\eta_1, \ldots, \eta_k\}$ is a generating set of distributions of tension over E (see (6.3.1)). We examine successively for the axioms SSA and $ESSA$ the conditions of their restricted validity with respect to the κ-localizable regions. The canonical regions which were used in section 6.3 are not likely to fit in with distribution constraints. We shall consider here another logically complete set of regions, namely the regions of A which reach value 0 at some state $s \in S$ and that may therefore be called *strict* regions of A. These are all regions $R_{\eta,0,\delta}$ where η is a linear combination of the generators $\{\eta_1, \ldots, \eta_k\}$ and $\delta : E \to \mathbb{N}$ satisfies condition 6.6 of Prop. 6.2.9. As a result, we establish the following:

Theorem 6.4.5 *Deciding whether a finite reachable and event reduced automaton with locations is isomorphic to the sequential state graph of a distributable net system and producing this net when it exists takes time polynomial in the number of transitions of the automaton.*

6.4.1 Re-examining states separation

Consider a pair of distinct states s' and s'' and suppose that $\sigma(s') \neq \sigma(s'')$ for some κ-localizable region $(\sigma, {}^\bullet\eta, \eta^\bullet)$. Let $\eta = \eta^\bullet - {}^\bullet\eta$. From $\sigma(s') \neq \sigma(s'')$ follows the assertion *(i)* that $\eta \cdot (\psi(P_{s'}) - \psi(P_{s''})) \neq 0$. From the assumption that $(\sigma, {}^\bullet\eta, \eta^\bullet)$ is κ-localizable, and since $\eta(e) + {}^\bullet\eta(e) \geq 0$ for all e, follows the assertion *(ii)* that there is at most one location $\iota \in \{1, \ldots, m\}$ such that $\eta(e) < 0$ for some event with location $\kappa(e) = \iota$. Conversely, if a distribution of tension η satisfies *(i)* and *(ii)*, one may easily produce from η a κ-localizable region separating s' from s'': one may choose e.g. the region $R_{\eta,0,\mathbf{0}}$ (both strict and pure). Deciding on the existence of a κ-localizable region separating s' from s'' thus reduces to deciding whether *(i)* and *(ii)* are satisfied for some distribution of tension η.

Since η writes as a linear combination $\sum_i z_i \eta_i$ with coefficients $z_i \in \mathbb{Z}$, this can be done by solving or showing unfeasible each system in the indexed family $\Sigma_{\ltimes,\iota}$ of $2(m+1)$ homogeneous systems of linear inequalities in the z_i's defined as follows, with $\ltimes \in \{<, >\}$ and $\iota \in \{0, \ldots, m\}$. Each system $\Sigma_{\ltimes,\iota}$ has one strict inequality expressing condition *(i)*, namely $\sum_i z_i (\eta_i \cdot (\psi(P_{s'}) - \psi(P_{s''}))) \ltimes 0$, plus inequalities enforcing *(ii)*, namely one inequality $\sum_i z_i \cdot \eta_i(e) \geq 0$ for each event e such that $\kappa(e) \neq \iota$.

Because κ is a surjective map, m is bounded by $|E|$ and hence by $|T|$. As each system $\Sigma_{\ltimes,\iota}$ has size polynomial in $|T|$, the indexed family $\Sigma_{\ltimes,\iota}$ has size polynomial in $|T|$. Because all inequalities are homogeneous, $\Sigma_{\ltimes,\iota}$ has a solution in $\mathbb{Z}^k$ if and only if it has a solution in $\mathbb{Q}^k$. Thus

each system in the indexed family $\Sigma_{\ltimes,\iota}$ may be solved up to a multiplicative factor or be shown unfeasible within time polynomial in $|T|$ following the ellipsoid method. Deciding whether the states separation axiom is valid with respect to κ-localizable regions and computing κ-localizable regions that witness its validity takes therefore time polynomial in $|T|$.

6.4.2 Re-examining event/state separation

Consider a state $s' \in S$ and an event $e' \in E$ disabled at s'. Suppose $\sigma(s') < {}^\bullet\eta(e')$ for some κ-localizable region $(\sigma, {}^\bullet\eta, \eta^\bullet)$. Then ${}^\bullet\eta(e') > 0$, hence ${}^\bullet\eta(e) = 0$ for every event e such that $\kappa(e) \neq \kappa(e')$. Therefore, if $\eta = \eta^\bullet - {}^\bullet\eta$, it holds *(iii)* that $\kappa(e) \neq \kappa(e') \Rightarrow \eta(e) \geq 0$ for all $e \in E$. Moreover, it holds *(iv)* that $\eta \cdot (\psi(P_s) - \psi(P_{s'})) > 0$ for every state s enabling e'. Conversely, if a distribution of tension η satisfies *(iii)* and *(iv)*, one may always compute from η a κ-localizable region separating e' from s': one may choose *e.g.* the strict region $R_{\eta,0,\delta}$ with $\delta : E \to \mathbb{N}$ defined by

$$\delta(e') = \mathbf{min}\{0, \eta(e')\} + \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \xrightarrow{e'}\} - \mathbf{min}\{\int_{s_0}^{s} \eta \mid s \in S\}$$

and $\delta(e) = 0$ for $e \neq e'$. Deciding upon the existence of a κ-localizable region separating e' from s' reduces therefore to deciding whether *(iii)* and *(iv)* are satisfied for some distribution of tension η.

Now letting $\eta = \sum_i z_i \cdot \eta_i$ with $z_i \in \mathbb{Z}$, consider the system of homogeneous linear inequalities in the unknown z_i's defined as follows: Σ has a first series of inequalities reflecting *(iii)*, namely one inequality $\sum_i z_i \cdot \eta_i(e) \geq 0$ for each event e such that $\kappa(e) \neq \kappa(e')$, and a second series of inequalities reflecting *(iv)*, namely one inequality $\sum_i z_i \, (\eta_i \cdot (\psi(P_s) - \psi(P_{s'}))) > 0$ for each state s enabling e'. All inequalities are homogeneous, hence Σ may be solved or shown unfeasible using time polynomial in $|T|$. Deciding whether the event/state separation axiom is valid with respect to κ-localizable regions and computing κ-localizable regions that witness its validity can therefore be done in polynomial time.

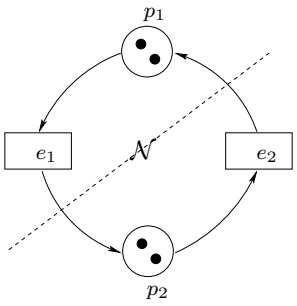
6.5 From Distributable Nets to Distributed Automata

6.5.1 Simple Distribution Scheme

As yet, we have shown how constructing from a finite automaton with locations, when this is possible, a distributable net with the specified locations for events and with a sequential state graph isomorphic to the given automaton. This net is obviously *bounded*: each place p has a least upper bound $\overline{p}$ in the reachable markings. In order to complete the machinery for distributing finite automata, it remains to show that a bounded distributable net with m locations may always be implemented by m finite automata $A_1, \ldots, A_m$ communicating with each other by asynchronous message passing.

From now on, $\mathcal{N} = (P, E, F, M_0, \lambda)$ is a bounded distributable net with set of locations $\{1, \ldots, m\}$, hence $\lambda : (P \cup E) \to \{1, \ldots, m\}$. In order to produce a distributed implementation $\{A_1, \ldots, A_m\}$ of $\mathcal{N}$, we proceed in two stages. In the first stage, we extend $\mathcal{N}$ to a larger net $\mathcal{N}'$ in which $i)$ each place p of $\mathcal{N}$ is split to $m + 1$ places: one *local place* (p, i) for each location $i \in \{1, \ldots, m\}$, and one *global place* $(p, 0)$ whose tokens represent messages in transit, and $ii)$ *silent* events are supplied for *sending* tokens i.e. for moving them from (p, i) to $(p, 0)$ when $\lambda(p) \neq i$, or for *receiving* tokens i.e. for moving them from $(p, 0)$ to (p, i) when $\lambda(p) = i$. We show that $\mathcal{N}'$ is *divergence free* and *branching bisimilar* to $\mathcal{N}$ under the assumption that silent events are unobservable. In the second stage, we remove the global places. The effect of the removal is to disconnect $\mathcal{N}'$ and to produce m component nets $\mathcal{N}_i$. The finite automata A_i are obtained from the state graphs of the nets $\mathcal{N}_i$ by cutting out every marking in which some place (p, i) exceeds $\overline{p}$ (the least upper bound of p in $\mathcal{N}$).

We describe now the construction of $\mathcal{N}' = (P', E', F', M_0')$. An illustration is given in Fig. 6.4 (where locations are indicated as subscripts). Each place of $\mathcal{N}$ is replicated $m + 1$ times in $\mathcal{N}'$,



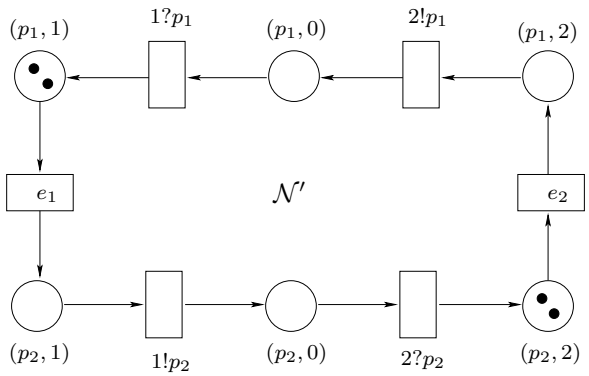


Figure 6.4: Constructing $\mathcal{N}'$

thus $P' = P \times \{0, \ldots, m\}$. The initial marking is determined from the initial marking of $\mathcal{N}$

by the relation $M'_0((p,i)) = M_0(p)$ if $\lambda(p) = i$ and 0 otherwise, suggesting the privileged role of the place $(p, \lambda(p))$ among the representatives of p. E' is the union $E \cup S \cup R$ of the set of events of $\mathcal{N}$ and two new sets of events S (for sending) and R (for receiving). The flow of tokens attached to the events in E reproduces the flow of tokens in $\mathcal{N}$ up to the following adjustments. Let $F'((p,i),e) = F(p,e)$ if $\lambda(e) = i$ and 0 otherwise, and similarly let $F'(e,(p,i)) = F(e,p)$ if $\lambda(e) = i$ and 0 otherwise. As $\mathcal{N}$ is a distributable net, $F'((p,i),e) \neq 0 \Rightarrow \lambda(p) = i$, thus input and output are in fact dealt with asymmetrically. Let us come next to silent events. For each place p of $\mathcal{N}$ and for each location $i \neq \lambda(p)$, an event $i!p$ is supplied for sending tokens from the local place (p,i) to the global place $(p,0)$. Thus $S = \{\, i!p \,|\, 1 \leq i \leq m \wedge p \in P \wedge i \neq \lambda(p)\}$, and the flow of tokens attached to $i!p$ is defined by $F'((p,i),i!p) = F'(i!p,(p,0)) = 1$ and $F'(x,i!p) = F'(i!p,x) = 0$ for any other place $x \in P'$. Last, for each place p of $\mathcal{N}$ with location $\lambda(p) = i$, an event $i?p$ is supplied for receiving tokens into the local place $(p,\lambda(p))$ from the global place $(p,0)$. Thus $R = \{i?p \,|\, p \in P \wedge i = \lambda(p)\}$, and the flow of tokens attached to $i?p$ is defined by $F'((p,0),i?p) = F'(i?p,(p,i)) = 1$ and $F'(x,i?p) = F'(i?p,x) = 0$ for any other place $x \in P'$. This completes the definition of $\mathcal{N}'$.

We want to show that $\mathcal{N}$ and $\mathcal{N}'$ are equivalent up to an abstraction of the silent events $r \in R$ and $s \in S$. In order to define precisely this equivalence, let us relabel by some new symbol $\tau \notin E$ all the transitions labelled by $r \in R$ or $s \in S$ in the sequential state graph $\mathcal{N}'^{*} = (RM(\mathcal{N}'), E', T', M'_0)$. Next let $\sim \subseteq RM(\mathcal{N}) \times RM(\mathcal{N}')$ be the relation between the reachable markings of the respective nets $\mathcal{N}$ and $\mathcal{N}'$ such that $M \sim M'$ if and only if $M(p) = \sum\{M'((p,i)) \,|\, 0 \leq i \leq m\}$ for every $p \in P$. We shall prove the following facts:

1. $\mathcal{N}'^{*}$ is *divergence free* which means that no infinite sequence of τ-labelled transitions takes place in this graph,
2. different markings of $\mathcal{N}$ have disjoint images under $\sim$, i.e., relation $\sim^{-1}$ acts functionally on $RM(\mathcal{N}')$,
3. $\sim$ is a *branching bisimulation* [174], which means in our specific case that the following assertions are valid, with e ranging over E, and with the subscripted M and M' ranging respectively over $RM(\mathcal{N})$ and $RM(\mathcal{N}')$.
   - **a)** $M_0 \sim M'_0$,
   - **b)** $M_1 \sim M'_1 \wedge M'_1 \xrightarrow{\tau} M'_2 \Rightarrow M_1 \sim M'_2$,
   - **c)** $M_1 \sim M'_1 \wedge M'_1 \xrightarrow{e} M'_2 \Rightarrow \exists M_2 \cdot M_1 \xrightarrow{e} M_2 \wedge M_2 \sim M'_2$,
   - **d)** $M_1 \sim M'_1 \wedge M_1 \xrightarrow{e} M_2 \Rightarrow \exists M'_2 \cdot M'_1 \,(\xrightarrow{\tau})^{*} \xrightarrow{e} M'_2 \wedge M_2 \sim M'_2$.

Relations **(a)** and **(b)** follow directly from the definitions of M', F' and $\sim$. The considered definitions also entail the following:

e) $M_1 \sim M'_1 \wedge M_1 \xrightarrow{e} M_2 \wedge M'_1 \xrightarrow{e} M'_2 \Rightarrow M_2 \sim M'_2$,

f) $M_1 \sim M'_1 \wedge M'_1 \xrightarrow{e} \Rightarrow M_1 \xrightarrow{e}$,

g) $M_1 \sim M'_1 \wedge M_1 \xrightarrow{e} \wedge \forall p \cdot M_1(p) = M'_1(p,\lambda(p)) \Rightarrow M'_1 \xrightarrow{e}$.

Now **e** $\wedge$ **f** $\Rightarrow$ **c**, and **b** $\wedge$ **e** $\wedge$ **g** $\Rightarrow$ **d** provided that in $\mathcal{N}'^*$ all maximal sequences of τ-labelled transitions originated from M'_1 lead to a marking M' such that $M'(p,\lambda(p)) = \sum\{M'_1((p,i)) \,|\, 0 \leq i \leq m\}$ for every $p \in P$.

For $p \in P$, let $\delta_p(M'_1) = \sum\{M'_1((p,i)) \,|\, 1 \leq i \leq m \wedge i \neq \lambda(p)\}$ and $\Delta_p(M'_1) = \sum\{M'_1((p,i)) \,|\, 0 \leq i \leq m \wedge i \neq \lambda(p)\}$. With these definitions, any maximal sequence of τ-labelled transitions originated from M'_1 includes exactly $\delta_p(M'_1)$ occurrences of sending events $i!p$ and $\Delta_p(M'_1)$ occurrences of receiving events $i?p$ for each $p \in P$. Thus, all maximal sequences of τ-labelled transitions originated from M'_1 have the same bounded length $\sum_p(\delta_p(M'_1) + \Delta_p(M'_1))$, which establishes fact 1. Moreover, by (b), any maximal sequence of τ-labelled transitions originated from M'_1 reaches the indicated marking M', which establishes fact 2. Therefore, $\mathcal{N}$ and $\mathcal{N}'$ behave in the same way up to an abstraction of the silent events; moreover, this simulation preserves distinctions between markings: separate markings of $\mathcal{N}$ have disjoint sets of images under relation $\sim$.

It remains to derive from $\mathcal{N}'$ finite communicating automata $A_1, \ldots, A_m$. For this purpose, we remove from $\mathcal{N}'$ the global places $(p,0)$. What is left is a family of nets $\mathcal{N}'_i$, $i \in \{1,\ldots,m\}$, as follows: $\mathcal{N}'_i = (P'_i, E'_i, F'_i, M'_{i,0})$, P'_i is the set of local places with location i, E'_i is the set of events or silent events with location i, and F'_i and $M'_{i,0}$ are the induced restrictions of F' and M'_0. Thus $P'_i = P \times \{i\}$ and $E'_i = \{e \in E \,|\, i = \lambda(e)\} \cup \{\, i!p \,|\, p \in P \wedge i \neq \lambda(p)\} \cup \{\, i?p \,|\, p \in P \wedge i = \lambda(p)\}$. For each $i \in \{1,\ldots,m\}$, let A_i be the finite automaton that derives from the state graph of $\mathcal{N}'_i$ by cutting out all markings where some place (p,i) exceeds $\overline{p}$ (the least upper bound of p in $\mathcal{N}$). The resulting family $\{A_1,\ldots,A_m\}$ is a distributed implementation of $\mathcal{N}'$, and therefore of $\mathcal{N}$. Each automaton A_i is installed in the associated location i. An event $i!p$ (hence $i \neq \lambda(p)$) is interpreted by sending message p to the destination $\lambda(p)$. An event $i?p$ (hence $i = \lambda(p)$) is interpreted by looking for message p in the mailbox at the address $\lambda(p)$. Each message is supposed to reach its destination. No other assumption is made on synchronization.

The markings of $\mathcal{N}'$ are thus represented by tuples $(s_0, s_1, \ldots, s_m)$ where $s_j : \{(p,j) \,|\, p \in P\} \rightarrow \mathbb{N}$: for $j = 0$, s_j represents the global state of the communication medium; for $j \in \{1,\ldots,m\}$, s_j is the local state of automaton A_j. This representation is injective; therefore, distinctions between markings are preserved by the distributed implementation of $\mathcal{N}$.

6.5.2 Optimized Distribution Scheme

We propose here two independent optimizations of the distribution scheme explained in section 6.5.1, aiming to decrease the flow of messages on the communication medium (6.5.2) or to decrease the size of the communicating automata (6.5.2).

Aggregating Messages

The distribution scheme specified above leads to inefficient communication scenarios. For instance, two messages are needed for implementing a transition that produces tokens for two

remote places even though they are mapped on the same location. A refined distribution scheme, producing automata with less communications, is presented now. It is based essentially on the same ideas, but less places are added and less tokens flow through the communication medium (see figure 6.5).

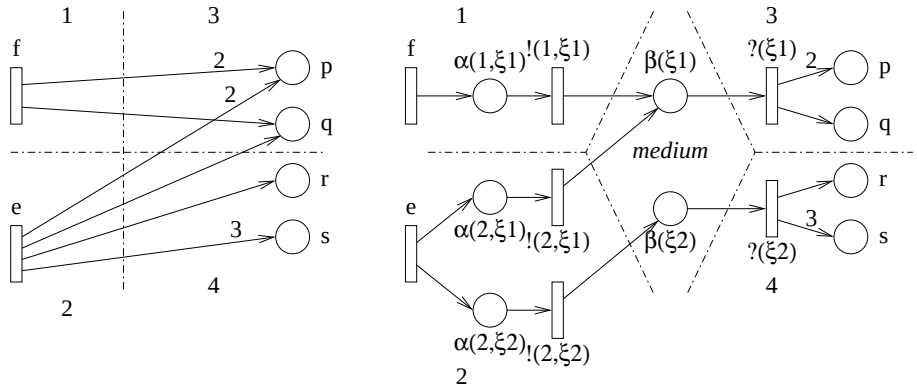


Figure 6.5: Communications between locations

From now on, let $\mathcal{N} = (P, E, F, M_0, \lambda)$ be a distributable net system over set of locations Λ. Each event $e \in E$ has an output flow $e^\bullet : P \to \mathbb{N}$ such that $e^\bullet(p) = F(e, p)$. This output flow may be decomposed thus into a *local* part $e^\bullet_{local} : P \to \mathbb{N}$ and a *remote* part $e^\bullet_{remote} : P \to \mathbb{N}$:

$$e^\bullet_{local}(p) = \begin{cases} e^\bullet(p) & \text{if } \lambda(e) = \lambda(p) \\ 0 & \text{otherwise} \end{cases}$$

$$e^\bullet_{remote}(p) = \begin{cases} e^\bullet(p) & \text{if } \lambda(e) \neq \lambda(p) \\ 0 & \text{otherwise} \end{cases}$$

The latter part may be further decomposed as $e^\bullet_{remote} = \Sigma_{l \in \Lambda}\, e^\bullet_l$ where $e^\bullet_l : P \to \mathbb{N}$ is the map such that:

$$e^\bullet_l(p) = \begin{cases} e^\bullet(p) & \text{if } l = \lambda(p) \neq \lambda(e) \\ 0 & \text{otherwise} \end{cases}$$

For $e \in E$ and $l \in \Lambda$, define:

$$R_e = \{e^\bullet_l \,|\, l \in \Lambda,\ e^\bullet_l \neq \mathbf{0}\}$$

$$R_l = \bigcup_{e \in \lambda^{-1}(l)} R_e$$

$$R = \bigcup_{l \in \Lambda} R_l$$

The distributable net system $\mathcal{N}' = (P', E', F', M'_0, \lambda')$ may be re-defined as follows. A new location, representing the communication medium, is added: $\Lambda' = \Lambda \uplus \{\mathit{medium}\}$. Two types of communication places are added: on the one hand places $\alpha(l, \xi)$ used as output buffers, where each packet $\xi \in R_l$ is represented by a single token, and on the other hand places $\beta(\xi)$ used as communication buffers, where each token represents a packet in transit through the communication medium. This results in the set of places $P' = P \uplus \{\, \alpha(l, \xi) \mid l \in \Lambda,\, \xi \in R_l \,\} \uplus \{\, \beta(\xi) \mid \xi \in R \,\}$. Silent events $!(l, \xi)$ are supplied for moving packets $\xi \in R_l$ from output buffers $\alpha(l, \xi)$ to the communication buffer $\beta(\xi)$; silent events $?(\xi)$ are supplied for picking packets ξ from the communication medium and dispatching them on arrival. This results in the set of events $E' = E \uplus \{\, !(l, \xi) \mid l \in \Lambda,\, \xi \in R_l \,\} \uplus \{\, ?(\xi) \mid \xi \in R \,\}$. Let the locations of the new places and events as follows:

$$\begin{array}{lcl} \lambda'(\alpha(l, \xi)) & = & l \\ \lambda'(\beta(\xi)) & = & \mathit{medium} \\ \lambda'(!(l, \xi)) & = & l \\ \lambda'(?(e_l^\bullet)) & = & l \end{array}$$

The other locations are like in $\mathcal{N}$. For places $p \in P$ and events $e \in E$ inherited from $\mathcal{N}$, the flow relation F' is defined thus: $F'(x, y) = F(x, y)$ if $\lambda(x) = \lambda(y)$, 0 otherwise. The flow relation F' is extended as follows to the new places or events, with $F'(x, y) = 0$ in all cases left unspecified:

$$\begin{array}{ll} (\, e \in E,\, \xi \in R_e \,) & F'(e, \alpha(\lambda(e), \xi)) = 1 \\ (\, e \in E,\, \xi \in R_e \,) & F'(\alpha(\lambda(e), \xi), !(\lambda(e), \xi)) = 1 \\ (\, l \in \Lambda,\, \xi \in R_l \,) & F'(!(l, \xi), \beta(\xi)) = 1 \\ (\, \xi \in R \,) & F'(\beta(\xi), ?(\xi)) = 1 \\ (\, \xi \in R,\, p \in P \,) & F'(?(\xi), p) = \xi(p) \end{array}$$

One proves that $\mathcal{N}'^*$ is divergence free and branching bisimilar to $\mathcal{N}^*$ in the same way as in section 6.1, but using now the alternative relation $M \sim M'$ if and only if $\forall p \in P,\, M(p) = M'(p) + \Sigma_{\xi \in R}\, \xi(p) \times M'(\beta(\xi)) + \Sigma_{i \in \Lambda,\, \xi \in R}\, \xi(p) \times M'(\alpha(i, \xi))$.

A distributed implementation of $\mathcal{N}'$ follows along the same lines as in section 6.5.1 : the automata A_i are obtained by expanding the state graphs of the sub-nets $\mathcal{N}'_\rangle$ of $\mathcal{N}'$, with bounds $\overline{p}$ set on places $(p, i) \in P'_i$.

Pruning Automata

A drawback of the distribution schemes described so far is to produce automata bigger than needed: the automata A_i may be ready to accept inputs from the communication medium at states where no input will ever come! One may remedy this drawback by computing directly the A_i from the state graph of $\mathcal{N}'$. The stages of the computation are as follows. For each $i \in \{1, \ldots, m\}$, let A'_i be the automaton obtained from $\mathcal{N}'^*$ by renaming with τ ($\notin E'$) all transitions $M \xrightarrow{e} M'$ such that $e \in E' \setminus E'_i$ (E'_i is defined as in section 6.5.1). Next let A''_i be the finite non-deterministic automaton whose transitions $M \xrightarrow{e} M'$ are derived from the sequences of transitions $M = M_1 \xrightarrow{\tau} M_2 \ldots \xrightarrow{\tau} M_n \xrightarrow{e} M'$ in A'_i such that $e \neq \tau$. Finally let A_i be the finite deterministic automaton which derives from A''_i according to the traditional subset construction. Since the subset of events $E' \setminus E'_i$ thus abstracted from is precisely the subset of events which do not affect the places in P'_i, the automaton A_i is isomorphic to the induced restriction of $\mathcal{N}'^*_i$ on the subset of markings of $\mathcal{N}'_i$ which occur as sub-markings of reachable markings of $\mathcal{N}'$. Therefore, the system of communicating automata $\{A_1, \ldots, A_m\}$ has a state graph isomorphic to $\mathcal{N}'^*$. Since the bisimulation $\sim^{-1}$ between $\mathcal{N}'^*$ and $\mathcal{N}^*$ acts functionally on $RM(\mathcal{N}')$, $\{A_1, \ldots, A_m\}$ is a distributed implementation of $\mathcal{N}$, and this implementation preserves distinctions between markings.

Remark 6.5.1 *If one does not insist on realizing $\mathcal{N}'^*$ up to isomorphism, one can go one step further by minimizing the automata A_i. The language of $\mathcal{N}$ then coincides with the set of sequences of events in E performed by the communicating automata.*

Remark 6.5.2 *In order to abstract from a subset of events $\mathcal{E} \subseteq E$, it suffices to redefine A'_i as the copy of $\mathcal{N}'^*$ in which all events in $(E' \setminus E'_i) \cup \mathcal{E}$ have been replaced with τ. If $\mathcal{N} = A^*[P]$, the set of sequences of events in $E \setminus \mathcal{E}$ that may be performed by the communicating automata A_i coincides now with the language of the $\mathcal{E}$-collapse of A (the automaton obtained by collapsing every pair of states s and s' such that $s \xrightarrow{e} s'$ and $e \in \mathcal{E}$).*

6.6 Case Studies in Distributing Reactive Automata

This section details two applications of distributable Petri net synthesis. The first case is the systematic derivation of two distributed protocols for mutual exclusion. The second case is the synthesis of a simplified transport level communication protocol derived from the INRES protocol [106]. Both examples illustrate a new methodology for distributed program synthesis, based on the algorithms for distributable Petri net synthesis presented in the previous sections : a distributable Petri net is first synthesized and then turned into a collection of communicating finite state automata, each of which defines the behaviour of the sequential process located at a specific site of the distributed architecture. We could have contented ourselves with distributing the sytnhesized Petri net over the distributed architecture without going to sequential machines. This would fit nicely if the goal was to implement protocols in hardware, since concurrency at each

site could then be exploited within circuit synthesis. However, this would not meet the customary requirements for software implementations of low-level protocols : each site is one processor, and emulating concurrency would be too inefficient.

6.6.1 Mutual Exclusion

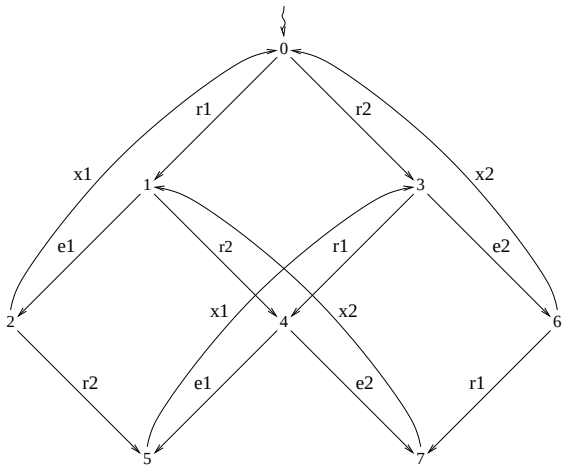


Figure 6.6: Specification of the mutual exclusion service

Introduction

Mutual exclusion is one of the basic services that are often present in distributed operating systems. We focus on the possible ways of achieving mutual exclusion on an asynchronous network of processors in which communication is performed by message passing and such that no message shall be lost or replicated.

The specification of mutual exclusion between two users U_1 and U_2 is given in figure 6.6. Event r_i, $i \in \{1, 2\}$ corresponds to a request by user U_i to enter critical section. The meaning of event e_i, $i \in \{1, 2\}$ is that user U_i is allowed to enter critical section while event x_i, $i \in \{1, 2\}$ corresponds to the exit of user U_i from critical section. Mutual exclusion is ensured whenever both agents cannot be in critical section at the same time.

A distributed implementation of mutual exclusion consists in a pair of sequential processes P_1, P_2 where process P_i, $i \in \{1, 2\}$ controls user U_i and both processes communicate only by asynchronous message passing.

We wish to apply the method sketched in sections 6.4 and 6.5 to the derivation of processes P_1 and P_2. Unfortunately, the automaton in figure 6.6 is not isomorphic to the reachability graph of any distributable Petri net. This is due to the conflict between events e_1 and e_2 in state 4 while these two events are mapped to distinct locations.

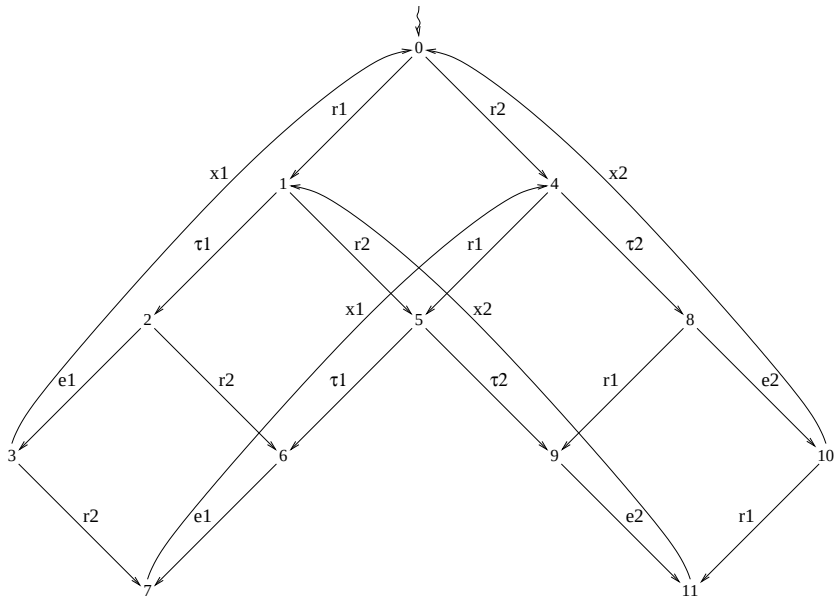


Figure 6.7: Mutual exclusion: a first refinement

The next sections describe two ways of inserting silent events in the automaton of figure 6.6 so that distributable Petri net synthesis becomes feasible. The process of inserting silent transitions is manual and does rely solely on designer's intuition. In this respect, it doesn't differ in any way from silent transition insertion in [177].

A monitor-based solution

A first solution consists in refining event e_i, $i \in \{1, 2\}$ by $\tau_i e_i$ where τ_i is located on a third process M. This leads to the automaton in figure 6.7 which is comformant to the original automaton with respect to the **ioco** testing equivalence [172]. The refined automaton is isomorphic to the reachability graph of the distributable Petri net in figure 6.8.

Communicating automata can then be produced from the distributable Petri net, following the method described in sections 6.5.1 and 6.5.2. By abstracting from the subset of events $\mathcal{E} = \{\tau_1, \tau_2\}$, and by applying minimization, one obtains the automata shown in figure 6.9. The process at location M acts as a monitor for the two other processes $P_i, i = 1 \ldots 2$, which in turn act as interfaces between user processes U_i and the monitor M. It follows from the method that the implementation is behaviorally correct, hence all and only the sequences of events which are compatible with the service specifications in figure 6.6 can occur. In spite of the minimization, the automaton in figure 6.6 can moreover be reconstructed from the distributed implementation.

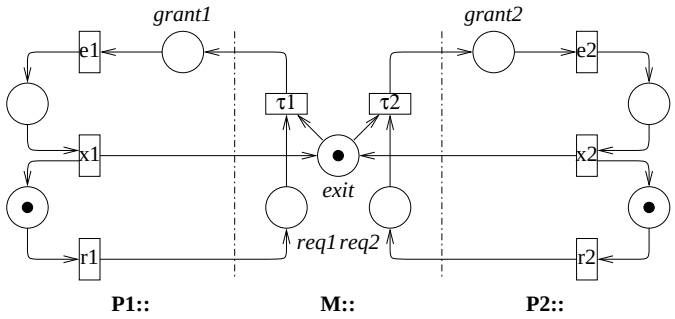


Figure 6.8: Mutual exclusion: distributable Petri net generated from the first refinement

A token-based solution

A second solution to the mutual exclusion protocol synthesis problem is to split states 0, 1, 3 and 4 and to insert converse transitions labelled by silent events τ_1 and τ_2 as represented in figure 6.10. Event τ_i, $i \in \{1, 2\}$ is located on process P_i therefore no distributed conflict is created and more importantly, the distributed conflict between events e_1 and e_2 is alleviated. Not surprisingly, the reachability graph of the synthesized Petri net shown in figure 6.11 is isomorphic to the refined automaton.

Two communicating automata may be derived from the distributable Petri net, following the method described in sections 6.5.1 and 6.5.2. By abstracting from the subset of events $\mathcal{E} = \{\tau_1, \tau_2\}$, one obtains the automata shown in figure 6.12; mutual exclusion is thus achieved by circulating a token between processes P_1 and P_2. Minimization has not been used: the automata produced by determinization are already minimal. It follows therefore from the method that the automaton in figure 6.6 can be reconstructed from this distributed implementation.

6.6.2 A Simplified INRES Protocol

In this section, we consider a simplified version of the INRES communication protocol [106]. The specifications of service of the INRES protocol are given in figure 6.13. For the sake of the exposition, we consider the simplified service described in figure 6.14. The simplified protocol defines the transmission of data between two users $user - A$ and $user - B$ linked to respective protocol entities A and B. Event s means that entity A is given by $user - A$ some data to transmit; event r means that entity B delivers some data to $user - B$; event d is a disconnection request (located on B); event a is the notification of the disconnection (located on A). With this protocol, data exchanges (words in $(sr)^\star$) may take place until a disconnection is requested.

We aim at synthesizing a distributed implementation of this protocol: two processes A and B, communicating with one another through a reliable communication medium. Events s and a are located on process A, while events r and d are located on B. This is defined by the location map λ: $\lambda(s) = \lambda(a) = A$ and $\lambda(r) = \lambda(d) = B$.

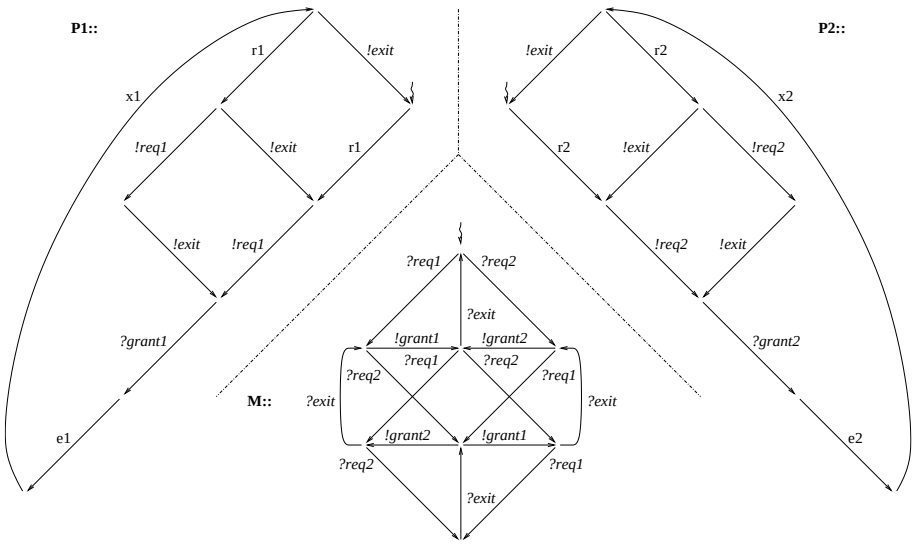


Figure 6.9: Mutual exclusion: communicating automata generated from the first refinement

We wish to use *distributable* Petri net synthesis to produce a Petri net implementation of the communication protocol: thus, for every place p of the net, the flow relation F must satisfy:

$$F(p,s) = F(p,a) = 0 \text{ or } F(p,r) = F(p,d) = 0$$

Unfortunately the automaton specifying the service (shown on the left hand of figure 6.14) is not isomorphic to the marking graph of any distributable Petri net: event a and state 2 cannot be separated. However this can be alleviated by refining the automaton into a weakly bisimilar automaton which is actually the marking graph of a distribuable Petri net.

In [72, 70], is advocated an event splitting heuristics for refining non separated automata into separated automata. In our case, the two occurrences of s may be replaced by s_1 and s_2, and similarly for d, leading to three different refinements. However, none of the refined automata is separated with respect to the restricted set of *localizable* regions, compatible with the distribution constraints. Even though in the non distributed case event splitting is a systematic method for refining automata into separated automata, it is potentially useless in the distributed case.

As an alternative resort, refinement can be done by inserting silent transitions while keeping weak bisimilarity. This has been used in [177] for a similar purpose. In our case, the adequate refinement step consists in replacing transition $3 \xrightarrow{a} 4$ by two transitions: $3 \xrightarrow{\tau} 4 \xrightarrow{a} 5$ (see the automaton on the right hand side of figure 6.14), with silent event τ located on process B. The refined automaton is then isomorphic to the reachable marking graph of the Petri net shown in figure 6.15. Places and transitions are sorted according to their locations: A on the left and B on the right.

The distributable Petri net is then expanded into the Petri net shown in figure 6.16 by making

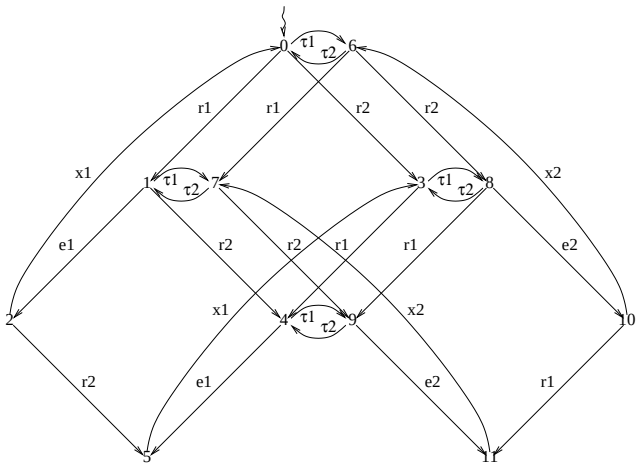


Figure 6.10: Mutual exclusion: a second refinement

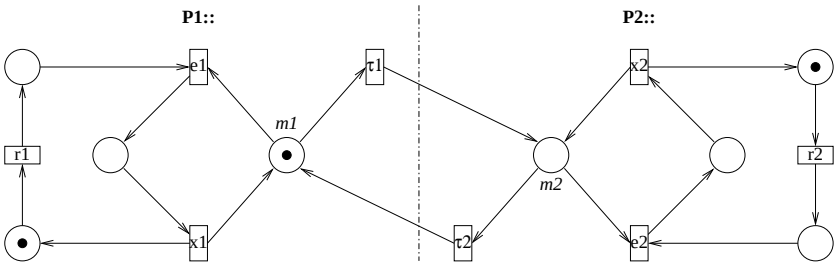


Figure 6.11: Mutual exclusion: distributable Petri net generated from the second refinement

communications between processes explicit. By distributing its reachable state graph as indicated in section 6.5.2, with an abstraction from $\mathcal{E} = \{\tau\}$ and with minimization, one obtains the automata shown in figure 6.17. It follows from the method that this distributed implementation is behaviorally correct. In spite of the minimization, the automaton on the left hand side of figure 6.14 can moreover be reconstructed from this distributed implementation. The full INRES protocol can be dealt with in much the same way, however with an increased complexity.

Another solution consists in a partial unfolding of the service automaton such that (i) in every directed cycle, the number of occurrences of each state is congruent to zero modulo two and (ii) concurrency is preserved. This means that the cycle $s.r$ is unfolded into a cycle $s_1.r_1.s_2.r_2$ and that the concurrent diamond $s|d$ is preserved in the unfolding as two concurrent diamonds $s_1|d_1$ and $s_2|d_2$, as shown on the left-hand side of figure 6.18. This unfolding is a correct refinement of the service since it is bisimilar to the service automaton. Interestingly, the unfolded automaton is isomorphic to the marking graph of a distributable net, shown on the right-hand side of figure 6.18. This gives another implementation of the protocol, without silent transition.

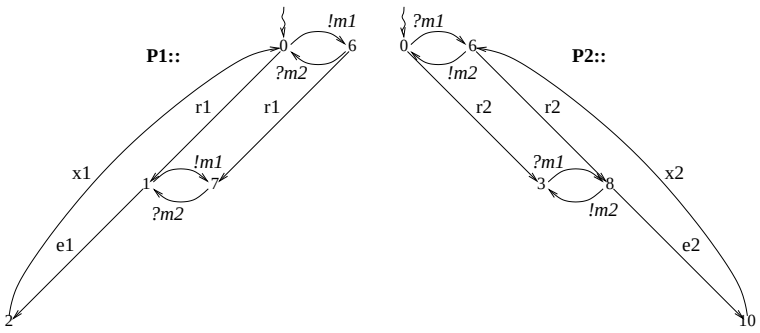


Figure 6.12: Mutual exclusion: communicating automata generated from the second refinement

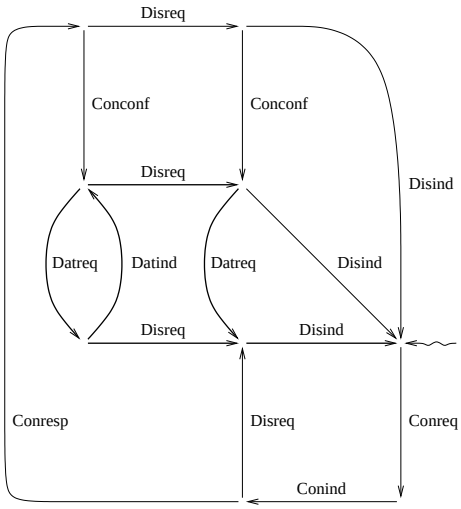


Figure 6.13: The INRES protocol: specification of service

Remark that unfolding the service automaton is by no means a general method. The counter-example is given figure 6.19 (left-hand side), where label a is mapped to location A, b to B, c to C, x and y to D. Consider an unfolding of this automaton such that concurrent diamonds $a|x$ and $a|y$ are preserved (see for instance figure 6.19, right-hand side). Event-state separation problems remain unsolvable in every state preceding $a_i.b_i$ (resp. $a_i.c_i$) whenever this state can be reached by a word of the form $u.c_j.x_i$ (resp. $u.b_j.y_i$). Consider for instance the unfolding given figure 6.19.

6.7 Conclusion

A novel method for producing distributed implementations of finite reactive automata was presented in this chapter, based on the synthesis of general Petri nets, that relies in turn on linear

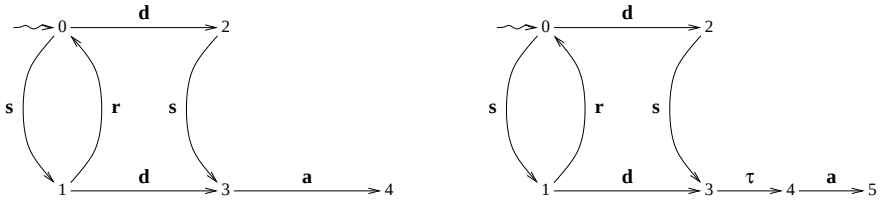


Figure 6.14: Simple protocol: specification of service and its refinement

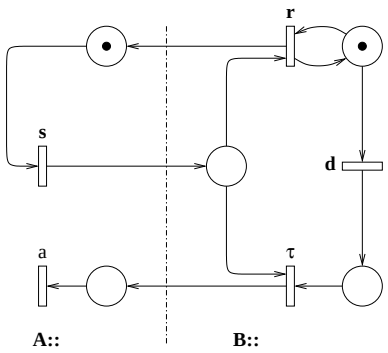


Figure 6.15: Simple protocol: synthesized Petri net

algebra. A tool called SYNET was built to this effect, integrating elementary algorithms on graphs and standard algorithms of linear algebra. Our limited experience with the application of this tool to the synthesis of distributed protocols makes us rather confident in the practical usefulness of the method. Nevertheless, different approaches to the distributed realization of finite reactive automata may emerge, and we do not claim that the method based on distributable Petri nets will always give the best results. Notwithstanding this fact, two major problems remain still to be solved in the framework of distributable Petri net synthesis. The first problem, touched upon in section 7, is to define techniques for transforming arbitrary automata into synthesizable automata. We saw that event splitting is not the right answer in the context of distribution. More promising is the investigation of silent event insertions, started by W. Vogler in the simpler context of one-safe nets and with a view at independence rather than distribution [175]. The second problem, more general, is to re-examine Petri net synthesis for transition systems given by computational rules rather than by extension (i.e., as sets of states and transitions). One may take as examples non expanded products of finite automata, or guarded expressions in some logical language, and more generally symbolic transition systems. Dealing with both types of problems would open large perspectives to the ideas exposed here.

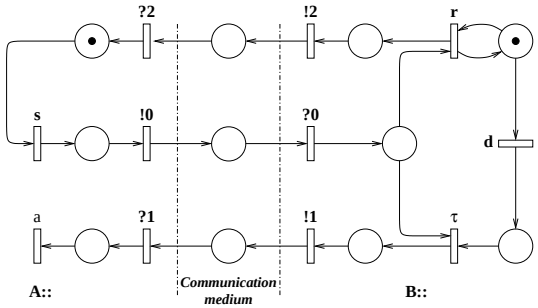


Figure 6.16: Simple protocol: Petri-net with communications

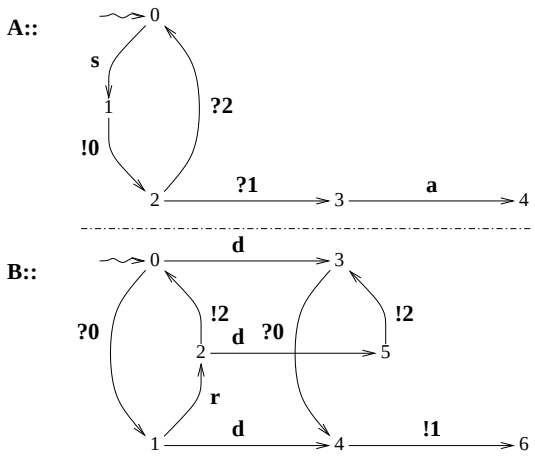


Figure 6.17: Simple protocol: communicating automata

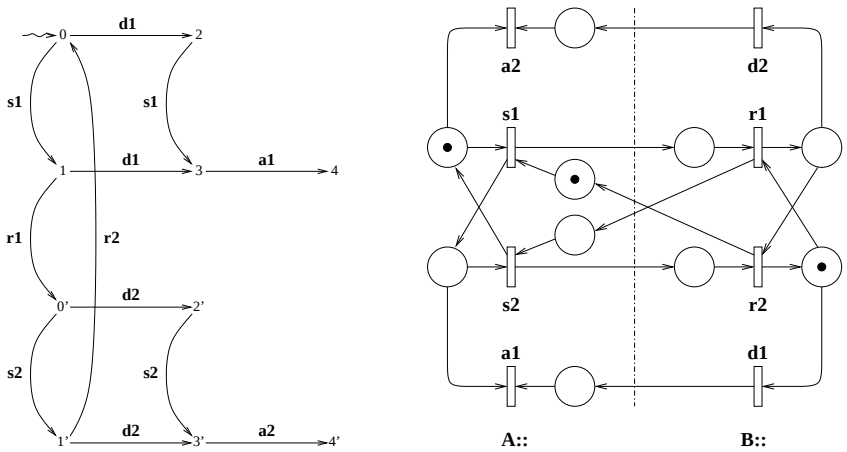


Figure 6.18: Simple protocol: unfolding modulo 2 and resulting distributable net

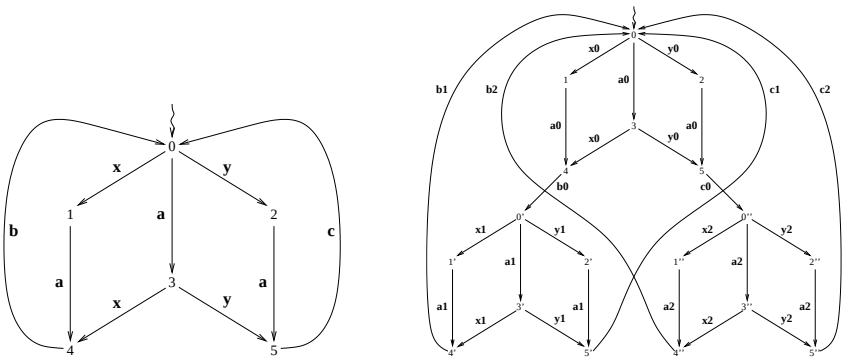


Figure 6.19: Counter-example proving unfolding lacks generality and one possible unfoldingcd fig

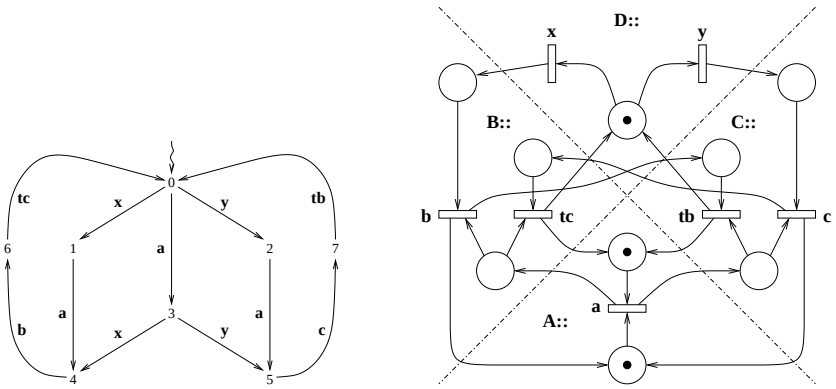


Figure 6.20: Insertion of τ-transitions and resulting distributable net

Chapter 7

Concurrent secrets

Résumé : *Il arrive fréquemment que les médias révèlent l'exploitation frauduleuse d'une faille dans le système d'information d'une compagnie ou d'un organisme public. Ceci n'est pas une surprise quand on connait la complexité de ces systèmes. De fait, l'analyse et le contrôle des flux d'information dans les systèmes de ce type est un sujet de grande importance. En effet, il est souhaitable de pouvoir empêcher toute fuite d'information confidentielle, avant même qu'elle ait eu lieu.*

Reprenant la propriété d'opacité proposée par Mazaré et al. [133, 47], ce chapitre, repris de [11], aborde la question de la synthèse de contrôle de supervision, pour assurer l'absence de fuite d'informations dites secrètes, quand le système est exposé à un environnement constitué d'un ensemble d'agents pouvant être pernicieux. L'existence d'un contrôle maximal permissif est démontrée. Ce contrôle peut être calculé dans plusieurs cas particuliers, en fonction des ensembles d'actions observables par chacun des agents et des langages secrets.

7.1 Introduction

This work is an attempt to import supervisory control into the area of computer security. Given an automaton, or plant, and given specifications of the desired behaviour of the plant, Ramadge and Wonham's theory presented in [155, 154] yields a finite, non blocking, and maximal permissive control of the plant enforcing this behaviour (in the normal case or when unobservable events are uncontrolable). Controller synthesis is a desirable complement to model checking, for it allows curing the problems that model checkers can reveal. Supervisory control has found applications in manufacturing systems, in embedded systems, and more generally in safety critical systems. We feel it could find applications as well in computer security, and we shall strive to support this thesis.

With the above goal in mind, we have searched for a class of security problems likely to be dealt with as control problems. We model an interactive computer system and its users as a closed entity in which the users observe their own interactions with the system. The closed entity is

represented with a finite automaton over an alphabet Σ. The synchronous interactions between each user i and the system are figured by the elements of a corresponding sub-alphabet $\Sigma_i \subseteq \Sigma$ (users may synchronize when their sub-alphabets intersect). Usually in supervisory control, the control objective is a predicate on the runs of the plant, specifying some combination of safety and liveness properties, and the observers act as sensors, *i.e.* they supply informations on the status of the plant, used by the controller to produce an adequate feedback enabling or disabling events in the plant. Here, the game is different: the observers are not on the side of the controller but they are opponents. As for the control objective, there are still predicates (S_i) on the runs of the system, but the interpretation is again different: an observer i should never find out that the actual trajectory of the system belongs to the secret (S_i) he has been assigned.

One reason why we believe the model sketched above is worth investigating is that, in the case of a single observer, it has already been introduced independently in [133] and studied further in [47]. What we call *secrets* here was called there *opaque predicates*, albeit with larger families of predicates (sets of runs) and observation functions. It was shown in [47] that anonymity problems and non-interference problems may be reduced to opacity problems, using suitable observation functions. It was shown *ibidem* that model-checking a system for opacity is undecidable in the general case where an opaque predicate may refer to the visited states or may be any recursive predicate on sequences of event labels. Nonetheless, techniques based on abstract interpretation were proposed in [47] for checking opacity in unbounded Petri nets.

In this chapter, we limit ourselves to deal with finite state systems and with regular predicates defined on sequences of transition labels. We have thus all cards in hands to decide opacity, even though several pairs *(observer, secret)* are taken into simultaneous account. Now differing from [47], we want to be able to *enforce* opacity by supervisory control when the result of the decision is negative. In other terms, we want to disable the least possible family of trajectories such that no observer can ever find out that the system's actual trajectory belongs to some secret. At first sight, this looks like a simple problem, all the more when it is assumed that all events are controllable as we do in this chapter (we leave the uncontrollable events to further consideration). The problem is in fact not that simple, for the observers have full knowledge of the system, hence any control device that may be added to the system is known to them. We will nevertheless show that there exists always an optimal control for enforcing the concurrent secrets on opponents, fully aware of this control. We will also provide techniques for computing this optimal control under assumptions that fit at least with some applications.

The rest of the chapter is organized as follows. The notation and the problem are introduced in section 7.2. Section 7.3 shows that a unique optimal solution always exists, but it is generally not regular. Using the fixpoint characterization of the optimal control, proofs of control enabledness of trajectories are presented as infinite trees in section 7.4; conditions on proof trees entailing the regularity of the optimal control are also stated there. Section 7.5 produces closely connected conditions on concurrent secrets. An application is sketched in section 7.6, where directions for further work are also suggested.

7.2 Secrets, concurrent secrets, and the control problem

To begin with, let us fix the notation. Σ is a finite alphabet, Σ^* is the free monoid generated by Σ, and $Rat(\Sigma^*)$ is the family of rational subsets of Σ^* i.e. the family of regular languages over Σ. Let uv denote the concatenation product of the words u and v, thus u is a prefix of uv and the empty word ε is a prefix of every word. The length of u is denoted by $|u|$. For $l \leq |u|$, $u[l]$ denotes the prefix of u with the length l, and for $0 < l \leq |u|$, $u(l)$ denotes the l^{th} letter occurring in u. For any sub-alphabet $\Sigma_i \subseteq \Sigma$, let $\pi_i : \Sigma^* \to \Sigma_i^*$ be the unique monoid morphism extending the map $\pi_i(\sigma) = \sigma$ if $\sigma \in \Sigma_i$ else ε (letters $\sigma \in \Sigma$ are mapped to words by the usual embedding of Σ into Σ^*). For $u, v \in \Sigma^*$, let $u \simeq_i v$ if $\pi_i(u) = \pi_i(v)$. Throughout the chapter, L is a non-empty prefix-closed language in $Rat(\Sigma^*)$ and for all $i \in \{1, \cdots, n\}$, $\Sigma_i \subseteq \Sigma$, $S_i \in Rat(\Sigma^*)$, and $S_i \subseteq L$.

The language L represents the behaviour of a system with n users. For $i \in \{1, \cdots, n\}$, the sub-alphabet Σ_i represents the set of the interactions that may take place between the system and the user i. Users observe the system by interacting with it. If the system's trajectory is represented by $w \in L$, then the induced observation for the user i is $\pi_i(w)$. Two users can communicate only by jointly interacting with the system, *e.g.* $\sigma \in \Sigma_i \cap \Sigma_j$ is an interaction of the system with the users i and j.

For each $i \in \{1, \cdots, n\}$, the membership of the actual system's trajectory to the subset $S_i \subseteq L$ is intended to be kept *secret* from the user i. In the terminology of [133] and [47], the predicate S_i should be *opaque* w.r.t. the observation function π_i and the language L.

Definition 7.2.1 *S_i is* opaque *w.r.t. π_i (and L) if* $(\forall w \in S_i)\,(\exists w' \in L \setminus S_i)\ w \simeq_i w'$

When the predicate S_i coincides with its prefix closure $\overline{S_i}$, non-opacity is the same as normality which may be expressed as $\forall w \in \overline{S_i}\ \forall w' \in L\ w \simeq_i w' \Rightarrow w' \in \overline{S_i}$. However, opacity is not the opposite of normality, as the following example shows. Given $L = (ab)^* + (ab)^*a$ let $\Sigma_i = \{b\}$ and $S_i = (ab)^*a$ then S_i is both opaque and normal.

As we explained in the introduction, we use here a strongly restricted form of the original definition of opacity where the observation functions may be state and history dependent. On the other hand, we consider a concurrent version of opacity.

Definition 7.2.2 *$(S_i)_i$ is* concurrently opaque *(w.r.t. L) if for all i, S_i is opaque w.r.t. π_i.*

Dealing with concurrent opacity does not make a big change for checking opacity, which is easy in our case (although not necessarily computationally simple) since we consider exclusively regular systems and secrets.

Proposition 1 *It is decidable whether $(S_i)_i$ is concurrently opaque.*

Proof: By definition, it suffices to decide for each $i \in \{1, \ldots, n\}$ whether S_i is opaque w.r.t. π_i. The considered property holds if and only if $\pi_i(S_i) \subseteq \pi_i(L \setminus S_i)$. As L and S_i are regular, $L \setminus S_i$ is regular, and since morphic images of regular languages are regular, this relation can be decided. □

Example 7.2.3 *Let $\Sigma = \{a, b, c\}$ and L be the set of prefixes of words in $(a + b)\, c$. Let $\Sigma_1 = \Sigma_2 = \{c\}$, and let S_1 and S_2 be the intersections of L with $\Sigma^* \, a \, \Sigma^*$ and $\Sigma^* \, b \, \Sigma^*$, respectively. The concurrent secret (S_1, S_2) is opaque. From the observation of the event c, one is indeed unable to infer whether it was preceded with an a or with a b.*

In the sequel, $\mathcal{S} = \{(\Sigma_1, S_1), \ldots, (\Sigma_n, S_n)\}$ denotes a concurrent secret upon a fixed language $L \subseteq \Sigma^*$ ($\Sigma_i \subseteq \Sigma$ and $S_i \subseteq L \subseteq \Sigma^*$ for all i). We say that $\mathcal{S}$ is opaque if $(S_i)_i$ is concurrently opaque. A control is any non-empty prefix-closed language $L' \subseteq L$ (we assume here that all events $\sigma \in \Sigma$ are controllable). We say that $\mathcal{S}$ is opaque under the control $L' \subseteq L$ if the induced secret $(S'_i)_i$ defined with $S'_i = S_i \cap L'$ is concurrently opaque w.r.t. L'.

Our purpose is to solve the concurrent opacity control problem stated as follows.

Problem 1 *Show that the set of controls enforcing the opacity of $\mathcal{S}$ either is empty or has a greatest element, and compute this maximal permissive control.*

Enforcing concurrent opacity ($n > 1$) requires, as we shall see, significantly more efforts than enforcing opacity.

7.3 Maximal permissive control enforcing concurrent opacity

In this section, we show that the concurrent opacity control problem has a unique maximal solution that we characterize as a greatest fixpoint. We propose two counter-examples in which this maximal permissive control either is not regular or cannot be computed within a finite number of fixpoint iterations.

Definition 7.3.1 *For any prefix-closed subset L' of L, the* safe kernel *of L' w.r.t. the secret $\mathcal{S}$, notation $K(L', \mathcal{S})$, is the subset of all words $w \in L'$ such that $w = uv \Rightarrow (\forall i)(\exists u' \in L' \setminus S_i)\ u \simeq_i u'$.*

Thus, $\mathcal{S}$ is opaque under the control $L' \subseteq L$ if and only if $L' = K(L', \mathcal{S})$, *i.e.* L' is a fixpoint of $K(\bullet, \mathcal{S})$. It is immediately observed that $K(L', \mathcal{S})$ is continuous in the first argument (w.r.t. set inclusion). As the prefix-closed subsets of L form a complete sub-lattice of $\mathcal{P}(\Sigma^*)$, it follows from Knaster-Tarski's theorem [171] that $K(\bullet, \mathcal{S})$ has a greatest fixpoint in this sub-lattice.

Definition 7.3.2 *Let $SupK(L, \mathcal{S})$ be the greatest fixed point of the operator $K(\bullet, \mathcal{S})$.*

Proposition 2 *$SupK(L, \mathcal{S})$ is the union of all controls enforcing the opacity of $\mathcal{S}$. If $SupK(L, \mathcal{S}) \neq \emptyset$, then it is the maximal permissive control enforcing the opacity of $\mathcal{S}$, otherwise no such control can exist.*

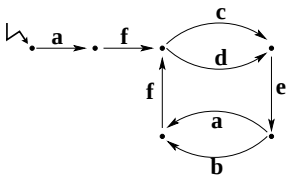


Figure 7.1: An automaton

Proof: This is a direct application of the Knaster-Tarski's fixpoint theorem. □

Remark 7.3.3 *The condition $L' \subseteq SupK(L,\mathcal{S})$ is necessary* but not sufficient *for some non-empty control L' to enforce the opacity of $\mathcal{S}$. For instance, in Example 7.2.3, $SupK(L,\mathcal{S}) = L$, but the secret S_1 is not opaque w.r.t. $L' = \varepsilon + a + ac$.*

The fixpoint characterization of the optimal control enforcing opacity does not show that $SupK(L,\mathcal{S})$ can be computed, nor that the control can be implemented with a finite device. When $n = 1$, *i.e.* when $\mathcal{S} = \{(\Sigma_1, S_1)\}$, this is not a problem because in this particular case, $SupK(L,\mathcal{S})$ is equal to $K(L,\mathcal{S})$ and it may be shown that $K(L,\mathcal{S})$ is the set of words with all prefixes in $L \cap \pi_1^{-1}(L \setminus S_1)$. Therefore, $SupK(L,\mathcal{S}) = \Sigma^* \setminus ((\Sigma^* \setminus (L \cap \pi_1^{-1}(L \setminus S_1))\,\Sigma^*)$ which is regular. When $n > 1$, two situations contrast. The nice situation is when $SupK(L,\mathcal{S})$ can be computed from L by a finite number of iterated applications of the operator $K(\bullet,\mathcal{S})$. Actually, when L' is a regular subset of L, the same holds for $K(L',\mathcal{S})$, hence in the considered case $SupK(L,\mathcal{S})$ is regular. The rest of the section illustrates the converse situation.

7.3.1 A case where the closure ordinal of $K(\bullet,\mathcal{S})$ is transfinite

Let $\Sigma = \{a,b,c,d,e,f\}$ and let L be the prefix-closed language accepted by the finite automaton of Figure 7.1 (where all states are accepting states). Define $\mathcal{S} = \{(\Sigma_1, S_1), (\Sigma_2, S_2)\}$ with $\Sigma_1 = \{c,\ f\}$, $S_1 = \Sigma^* afc\,(\Sigma \setminus \{c\})^*$ (this secret is safe if, by observing only c and f, one cannot find out in any run that the last occurrence of c was preceded by af), and $\Sigma_2 = \{b,\ e\}$, $S_2 = \Sigma^* deb\,(\Sigma \setminus \{b\})^*$ (this secret is safe if, by observing only b and e, one cannot find out in any run that the last occurrence of b was preceded by de). Let $L_1 = K(L,\mathcal{S})$ be the first language encountered in the greatest fixpoint iteration converging to $SupK(L,\mathcal{S})$, then $L_1 = L \setminus afc\Sigma^*$ (the run afc reveals the secret S_1 and the runs in $afd\Sigma^*$ reveal nothing). The second item $L_2 = K(L_1,\mathcal{S})$ is the language $L_1 \setminus afdeb\Sigma^*$ (relatively to L_1, the run $afdeb$ reveals the secret S_2, and the runs in $afdea\Sigma^*$ reveal nothing). After afc and $afdeb$ have been eliminated, the initial situation reproduces up to the prefix $afdea$. Therefore, the fixpoint iteration produces a strictly decreasing and infinite sequence of languages L_j. The limit $SupK(L,\mathcal{S})$ of this decreasing chain is the set of all prefixes of words in the regular set $L_\omega = (afde)^*$, hence it is regular. The optimal control enforcing the opacity of $\mathcal{S}$ may be implemented by any finite automaton recognizing L_ω.

Let us now extend the concurrent secret into $\mathcal{S} = \{(\Sigma_1, S_1), (\Sigma_2, S_2), (\Sigma_3, S_3)\}$ with (Σ_1, S_1) and (Σ_2, S_2) as above, $\Sigma_3 = \emptyset$ and $S_3 = \Sigma^* \setminus (\Sigma^* c \Sigma^*)$. Then, the closure ordinal of $K(\bullet, \mathcal{S})$ increases from ω to $\omega + 1$. To see this observe that, since Σ_3 is empty, the secret S_3 is safe relatively to any language $L' \subseteq L$ containing at least one word containing at least one occurrence of c. The greatest fixpoint iteration for $SupK(L, \mathcal{S})$ starts with the same decreasing sequence L_j as before, but $K(L_\omega, \mathcal{S})$ differs now from L_ω because L_ω contains no word containing c (differing in that form all L_j). In fact, $L_{\omega+1} = K(L_\omega, \mathcal{S}) = \emptyset$ and this is a fixpoint. Opacity can therefore not be enforced.

7.3.2 A case where $SupK(L, \mathcal{S})$ is not regular

Let $\Sigma = \{a, b, x, y\}$ and L be the set of prefixes of words in $(ax)^* (\varepsilon + ab) (yb)^*$. Define $\mathcal{S} = \{(\Sigma_i, S_i) \mid 1 \leq i \leq 3\}$ as follows (letting $\neg L' = L \setminus L'$ for $L' \subseteq L$):

1. $\Sigma_1 = \{a, b\}$, $\neg S_1 = \varepsilon + (ax)^* ab (yb)^* + (\Sigma \setminus \{b\})^*$
2. $\Sigma_2 = \{x, y\}$, $\neg S_2 = (ax)^* (yb)^*$
3. $\Sigma_3 = \{a, b, x, y\}$, $\neg S_3 = \varepsilon + a\Sigma^*$

We claim that $SupK(L, \mathcal{S})$ is not a regular language and worse, the family of regular controls enforcing the opacity of $\mathcal{S}$ has no largest element. Recall that the subset of maximal words in a regular language is regular. In order to establish the first part of the claim, one can show that $SupK(L, \mathcal{S})$ is equal to the set L' of all prefixes of words in the non regular language $\cup_{n \in \mathbb{N}} (ax)^n (\varepsilon + ab) (yb)^n$. A detailed proof of this fact may be found in [7].

To show that the family of regular controls enforcing the opacity of $\mathcal{S}$ has no largest element, one assumes the opposite. Let R be the largest prefix-closed regular subset of L such that $\mathcal{S}$ is opaque w.r.t. R. Necessarily, $(ax)^n (yb)^n \notin R$ for some n. If it were otherwise, because $(ax)^{n-1} (yb)^{n-1}$ is the sole word $w' \in L' \setminus S_1$ such that $w \simeq_1 w'$ for $w = (ax)^n (yb)^n$, R would coincide with L', which is not possible (L' is not regular). Let n be the least integer such that $(ax)^n (yb)^n \notin R$, and let R' be R augmented with all prefixes of words in $\{(ax)^n (yb)^n, (ax)^{n-1} ab (yb)^{n-1}\}$ not already in R. The language R' is prefix-closed and regular, and one can verify that $\mathcal{S}$ is opaque w.r.t. R'. Thus, a contradiction has been reached.

7.4 Control enabling and ω-trees

Warning 1 *From now on, $S_i \Sigma^* \subseteq S_i$ is imposed on all sets S_i in $\mathcal{S} = \{(\Sigma_1, S_1), \ldots, (\Sigma_n, S_n)\}$.*

This section serves as a bridge between the general problem and the practical solutions that we shall propose in specific cases. The working assumption that secrets are suffix-closed is motivated by its convenience (if not its necessity) for enforcing opacity with finite control. Although this working assumption was not satisfied in the examples from sections 7.3.1 and 7.3.2, it is quite natural since it amounts to strengthening the secrecy requirement as follows: an observer i should

never have the knowledge that the trajectory of the system is in S_i *or was* in S_i at some instant in the past. We give below a simpler definition of the operator $K(\bullet, \mathcal{S})$, which is equivalent to the earlier definition when secrets are suffix-closed. Then we consider ω-trees that may be seen as proofs of control enabledness of trajectories. Finally, we propose conditions on sets of proof trees entailing the regularity of $SupK(L, \mathcal{S})$, thus paving the way for section 7.5.

Definition 7.4.1 (modified form of Def. 7.3.1) *For any prefix-closed subset L' of L, the* safe kernel *of L' w.r.t. the secret $\mathcal{S}$, notation $K(L', \mathcal{S})$, is the largest subset of L' such that $w \in K(L', \mathcal{S}) \Rightarrow (\forall i)(\exists w' \in L' \setminus S_i)\ w \simeq_i w'$.*

Proposition 3 *Definitions 7.3.1 and 7.4.1 are equivalent.*

Proof: For the duration of this proof, let $K(\bullet, \mathcal{S})$ and $K'(\bullet, \mathcal{S})$ be the two operators from Def. 7.3.1 and Def. 7.4.1, respectively. Clearly, $K(L', \mathcal{S}) \subseteq K'(L', \mathcal{S})$ for any L'. We show the converse relation. Consider any word $w \in K'(L', \mathcal{S})$ and let $w = uv$ be any decomposition of this word into two factors. We should prove that for all $i \in \{1, \ldots, n\}$, $u \simeq_i u'$ for some $u' \in L' \setminus S_i$. As $w \in K'(L', \mathcal{S})$ and by definition, $w \simeq_i w'$ for some $w' \in L' \setminus S_i$. Now $w' \simeq_i uv$, hence there exists at least one decomposition $w' = u'v'$ such that $u \simeq_i u'$. Finally, $u' \in L'$ by prefix-closedness of L', and $u' \notin S_i$ by suffix-closedness of S_i. Therefore, $w \in K(L', \mathcal{S})$. □

Definition 7.4.2 *Given $w \in L$, a* proof *of* enabledness *of w is a map $f : \{1, \ldots, n\}^* \to L$ such that $f(\varepsilon) = w$ and for all $\tau \in \{1, \ldots, n\}^*$ and $j \in \{1, \ldots, n\}$, $f(\tau) \simeq_j f(\tau j)$ and $f(\tau j) \notin S_j$.*

The map f in the above definition is just a complete n-ary ordered tree labelled on nodes, thus in particular it is an infinite tree. The next proposition follows immediately from the co-inductive definition of $SupK(L, \mathcal{S})$.

Proposition 4 *For any $w \in L$, $w \in SupK(L, \mathcal{S})$ if and only if there exists a proof of the control enabledness of w.*

A nice situation is when the control enabledness of a trajectory may be proved with a regular tree. Let us recall the definition.

Definition 7.4.3 *Let $f : \{1, \ldots, n\}^* \to L$ be a (complete n-ary ordered labelled) tree. For any $\tau \in \{1, \ldots, n\}^*$, the sub-tree of f rooted at τ, in notation f/τ, is the (complete n-ary ordered labelled) tree defined with $(f/\tau)(\tau') = f(\tau\tau')$ for all $\tau' \in \{1, \ldots, n\}^*$. The tree f is regular if it has a finite number of sub-trees f/τ.*

Any regular tree f may be folded to a finite rooted graph. When the control enabledness of the (good) trajectories may be proved using regular trees exclusively, this predicate is therefore recursively enumerable. This condition is necessary and sufficient for being able to enforce control,

but not efficiently. In the rest of the section, we search for additional conditions entailing the regularity of the control $SupK(L, \mathcal{S})$.

A first attempt towards this goal is to impose an upper bound on the number of (different) subtrees of a regular proof tree. Equivalently, one may require that all proof trees conform to a finite collection of finite patterns as follows.

Definition 7.4.4 *A* finite pattern for proofs *(of enabledness of trajectories) is a finite, deterministic and complete automaton* $(Q, \{1, \ldots, n\}, q_0)$ *(thus* $q_0 \in Q$ *and any* $i \in \{1, \ldots, n\}$ *maps* Q *to itself). A proof tree* $f : \{1, \ldots, n\}^* \to L$ conforms to *a finite pattern if there exists a labelling map* $\lambda : Q \to L$ *such that* $f(\tau) = \lambda(q_0 \cdot \tau)$ *for all* $\tau \in \{1, \ldots, n\}^*$ *letting* $q \cdot \tau$ *be defined inductively with* $q \cdot \varepsilon = q$ *and* $q \cdot (\tau_1 \tau_2) = (q \cdot \tau_1) \cdot \tau_2$ *for all* $q \in Q$.

The idea behind this definition is that proof trees contain bounded information up to the choice of a bounded number of words in L.

Example 7.4.5 *Let* $\Sigma = \{a, b\}$ *and* $L = \Sigma^*$. *Let* $\mathcal{S} = \{(\Sigma_1, S_1), (\Sigma_2, S_2)\}$ *with* $\Sigma_1 = \{a\}$, $\neg S_1 = b^*a^*$ *and* $\Sigma_2 = \{b\}$, $\neg S_2 = a^*b^*$. *The finite pattern shown on Figure 7.2 supplies*

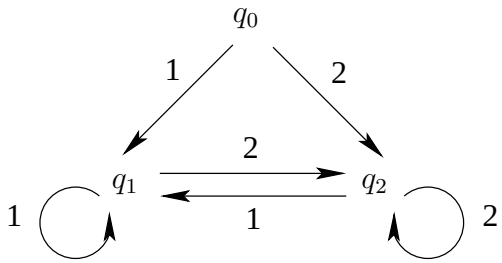


Figure 7.2

proofs of control enabledness for all trajectories. For any word w *with* n *occurrences of* a *and* m *occurrences of* b, *the labelling map defined with* $\lambda(q_0) = w$, $\lambda(q_1) = b^m a^n$, *and* $\lambda(q_2) = a^n b^m$ *induces in fact an* ω*-tree witnessing that* $w \in SupK(L, \mathcal{S})$.

There are two sources of problems with the proof patterns from Def. 7.4.4. The first difficulty is that, given L, $\mathcal{S}$ and $(Q, \{1, \ldots, n\}, q_0)$, the set of the labelling maps $\lambda : Q \to L$ considered in this definition is generally not regular, *i.e.* it cannot be defined with a finite automaton on $(\Sigma^*)^{|Q|}$. For instance, if the labelling maps considered in example 7.4.5 did form a regular set, then the set of all pairs $(b^m a^n, a^n b^m)$ would be regular, but the iteration lemma for rational sets [37] entails the opposite (if the set is regular, for some $N > 1$ and for large enough n and m, $(b^m a^n, a^n b^m)$ could be written as $(x, x')(y, y')(z, z')$ where $0 < |y|+|y'|$, $|x|+|x'|+|y|+|y'| \leq N$, and $(x, x')(y, y')^*(z, z')$ is included in the set). The second difficulty is that, given L, $\mathcal{S}$ and $(Q, \{1, \ldots, n\}, q_0)$, the set of values taken at $q = q_0$ by the labelling maps from Def. 7.4.4 is sometimes not regular. An example is shown hereafter.

Example 7.4.6 *Let* $\Sigma = \{a, b\}$ *and* $L = \Sigma^*$. *Let* $\mathcal{S} = \{(\Sigma_1, S_1), (\Sigma_2, S_2)\}$ *where* $\Sigma_1 = \{a\}$, $\Sigma_2 = \{b\}$, *and* $\neg S_1 = \neg S_2 = (\varepsilon + b)(ab)^*(\varepsilon + a)$. *Consider the set of all maps labelling adequately the finite proof pattern from Figure 7.3. The set of values taken by these maps at* $q = q_0$ *is the set of all words in which the numbers of occurrences* a *and* b *differ by at most one, hence it is not regular.*

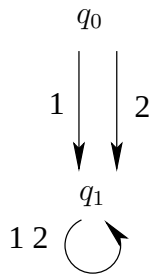


Figure 7.3

Note that in both examples 7.4.5 and 7.4.6, $SupK(L, \mathcal{S}) = \Sigma^*$, and proofs of enabledness may be obtained for all $w \in \Sigma^*$ by labelling the finite proof pattern shown in Figure 7.4.

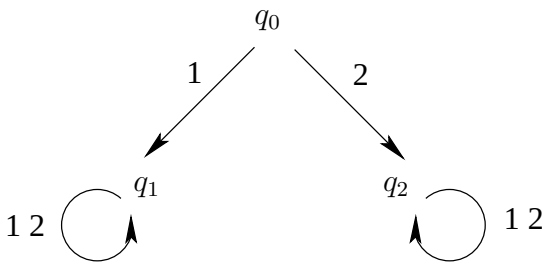


Figure 7.4

In order to dodge the problems, one may concentrate on restricted proof patterns as follows.

Definition 7.4.7 *A* type *(of proof of enabledness) is a finite pattern* $\mathcal{T} = (Q, \{1, \ldots, n\}, q_0)$ *with a prefix-closed subset* $T \subseteq \{1, \ldots, n\}^*$ *such that* $(\forall q \in Q)\,(\exists!\, \tau \in T)\,(q = q_0 \cdot \tau)$ *and for any map* $\lambda : Q \to L$,
$(\forall \tau)\,(\forall j)\ (\tau j \in T \wedge \lambda(q_0 \cdot \tau) \simeq_j \lambda(q_0 \cdot \tau j) \wedge \lambda(q_0 \cdot \tau j) \notin S_j) \ \Rightarrow (\forall q)\,(\forall j)\ (\lambda(q) \simeq_j \lambda(q \cdot j)$
$\wedge\, \lambda(q \cdot j) \notin S_j)$
where τ *and* j *range over* $\{1, \ldots, n\}^*$ *resp. over* $\{1, \ldots, n\}$. *A proof tree* $f : \{1, \ldots, n\}^* \to L$ *has type* $\mathcal{T}$ *if it conforms to this pattern (see Def. 7.4.4).*

The set T in Def. 7.4.7 induces a (finite) tree, rooted at q_0, that spans the automaton $(Q, \{1, \ldots, n\}, q_0)$. The point is that for any map $\lambda : Q \to L$, if $(\lambda(q) \simeq_j \lambda(q \cdot j) \wedge \lambda(q \cdot j) \notin S_j)$ for all arcs $(q, q \cdot j)$

in the spanning tree, then it holds also for all chords, *i.e.* for all remaining edges of (the underlying graph of) $(Q, \{1, \ldots, n\}, q_0)$.

Theorem 7.4.8 *If there exists a finite number of types of proofs of enabledness for all trajectories $w \in SupK(L, \mathcal{S})$, then $SupK(L, \mathcal{S})$ is a regular language.*

Proof: It suffices to show that when type $\mathcal{T} = (Q, \{1, \ldots, n\}, q_0, T)$ has been fixed, the set of trajectories $w \in L$ with proofs of enabledness of type $\mathcal{T}$ is regular. In view of the definitions 7.4.4 and 7.4.7, a word w belongs to the considered set if and only if $\lambda(q_0) = w$ for some map $\lambda : Q \to L$ satisfying $\lambda(q_0 \cdot \tau) \simeq_j \lambda(q_0 \cdot \tau j)$ and $\lambda(q_0 \cdot \tau j) \notin S_j$ whenever $\tau j \in T$ and $j \in \{1, \ldots, n\}$. In order to show that this is a regular set, we construct the Arnold-Nivat product [4] of a family of automata $\mathcal{A}_\tau$ indexed with $\tau \in T$, as follows. Let $\mathcal{A}_\varepsilon$ be a (finite deterministic) partial automaton recognizing L, and for each sequence τj in T with $j \in \{1, \ldots, n\}$, let $\mathcal{A}_{\tau j}$ be a (finite deterministic) partial automaton recognizing $L \setminus S_j$. This defines the components of the product. As for the synchronizations, let $\mathcal{V}$ be the set of T-vectors $\vec{v} : T \to (\Sigma \cup \{\varepsilon\})$ such that $(\vec{v}(\tau) \in \Sigma_j \vee \vec{v}(\tau j) \in \Sigma_j) \Rightarrow \vec{v}(\tau) = \vec{v}(\tau j)$ whenever τj in T and $j \in \{1, \ldots, n\}$. The induced product is a (finite deterministic) partial automaton $\mathcal{A} = (\mathcal{Q}, \mathcal{V}, \vec{q}_0)$ defined as follows:
- the set of states $\mathcal{Q}$ is a set of T-vectors,
- for each $\tau \in T$, $\vec{q}_0(\tau)$ is the initial state of $\mathcal{A}_\tau$,
- for all $\vec{q} \in \mathcal{Q}$ and $\tau \in T$, $\vec{q}(\tau)$ is a state of $\mathcal{A}_\tau$,
- for all $\vec{q} \in \mathcal{Q}$, $\vec{v} \in \mathcal{V}$ and $\tau \in T$, $(\vec{q} \cdot \vec{v})(\tau) = \vec{q}(\tau) \cdot \vec{v}(\tau)$.

Therefore, $\vec{q} \cdot \vec{v}$ is defined if and only if $\vec{q}(\tau) \cdot \vec{v}(\tau) = \vec{q'}(\tau)$ is defined for all τ.

Let $\vec{v}_1 \ldots \vec{v}_m$ be a word over $\mathcal{V}$ accepted by $\mathcal{A}$. An associated T-vector $\vec{w} : T \to L$ may be defined by setting $\vec{w}(\tau) = \vec{v}_1(\tau) \ldots \vec{v}_m(\tau)$ for all $\tau \in T$. It follows directly from the construction that the map $\lambda : Q \to L$ such that $\lambda(q_0 \cdot \tau) = \vec{w}(\tau)$ for all $\tau \in T$ satisfies $\lambda(q_0 \cdot \tau) \simeq_j \lambda(q_0 \cdot \tau j)$ and $\lambda(q_0 \cdot \tau j) \notin S_j$ for $\tau j \in T$ and $j \in \{1, \ldots, n\}$, hence $\vec{w}(\varepsilon) \in SupK(L, \mathcal{S})$.

As $\mathcal{A}$ is a finite automaton, the projection of the language of $\mathcal{A}$ along ε is a regular language. In order to complete the proof, it suffices therefore to show that for any map $\lambda : Q \to L$ satisfying $\lambda(q_0 \cdot \tau) \simeq_j \lambda(q_0 \cdot \tau j)$ and $\lambda(q_0 \cdot \tau j) \notin S_j$ for all $\tau j \in T$, the vector $\vec{w} : T \to L$ defined with $\vec{w}(\tau) = \lambda(q_0 \cdot \tau)$ for all $\tau \in T$ may be written as a word $\vec{v}_1 \ldots \vec{v}_m$ recognized by $\mathcal{A}$. Given the construction of this automaton, it suffices to exhibit a sequence $\vec{v}_1 \ldots \vec{v}_m \in \mathcal{V}^*$ such that $\vec{w}(\tau) = \vec{v}_1(\tau) \ldots \vec{v}_m(\tau)$ for all $\tau \in T$. This is the contribution of the lemma 7.4.9. □

Lemma 7.4.9 *Let $\vec{w} : T \to \Sigma^*$ where T is a prefix-closed subset of $\{1, \ldots, n\}^*$ and $\vec{w}(\tau) \simeq_j \vec{w}(\tau j)$ for all $\tau j \in T$ with $j \in \{1, \ldots, n\}$. Then $\vec{w}(\tau) = \vec{v}_1(\tau) \ldots \vec{v}_m(\tau)$ for all $\tau \in T$ for some sequence of vectors $\vec{v}_k : T \to \Sigma \cup \{\varepsilon\}$ such that for all $\tau j \in T$, $(\vec{v}_k(\tau) \in \Sigma_j \vee \vec{v}_k(\tau j) \in \Sigma_j) \Rightarrow \vec{v}_k(\tau) = \vec{v}_k(\tau j)$.*

Proof: Let $\mathcal{E}$ be the set of all pairs (τ, i) such that $\tau \in T$ and $0 < i \leq |\vec{w}(\tau)|$. Let $<$ be the partial order on $\mathcal{E}$ defined with $(\tau, i) < (\tau', i')$ if $\tau = \tau'$ and $i < i'$. For each $j \in \{1, \ldots, n\}$, let

$(\tau, i) \Vdash_j (\tau j, k)$ if $\pi_j(\vec{w}(\tau)[i])) = \pi_j(\vec{w}(\tau j)[k]))$, $\vec{w}(\tau)(i) = \vec{w}(\tau j)(k)$, and this letter is in Σ_j. Let $\equiv$ denote the equivalence on $\mathcal{E}$ generated from the union of the relations $\Vdash_j$. We claim that this equivalence does not intersect and is compatible with the partial order $<$. Let us establish this double claim.

i) Suppose for a contradiction that $(\tau, i) < (\tau, i')$ and $(\tau, i) \equiv (\tau, i')$. Then, by definition of $\equiv$ and the relations $\Vdash_j$, the words $\vec{w}(\tau)[i]$ and $\vec{w}(\tau)[i']$ end with a common letter $\vec{w}(\tau)(i) = \vec{w}(\tau)(i')$, and this letter occurs the same number of times in both words. As $i < i'$, this is clearly not possible.

ii) Suppose for a contradiction that $(\tau, i) < (\tau, i')$ and $(\tau', j) < (\tau', j')$ while $(\tau, i) \equiv (\tau', j')$ and $(\tau, i') \equiv (\tau', j)$. Then, by definition of $\equiv$ and the relations $\Vdash_j$, $\vec{w}(\tau)(i) = \vec{w}(\tau')(j')$ and this letter σ occurs the same number of times in both words $\vec{w}(\tau)[i]$ and $\vec{w}(\tau')[j']$. In the same way, $\vec{w}(\tau)(i') = \vec{w}(\tau')(j)$ and this letter σ' occurs the same number of times in both words $\vec{w}(\tau)[i']$ and $\vec{w}(\tau')[j]$. Since $i < i'$ and $j < j'$, it follows that σ and σ' are different letters (see Figure 7.5).

Now let $\tau = \rho x_1 \dots x_k$ and $\tau' = \rho y_1 \dots y_l$ where ρ is the longest common prefix of τ and τ' and $x_h, y_h \in \{1, \dots, n\}$. Then by definition of $\equiv$ and the relations $\Vdash_j$, $(\tau, i) \equiv (\tau', j')$ and $(\tau, i') \equiv (\tau', j)$ entail that σ and σ' belong jointly to all the alphabets Σ_{x_h} $(1 \leq h \leq k)$ and Σ_{y_h} $(1 \leq h \leq l)$. On the other hand, by the *i* part of the proof, $(\tau, i) \equiv (\tau', j')$ entails that necessarily $\vec{w}(\tau)[i] \Vdash_{x_k}^{-1} \circ \dots \circ \Vdash_{x_1}^{-1} \circ \Vdash_{y_1} \circ \dots \circ \Vdash_{y_l} \vec{w}(\tau')[j']$. Therefore the words $\vec{w}(\tau)[i]$ and $\vec{w}(\tau')[j']$ must have the same number of occurrences of the letter σ', which is obviously not possible.

Let $\mathcal{C} = (\mathcal{E}/\equiv)$. Since $<$ is compatible and does not intersect with $\equiv$, the binary relation $(< \cup \equiv)^*/\equiv$ is a strict partial order on $\mathcal{C}$. Let $C_1 \dots C_m$ be an enumeration of $\mathcal{C}$ compatible with this order. Each equivalence class $C \in \mathcal{C}$ induces naturally a vector $\vec{v} \in \mathcal{V}$, *viz.* $\vec{v}(\tau) = \vec{w}(\tau)(i)$ if $(\tau, i) \in C$ for some i, or ε otherwise. Let $\vec{v}_1 \dots \vec{v}_m$ be the vectors associated with $C_1 \dots C_m$, respectively. Then for any $\tau \in T$, $\vec{w}(\tau) = \vec{v}_1(\tau) \dots \vec{v}_m(\tau)$ as desired. ⌟ □

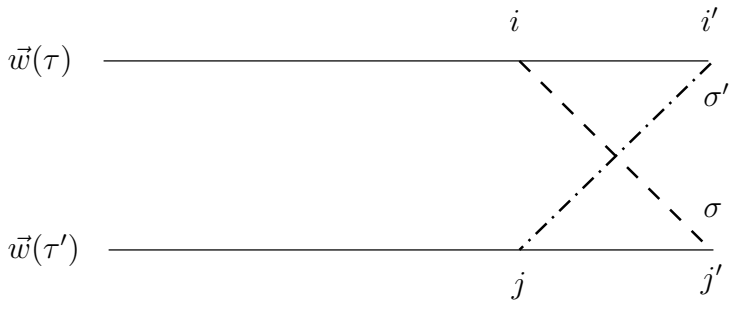


Figure 7.5

Theorem 7.4.8 opens the way to the practical synthesis of supervisory control for concurrent opacity. The conditions for its application are examined further in section 7.5.

7.5 Concurrent secrets with regular opacity control

We propose here conditions on concurrent secrets $\mathcal{S} = \{(\Sigma_1, S_1), \ldots, (\Sigma_n, S_n)\}$ ensuring that the maximal permissive opacity control $SupK(L, \mathcal{S})$ is the language of a finite automaton, that may effectively be constructed from finite automata accepting the language L and the secrets S_i. We examine first the case where the alphabets Σ_i form a chain for the inclusion, second the case where the secrets S_i form a chain for the inclusion, third the case where every secret S_i is saturated by any equivalence $\simeq_j$ such that $i \neq j$ (a set is *saturated* by an equivalence if it is a union of equivalence classes). We consider finally the combinations of the three cases for the different pairs (i, j).

Proposition 5 *If the alphabets Σ_i form a chain for the inclusion, then the enabledness of all trajectories $w \in SupK(L, \mathcal{S})$ may be shown with a single type of proofs $\mathcal{T}_1$.*

Proof: Given the chain $\Sigma_1 \subseteq \Sigma_2 \subseteq \ldots \subseteq \Sigma_n$, we construct a type $\mathcal{T}_1 = (Q, \{1, \ldots, n\}, q_0, T)$ as follows. T is the set of strictly increasing sequences of numbers in $\{1, \ldots, n\}$ (T is drawn with solid arcs in Figure 7.6), $Q = T$ and $q_0 = \varepsilon$. For any τ in T and $i \in \{1, \ldots, n\}$, $\tau \cdot i = \tau' i$ where τ' is the largest prefix of τ formed of integers strictly smaller than i (see again Figure 7.6). As $(\simeq_j \circ \simeq_i) \subseteq \simeq_i$ for $i \leq j$, $\mathcal{T}_1$ conforms to Def. 7.4.7. Finally, for any $w \in SupK(L, \mathcal{S})$, by Prop. 7.4.8, there must exist a map $\lambda : Q \to L$, *i.e.* $\lambda : T \to L$, such that $\lambda(\varepsilon) = w$ and for all $\tau j \in T$, $\lambda(\tau) \simeq_j \lambda(\tau j) \wedge \lambda(\tau j) \notin S_j$. □

Proposition 6 *If the secrets S_i form a chain for the inclusion, then the enabledness of all trajectories $w \in SupK(L, \mathcal{S})$ may be shown with a single type of proofs $\mathcal{T}_2$.*

Proof: Given the chain $S_1 \subseteq S_2 \subseteq \ldots \subseteq S_n$, we construct a type $\mathcal{T}_2 = (Q, \{1, \ldots, n\}, q_0, T)$ as follows. T is the set of strictly increasing sequences of numbers in $\{1, \ldots, n\}$ (T is drawn with solid arcs in Figure 7.7), $Q = T$ and $q_0 = \varepsilon$. For any τ in T and $i \in \{1, \ldots, n\}$, $\tau \cdot i = \tau i$ if $\tau i \in T$ and $\tau \cdot i = \tau$ otherwise (see again Figure 7.7). If $i \leq j$, then for any τj in T ($= Q$), and for any map $\lambda : Q \to L$, $\lambda(\tau j) \notin S_j \Rightarrow \lambda(\tau j \cdot i) \notin S_i$ since $S_i \subseteq S_j$. Therefore, $\mathcal{T}_2$ conforms to Def. 7.4.7, and the desired conclusion follows from Prop. 7.4.8. □

Proposition 7 *If for all distinct $i, j \in \{1, \ldots, n\}$, the secret S_i is saturated by the equivalence relation $\simeq_j$, then the enabledness of all trajectories $w \in SupK(L, \mathcal{S})$ may be shown with a type of proofs $\mathcal{T}_3$.*

Proof: We construct a type $\mathcal{T}_3 = (Q, \{1, \ldots, n\}, q_0, T)$ as follows. T is the set of sequences in $\{1, \ldots, n\}^*$ with at most one occurrence of each number (T is drawn with solid arcs in Figure 7.8), $Q = T$ and $q_0 = \varepsilon$. For any τ in T and $i \in \{1, \ldots, n\}$, $\tau \cdot i = \tau i$ if $\tau i \in T$ and $\tau \cdot i = \tau$ otherwise (see again Figure 7.8). Let $\lambda : Q \to L$ be any map such that $\lambda(\tau) \simeq_j \lambda(\tau j) \wedge \lambda(\tau j) \notin S_j)$

whenever $\tau j \in T$. One may show by induction on τ that $\lambda(\tau) \notin S_i$ for any $i \in \{1, \ldots, n\}$ occurring in τ. Indeed, if this property holds for τ, it must hold for τj because $\lambda(\tau) \simeq_j \lambda(\tau j)$ and $\simeq_j$ saturates S_i and $L \setminus S_i$ for all i occurring in τ. Therefore, $\mathcal{T}_3$ conforms to Def. 7.4.7, and the desired conclusion follows from Prop. 7.4.8. □

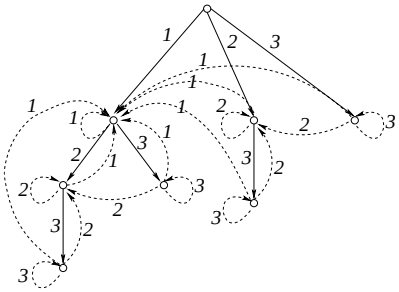


Figure 7.6: $\mathcal{T}_1$ for $n = 3$

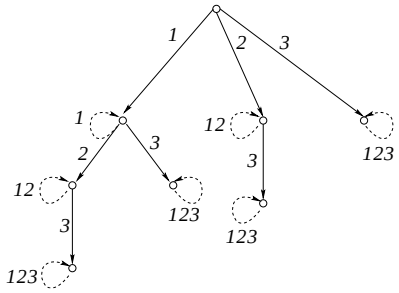


Figure 7.7: $\mathcal{T}_2$ for $n = 3$

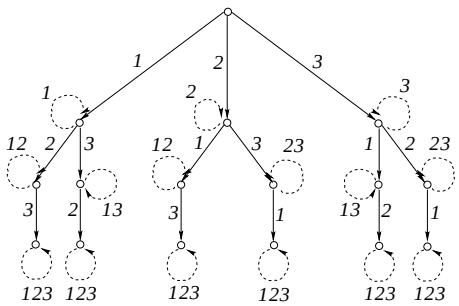


Figure 7.8: $\mathcal{T}_3$ for $n = 3$

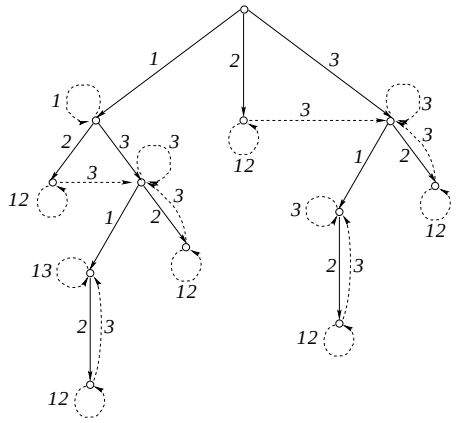


Figure 7.9: $\mathcal{T}_4$

One can deal similarly with many other situations where $\Sigma_i \subseteq \Sigma_j$ or $S_i \subseteq S_j$ or $\simeq_i$ saturates S_j, or conversely with i and j, for all distinct $i, j \in \{1, \ldots, n\}$. For instance, let $n = 3$, and suppose $S_1 \subseteq S_2$, $\Sigma_3 \subseteq \Sigma_2$, and $\simeq_1$ saturates S_3. Then the enabledness of all $w \in SupK(L, \mathcal{S})$ may be proved using the type $\mathcal{T}_4$ (see Figure 7.9).

Unfortunately, we cannot extend propositions 5,6, and 7 into a general proposition, for we do not know whether $SupK(L, \mathcal{S})$ is regular in three particular cases:
- $S_1 \subseteq S_2$, $\Sigma_2 \subseteq \Sigma_3$, and $\simeq_1$ saturates S_3,
- $S_1 \subseteq S_2$, $\simeq_2$ saturates S_3, and $\Sigma_3 \subseteq \Sigma_1$,
- $\simeq_1$ saturates S_2, $\simeq_2$ saturates S_3, and $\simeq_3$ saturates S_1.

The best we can do is therefore to propose an algorithm that constructs a unique type for all proofs of enabledness in all cases where this is possible. In this perspective, we introduce rewrite rules on labelled graphs. In each rule, one vertex of the left member is dropped and the edges that were incident to this vertex are redirected to other vertices. The vertices and edges present on both sides of a rule serve as an application context (indicated by the labels put on the concerned vertices). The rewrite rules are displayed in Figure 7.10 (where $i \neq j$ and sat is an abbreviation for "saturates").

Proposition 8 *Given $\mathcal{S} = \{(\Sigma_1, S_1), \ldots, (\Sigma_n, S_n)\}$, let $\mathcal{R}$ be the set of the rewrite rules that correspond to predicates true in $\mathcal{S}$. Whenever the complete n-ary tree rewrites to some finite graph, any such graph yields a uniform type $\mathcal{T}$ for proving the enabledness of all trajectories. The spanning tree of $\mathcal{T}$ is the subset of edges of the complete n-ary tree that have been preserved by the rewriting.*

Proof: In view of Def. 7.4.7 it is enough to show, for each graph G on the right hand side of a rewrite rule (see Figure 7.10), that any map $\lambda : \{x, y\} \to L$ or $\lambda : \{x, y, z\} \to L$ compatible with the rigid edges of G is compatible also with the dashed edge of G, where λ is compatible

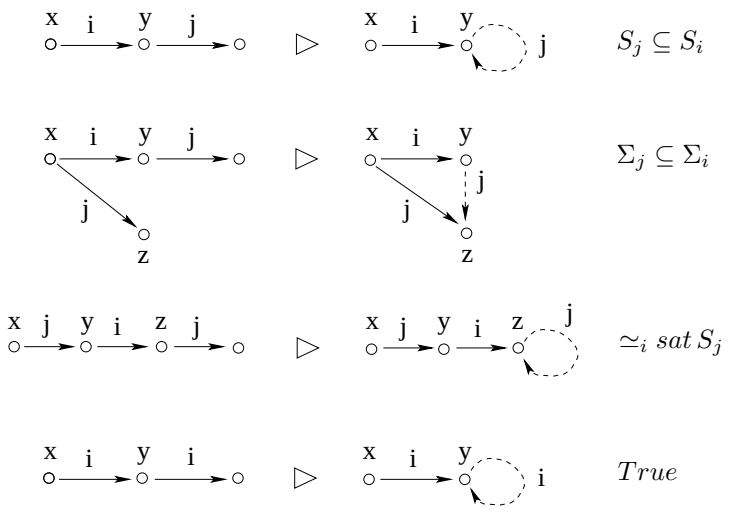


Figure 7.10: Four rules

with $x \xrightarrow{i} y$ if $\lambda(x) \simeq_i \lambda(y)$ and $\lambda(y) \notin S_i$. Considering the predicates defining the application conditions of the rewrite rules, this verification is immediate. □

When proposition 8 can be applied, the construction proposed in the proof of proposition 7.4.8 may be used to produce a finite automaton realizing the maximal permissive opacity control, but Prop. 8 is not immediately effective. We remedy now this deficiency.

Proposition 9 *It is decidable whether some finite graph may be derived from the complete n-ary tree using the rules in $\mathcal{R}$ and such graphs can be computed when they exist.*

Proof: As a preliminary remark, note that the rewrite rules in $\mathcal{R}$ are not necessarily confluent, hence the finite graph we compute is just one among a set of possible proof skeletons.

Let $I = \{1, \ldots, n\}$ and let $F \subseteq I^*$ be the set of all words ii, or ij, or iji such that $True$, or $(S_j \subseteq S_i \vee \Sigma_j \subseteq \Sigma_i)$, or $\simeq_i satS_j$, respectively. If the words in F are considered as forbidden factors for words in I^*, the remaining words form a regular language $T = I^* \setminus (I^*FI^*)$.

If T is infinite, the rewrite system $\mathcal{R}$ cannot terminate on the complete n-ary tree and it cannot produce any finite graph. If T is finite, let $(Q, \{1, \ldots, n\}, q_0)$ be the partial automaton defined with $Q = T$, $q_0 = \varepsilon$, and $\tau \cdot i = \tau i$ for τi in T.

To obtain a type $(Q, \{1, \ldots, n\}, q_0, T)$ conforming Def. 7.4.7, it suffices now to complete the partial automaton $(Q, \{1, \ldots, n\}, q_0)$ as follows: for all words τ in T, and by increasing lengths of words τ,
- set $\tau i \cdot j = \tau i$ if $\tau i \cdot j$ is undefined and $S_j \subseteq S_i$ or $\simeq_i satS_j$ and $\tau = \tau' \cdot j$,
- set $\tau i \cdot j = \tau \cdot j$ if $\tau i \cdot j$ is still undefined and $\Sigma_j \subseteq \Sigma_i$. □

Example 7.5.1 *Let $\Sigma = \{a, b, c\}$ and let L be a prefix-closed regular language over Σ. Define $\mathcal{S} = \{(\Sigma_1, S_1), \ldots, (\Sigma_3, S_3)\}$ such that $\Sigma_1 = \{a, c\}$, $\Sigma_2 = \{b, c\}$, $\Sigma_3 = \{b\}$, and $S_i = S_i\Sigma^*$ for all $i \in \{1, \ldots, 3\}$. The construction sketched in the proof of proposition 9 yields the type $\mathcal{T}_4$ and the spanning tree T displayed in Figure 7.9.*

$SupK(L, \mathcal{S})$ may be computed by stages following the structure of T. One computes first $SupK(L, \mathcal{S}) \setminus S_3$, using the type that appears in $\mathcal{T}_4$ at the end of both paths 13 and 3. Next, one computes $SupK(L, \mathcal{S}) \setminus S_1$ from $SupK(L, \mathcal{S}) \setminus S_3$, using the type at the end of the path 1 in $\mathcal{T}_4$. Finally, one computes $SupK(L, \mathcal{S})$ from $SupK(L, \mathcal{S}) \setminus S_1$ and $SupK(L, \mathcal{S}) \setminus S_3$.

7.6 conclusion

We shall try first in this section to illustrate the possible applications of the work we have presented. Consider a computer system that provides services to n users $U1, \ldots, Un$ with disjoint alphabets $\Sigma_1, \ldots, \Sigma_n$. Let $L \subseteq \Sigma^*$ be the language of the system, where $\Sigma \supseteq \Sigma_i$ for all i. One wants to give every user Ui the guarantee that no coalition of other users can ever be sure that he has started working. The problem is therefore to enforce the opacity of the concurrent secret $\mathcal{S} = \{(\Sigma'_1, S_1), \ldots, (\Sigma'_n, S_n)\}$ where for each i, $S_i = L \cap \Sigma^*\Sigma_i\Sigma^*$ and $\Sigma'_i = \cup_{j \neq i}\Sigma_j$. As $\simeq_j$ saturates S_i for $j \neq i$, one can construct a finite automaton accepting $SupK(L, \mathcal{S})$. We feel this example is typical of many practical security problems.

Some limitations of this work are voluntary, *e.g.* we restricted ourselves on purpose to regular languages and to regular control, but some other limitations could hopefully be lifted in continuations of this work. A list follows.

From the beginning of section 7.4, we worked with open secrets, *i.e.* secrets S_i such that $S_i\Sigma^* \subseteq S_i$. The goal was to make Def. 7.3.1 equivalent to the simpler definition Def. 7.4.1. Another way to obtain this equivalence is to impose on each secret S_i the following condition, where $\leq$ is the order prefix:
$(\forall w \in L \setminus S_i)\ \pi_i(w) = u\sigma \Rightarrow (\exists v \in L \setminus S_i)\ (v \leq w \wedge \pi_i(v) = u)$. Such secrets may *e.g.* carry the information that some system process *is* in a critical section.

As regards the control objective, we focussed our efforts on opacity, but we did not take the deadlock freeness or the liveness of the controlled system into consideration and this is a shortcoming. Another valuable extension would be to work with boolean combinations of opacity predicates, *e.g.* if S_1 is opaque w.r.t. Σ_1 then S_2 is not opaque w.r.t. Σ_2.

We end with a few words on observability and controlability. On the side of the observation functions, we have restricted our attention to projections on subalphabets, but it would be more adequate to accomodate also all alphabetic morphisms. As regards control, we dealt with all events as controlable events, but it would be more realistic to accomodate also uncontrolable events.

Bibliography

[1] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[2] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In *Proc. of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1998.

[3] S. Andova. Process algebra with probabilistic choice. In *ARTS*, volume 1601 of *LNCS*, pages 111–129. Springer, 1999.

[4] A. Arnold and M. Nivat. Comportements de processus. In *Actes du Colloque AFCET "Les mathématiques de l'informatique"*, pages 35–68, 1982.

[5] André Arnold. *Finite transition systems*. Prentice Hall, 1994.

[6] André Arnold and Maurice Nivat. Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Theoretical Computer Science*, 11, 1980.

[7] E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau. Concurrent secrets. Rapport de recherche 5771, INRIA, nov 2005.

[8] E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau. Concurrent secrets. In S. Lafortune, F. Lin, and D. Tilbury, editors, *8th Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, Michigan, USA, jul 2006.

[9] E. Badouel, B. Caillaud, and P. Darondeau. Distributing finite automata through petri net synthesis. *Journal on Formal Aspects of Computing*, 13:447–470, 2002.

[10] E. Badouel and Ph. Darondeau. On the synthesis of general petri nets. Research Report 3025, Inria, 1996.

[11] Eric Badouel, Marek Bednarczyk, Andrje Borzyszkowski, Benoît Caillaud, and Philippe Darondeau. Concurrent secrets. *Discrete Event Dynamic Systems*, 17(4):425–446, dec 2007.

[12] Eric Badouel, Luca Bernardinello, and Philippe Darondeau. Polynomial algorithms for the synthesis of bounded nets. In *Proceedings Caap 95*, volume 915 of *Lecture Notes in Computer Science*, 1995.

[13] Eric Badouel, Luca Bernardinello, and Philippe Darondeau. The synthesis problem for elementary net systems is np-complete. *Theor. Comput. Sci.*, 186(1–2):107–134, 1997.

[14] S. Basu. New results on quantifier elimination over real closed fields and applications to constraint databases. *Journal of the ACM*, 46(4):537–555, July 1999.

[15] M. A. Bednarczyk, L. Bernardinello, B. Caillaud, W. Pawlowski, and L. Pomello. Modular system development with pullbacks. In *Applications and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 140–160. Springer, jun 2003.

[16] Nikola Benes, Jan Kretinsky, Kim G. Larsen, and Jiri Srba. On determinism in modal transition systems. *Theoretical Computer Science*, 410(41):4026–2043, 2009.

[17] A. Benveniste, B. Caillaud, L. Carloni, P. Caspi, and A. Sangiovanni-Vincentelli. Causality and scheduling constraints in heterogeneous reactive systems modeling. In *FMCO 2003, Proceedings of the Second International Symposium on Formal Methods for Components and Objects*, volume 3188 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2004.

[18] A. Benveniste, B. Caillaud, L. Carloni, P. Caspi, and A. Sangiovanni-Vincentelli. Heterogeneous reactive systems modeling: Capturing causality and the correctness of loosely time-triggered architectures (ltta). In *Proceedings of the Fourth ACM International Conference on Embedded Software, EMSOFT'04*. ACM Press, sep 2004.

[19] A. Benveniste, B. Caillaud, L. Carloni, P. Caspi, and A. Sangiovanni-Vincentelli. Communication by sampling in time-sensitive distributed systems. In *Proceedings of the Sixth Annual ACM Conference on Embedded Software, EMSOFT'06*, pages 152–160. ACM Press, 2006.

[20] A. Benveniste, B. Caillaud, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. Tag machines. In *Proceedings of the fifth ACM International Conference on Embedded Software (Emsoft)*, pages 255–263, Jersey City, NJ, USA, sep 2005. ACM Press.

[21] A. Benveniste, B. Caillaud, and P. Le Guernic. From synchrony to asynchrony. In J.C.M. Baeten and S. Mauw, editors, *CONCUR'99, Concurrency Theory, 10th International Conference*, volume 1664 of *Lecture Notes in Computer Science*, pages 162–177. Springer, aug 1999.

[22] A. Benveniste, B. Caillaud, and P. Le Guernic. From synchrony to asynchrony. Research report 3641, INRIA Rennes, mar 1999. Also published as IRISA Research Report PI-1233.

[23] A. Benveniste, B. Caillaud, and P. Le Guernic. Compositionality in dataflow synchronous languages: specification and distributed code generation. *Information and Computation*, 163:125–171, 2000.

[24] A. Benveniste, B. Caillaud, and R. Passerone. A generic model of contracts for embedded systems. Research report 6214, INRIA Rennes, jun 2007.

[25] A. Benveniste, B. Caillaud, and M. Pouzet. The fundamentals of hybrid systems modelers. In *IEEE Conf. on Decision and Control, CDC*, 2010.

[26] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The Synchronous Languages Twelve Years Later. *Proceedings of the IEEE*, 91(1):64–83, 2003.

[27] Albert Benveniste. Compositional and uniform modelling of hybrid systems. *IEEE Trans. on Automatic Control*, 43(4):579–584, April 1998.

[28] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple viewpoint contract-based specification and design. In *Proceedings of the Software Technology Concertation on Formal Methods for Components and Objects (FMCO'07)*, volume 5382 of *Revised Lectures, Lecture Notes in Computer Science*, Amsterdam, The Netherlands, oct 2008. Springer.

[29] Albert Benveniste, Benoît Caillaud, and Paul Le Guernic. Compositionality in dataflow synchronous languages: Specification and distributed code generation. *Inf. Comput.*, 163(1):125–171, 2000.

[30] Albert Benveniste, Benoît Caillaud, Luca P. Carloni, Paul Caspi, and Alberto L. Sangiovanni-Vincentelli. Composing heterogeneous reactive systems. *ACM Trans. Embedded Comput. Syst.*, 7(4), 2008.

[31] Albert Benveniste, Benoît Caillaud, and Roberto Passerone. Multi-viewpoint state machines for rich component models. In Pieter Mosterman and Gabriela Nicolescu, editors, *Model-Based Design of Heterogeneous Embedded Systems*. CRC Press, 2009.

[32] Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.

[33] Albert Benveniste, Paul Le Guernic, Yves Sorel, and Michel Sorine. A denotational theory of synchronous reactive systems. *Inf. Comput.*, 99(2):192–230, 1992.

[34] L. Bernardinello, G. De Michelis, K. Petruni, and S. Vigna. On the synchronic structure of transition systems. In J. Desel, editor, *Structures in Concurrency Theory*, pages 11–31. Springer-Verlag, 1996.

[35] G. Berry. The constructive semantics of pure Esterel. Draft book available at http://www.esterel-technologies.com/, July 1999.

[36] Gérard Berry. Constructive Semantics of Esterel: From Theory to Practice (Abstract). In *AMAST '96: Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology*, page 225, London, UK, 1996. Springer-Verlag.

[37] J. Berstel. *Transductions and Context-Free Languages*. Teubner Verlag, 1978.

[38] Nathalie Bertrand, Sophie Pinchinat, and Jean-Baptiste Raclet. Refinement and consistency of timed modal specifications. In *Proc. of the 3rd International Conference on Language and Automata Theory and Applications (LATA'09)*, volume 5457 of *Lecture Notes in Computer Science*, pages 152–163, Tarragona, Spain, 2009. Springer.

[39] Simon Bliudze. *Un cadre formel pour l'étude des systèmes industriels complexes: un exemple basé sur l'infrastructure de l'UMTS*. PhD thesis, Ecole Polytechnique, 2006.

[40] Simon Bliudze and Daniel Krob. Modelling of complex systems: Systems as dataflow machines. *Fundam. Inform.*, 91(2):251–274, 2009.

[41] Simon Bliudze and Joseph Sifakis. A notion of glue expressiveness for component-based systems. In *Proc. of the 19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2008.

[42] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou. Handshake protocols for de-synchronization. In *Proceedings Async04*, pages 149–158, Crete, Greece, 2004.

[43] F. Boussinot. Une sémantique du langage Esterel . Technical Report 577, INRIA, 1986.

[44] Thomas Brihaye, François Laroussinie, Nicolas Markey, and Ghassan Oreiby. Timed concurrent game structures. In *Proc. of the 18th International Conference on Concurrency Theory (CONCUR'07)*, volume 4703 of *Lecture Notes in Computer Science*, pages 445–459. Springer, 2007.

[45] C. W. Brown. Simple cad construction and its applications. *Journal of Symbolic Computation*, 31(5):521–547, May 2001.

[46] C. W. Brown and J. H. Davenport. The complexity of quantifier elimination and cylindrical algeraic decomposition. In *Proceedings of the 2007 international symposium on Symbolic and algebraic computation (ISSAC'07)*, pages 54–60, Waterloo, ON, Canada, 2007.

[47] J.W. Bryans, M. Koutny, L. Mazaré, and P.Y.A. Ryan. Opacity generalised to transition systems. In *Proc. of the Workshop on Formal Aspects in Security and Trust (FAST 2005)*, 2005.

[48] J. F. M. Burg. *Linguistic instruments in requirements engineering*. IOS Press, 1997.

[49] B. Caillaud. SYNET : un outil de synthèse de réseaux de petri bornés, applications. Rapport de recherche 3155, INRIA, avril 1997.

[50] B. Caillaud. Bounded petri-net synthesis techniques and their applications to the distribution of reactive automata. *JESA, European Journal on Automated Systems*, 33(8–9):925–942, 1999.

[51] B. Caillaud, P. Caspi, A. Girault, and C. Jard. Distributing automata for asynchronous networks of processors. *European Journal on Automated Systems (JESA)*, 31(3):503–524, 1997.

[52] B. Caillaud, P. Darondeau, L. Hélouët, and G. Lesventes. Hmscs as specifications... with pn as completions. In F. Cassez, C. Jard, B. Rozoy, and M. Ryan, editors, *Proceedings of the summer school MOVEP'2k: Modelling and verification of parallel processes*, pages 87–103, Nantes, jun 2000.

[53] B. Caillaud, P. Darondeau, L. Hélouët, and G. Lesventes. Hmscs as specifications... with pn as completions. Research report 3970, INRIA Rennes, jul 2000.

[54] B. Caillaud, P. Darondeau, L. Hélouët, and G. Lesventes. *HMSCs as specifications... with PN as completions*, volume 2067 of *Lecture Notes in Computer Science*, pages 125–152. Springer, 2001.

[55] B. Caillaud, P. Darondeau, L. Lavagno, and X. Xie (eds.). *Synthesis and Control of Discrete Event Systems*. Kluwer Academic Press, 2002.

[56] Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Compositional Design Methodology with Constraint Markov Chains. Research Report RR-6993, INRIA, 2009.

[57] Benoît Caillaud, Benoit Delahaye, Kim G. Larsen, Axel Legay, Mikkel Larsen Pedersen, and Andrzej Wasowski. Compositional design methodology with constraint markov chains. In *Proceedings of the 7th International Conference on Quantitative Evaluation of SysTems (QEST) 2010*. IEEE Computer Society, 2010.

[58] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos*. Springer, 2006. ISBN 0-387-27802-8.

[59] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli. The theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):1059–1076, 2001.

[60] Luca P. Carloni, Roberto Passerone, Alessandro Pinto, and Alberto L. Sangiovanni-Vincentelli. Languages and tools for hybrid systems design. *Foundations and Trends in Electronic Design Automation*, 1(1/2), 2006.

[61] P. Caspi, A. Girault, and D. Pilaud. Automatic distribution of reactive systems for asynchronous networks of processors. *IEEE Trans. on Software Engineering*, 25(3):416–427, 1999.

[62] Paul Caspi and Marc Pouzet. A co-iterative characterization of synchronous stream functions. *Electr. Notes Theor. Comput. Sci.*, 11, 1998.

[63] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Freddy Y. C. Mang. Synchronous and bidirectional component interfaces. In *Proc. of the 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 414–427, 2002.

[64] Thomas Chatain, Alexandre David, and Kim G. Larsen. Playing games with timed games. Research Report LSV-08-34, Laboratoire Spécification et Vérification, ENS Cachan, France, December 2008. 15 pages.

[65] K. Chatterjee, K. Sen, and T. A. Henzinger. Model-checking omega-regular properties of interval Markov chains. In *FoSSaCS*, volume 4962 of *LNCS*, pages 302–317. Springer, 2008.

[66] W.K. Chen. *Applied Graph Theory*. North Holland, 1971.

[67] F. Ciesinski and M. Größer. On probabilistic computation tree logic. In *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 147–188. Springer, 2004.

[68] The SPEEDS Consortium. Speeds methodology - a white paper, 2008.

[69] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Complete state encoding based on the theory of regions. In *Proc. 2nd Int. Symposium on Advanced Research on Asynchronous Circuits and Systems*, pages 36–47. IEEE Computer Society Press, 1996.

[70] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Deriving petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882, 1998.

[71] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer, 2002.

[72] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing petri nets from state-based models. In *Proceedings of ICCAD'95*, pages 164–173. IEEE Computer Society Press, 1995.

[73] Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[74] S. Dasgupta, D. Potop-Butucaru, B. Caillaud, and A. Yakovlev. From weakly endochronous systems to delay-insensitive circuits. In *Proceedings of the second international workshop on formal methods for globally asynchronous locally synchronous design (FMGALS 2005)*, 2005.

[75] Luca de Alfaro. Game models for open systems. In *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 269–289. Springer, 2003.

[76] Luca de Alfaro, Leandro Dias da Silva, Marco Faella, Axel Legay, Pritam Roy, and Maria Sorea. Sociable interfaces. In *5th International Workshop on Frontiers of Combining Systems (FroCos'05)*, volume 3717 of *Lecture Notes in Computer Science*, pages 81–105. Springer, 2005.

[77] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In *Proc. of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003.

[78] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*, pages 109–120. ACM Press, 2001.

[79] Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Timed interfaces. In *Proc. of the 2nd Workshop on Embedded Software (EMSOFT'02)*, volume 2491 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2002.

[80] Benoît Delahaye, Benoît Caillaud, and Axel Legay. Compositional Reasoning on (Probabilistic) Contracts. Research Report RR-6970, INRIA, 2009.

[81] Benoît Delahaye, Benoît Caillaud, and Axel Legay. Probabilistic contracts : A compositional reasoning methodology for the design of systems with stochastic and/or nondeterministic aspects. *Formal Methods in System Design*, 2011. to appear.

[82] Benoît Delahaye and Benoît Caillaud. A model for probabilistic reasoning on assume/guarantee contracts. Research Report 6719, INRIA, 2008.

[83] Benoît Delahaye, Benoît Caillaud, and Axel Legay. Probabilistic contracts: A compositional reasoning methodology for the design of stochastic systems. In *Proc. of the 10th International Conference on Application of Concurrency to System Design (ACSD'10)*, Braga, Portugal, June 2010.

[84] Jörg Desel and Wolfgang Reisig. The synthesis problem of petri nets. *Acta Inf.*, 33(4):297–315, 1996.

[85] Laurent Doyen, Thomas A. Henzinger, Barbara Jobstmann, and Tatjana Petrov. Interface theories with component reuse. In L. de Alfaro and J. Palsberg, editors, *Proc. of the 8th International Conference on Embedded Software (EMSOFT'08)*, pages 79–88. ACM Press, 2008.

[86] M. Droste and R.M. Shortt. Petri nets and automata with concurrency relations - an adjunction. In M. Droste and Y. Gurevich, editors, *Semantics of Programming Languages and Model Theory*, pages 69–87, 1993.

[87] N. Cutland (ed.). *Nonstandard analysis and its applications*. Cambridge Univ. Press, 1988.

[88] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Theory of 2-structures, part i: Clans, basic subclasses, and morphisms. *Theor. Comput. Sci.*, 70(3):277–303, 1990.

[89] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Theory of 2-structures, part ii: Representation through labeled tree families. *Theor. Comput. Sci.*, 70(3):305–342, 1990.

[90] Johan Eker, Jörn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, Stephen Neuendorffer, S. Sachs, and Yuhong Xiong. Taming heterogeneity - the ptolemy approach. *Proc. of the IEEE*, 91(1):127–144, 2003.

[91] H. Fecher, M. Leucker, and V. Wolf. Don't Know in probabilistic systems. In *SPIN*, volume 3925 of *LNCS*, pages 71–88. Springer, 2006.

[92] G. Feuillade. Modal specifications are a syntactic fragment of the mu-calculus. Research Report RR-5612, INRIA, June 2005.

[93] A.F. Filippov. *Differential Equations with Discontinuous Right-hand Sides*. Wiley, 1988. ISBN 978-9027726995.

[94] Cédric Fournet, C. A. R. Hoare, Sriram K. Rajamani, and Jakob Rehof. Stuck-free conformance. In *Proc. of the 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2004.

[95] Angelo Gargantini, Dino Mandrioli, and Angelo Morzenti. Dealing with zero-time transitions in axiom systems. *Information and Computation*, 150(2):119–131, 1999.

[96] M. Gondran and M. Minoux. *Graphs and Algorithms*. John Wiley, 1984.

[97] T. Grandpierre and Y. Sorel. From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations. In *Proceedings MEMOCODE'03*, Mont Saint-Michel, France, 2003.

[98] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer Academic Publishers, 1993.

[99] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994.

[100] Thomas A. Henzinger and Joseph Sifakis. The embedded systems design challenge. In *Proc. of the 14th International Symposium on Formal Methods (FM'06)*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

[101] H. Hermanns, U. Herzog, and J. Katoen. Process algebra for performance evaluation. *Theor. Comput. Sci.*, 274(1-2):43–87, 2002.

[102] H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In *CAV*, volume 5123 of *LNCS*, pages 162–175. Springer, 2008.

[103] H. Hermans. *Interactive Markov Chains*. springer, 2002.

[104] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[105] Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, September 2005.

[106] D. Hogrefe. OSI formal specification case study: the INRES protocol and service. Technical Report 91-012, University of Bern, 1991.

[107] Richard P. Hopkins. Distributable nets. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1991, Papers from the 11th International Conference on Applications and Theory of Petri Nets*, volume 524 of *Lecture Notes in Computer Science*, pages 161–187. Springer, 1991.

[108] F. Hoppensteadt. Properties of solutions of ordinary differential equations with small parameters. *Comm. on Pure and Applied Math.*, 24:807–840, 1971.

[109] INRIA Rennes. *Proceedings of the Symposium on the Supervisory Control of Discrete Event Systems, SCODES'2001*, Paris, France, jul 2001.

[110] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, September 1999.

[111] Yumi Iwasaki, Adam Farquhar, Vijay A. Saraswat, Daniel G. Bobrow, and Vineet Gupta. Modeling time in hybrid systems: How fast is "instantaneous"? In *IJCAI*, pages 1773–1781, 1995.

[112] D. Klink J. Katoen and M. R. Neuhauser. Compositional abstraction for stochastic systems. In *Proceedings of the 7th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'09)*, LNCS, pages 195–211. Springer, 2009.

[113] B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In *LICS*, pages 266–277. IEEE Computer Society, 1991.

[114] J. Katoen, D. Klink, M. Leucker, and V. Wolf. Three-valued abstraction for continuous-time Markov chains. In *CAV*, volume 4590 of *LNCS*, pages 311–324. Springer, 2007.

[115] Robert Keller. Towards a theory of speed-independent modules. *IEEE Transactions on Computers*, C-23(1):21–33, January 1974.

[116] Robert Keller. A fundamental theorem of asynchronous parallel computation. *Lecture Notes in Computer Science*, 24:103–112, 1975.

[117] J. Klein, B. Caillaud, and L. Hélouët. Merging scenarios. In *Proceedings of the Ninth International Workshop on Formal Methods for Industrial Critical Systems, FMICS'04*, volume 133 of *Electronic Notes in Theoretical Computer Science*, pages 193–215, Linz, Austria, 2005.

[118] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[119] K. G. Larsen. Modal specifications. In *AVMS*, volume 407 of *LNCS*, pages 232–246. Springer, 1989.

[120] K. G. Larsen and A. Skou. Compositional verification of probabilistic processes. In *CONCUR*, volume 630 of *LNCS*, pages 456–471. Springer, 1992.

[121] K. Guldstrand Larsen, U. Nyman, and A. Wasowski. On modal refinement and consistency. In *CONCUR*, volume 4703 of *LNCS*, pages 105–119. Springer, 2007.

[122] Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. Modal I/O automata for interface and product line theories. In *Programming languages and systems, 16th European Symposium on Programming, ESOP 2007*, pages 64–79, 2007.

[123] Edward A. Lee and Alberto L. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 17(12):1217–1229, 1998.

[124] Edward A. Lee and Haiyang Zheng. Operational semantics of hybrid systems. In *HSCC*, pages 25–53, 2005.

[125] Edward A. Lee and Haiyang Zheng. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In *EMSOFT*, pages 114–123, 2007.

[126] T. Lindstrøm. An invitation to nonstandard analysis. In N.J. Cutland, editor, *Nonstandard Analysis and its Applications*, pages 1–105. Cambridge Univ. Press, 1988.

[127] N. López and M. Núñez. An overview of probabilistic process algebras and their equivalences. In *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 89–123. Springer, 2004.

[128] G. Lüttgen and W. Vogler. Conjunction on processes: Full abstraction via ready-tree semantics. *Theoretical Computer Science*, 373:19–40, 2007.

[129] N. Lynch and E. Stark. A proof of the Kahn principle for input/output automata. *Information and Computation*, 82(1):81–92, 7 1989.

[130] Nancy Lynch and Mark R. Tuttle. An introduction to Input/Output automata. *CWI-quarterly*, 2(3), 1989.

[131] S. Mac Lane and G. Birkhoff. *Algebra*. Chelsea Publishing Company, 1967.

[132] Alain Martin. The limitations of delay-insensitivity in asynchronous circuits. technical report CS-TR-90-02, Caltech, 1990.

[133] L. Mazaré. Using unification for opacity properties. In *Proc. of the Workshop on Issues in the Theory of Security (WITS'04)*, 2004.

[134] Pieter J. Mosterman, Justyna Zander, Gregoire Hamon, and Ben Denckla. Towards Computational Hybrid System Semantics for Time-Based Block Diagrams. In *3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09)*, pages 376–385, Zaragoza, Spain, September 2009. keynote paper.

[135] Madhavan Mukund. Petri nets and step transition systems. *Int. J. Found. Comput. Sci.*, 3(4):443–478, 1992.

[136] David E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243. Harvard University Press, 1959.

[137] M. Najafi and R. Nikoukhah. Implementation of Hybrid Automata in Scicos. In *IEEE Multi-conference on Systems and Control*, 2007.

[138] Ulrik Nyman. *Modal Transition Systems as the Basis for Interface Theories and Product Lines*. PhD thesis, Aalborg University, Department of Computer Science, September 2008.

[139] Julien Ouy, Jean-Pierre Talpin, Loïc Besnard, and Paul Le Guernic. Separate compilation of polychronous specifications. *Electr. Notes Theor. Comput. Sci.*, 200(1):51–70, 2008.

[140] D. Potop-Butucaru and B. Caillaud. Correct-by-construction asynchronous implementation of modular synchronous specifications. In *Proceedings of the Fifth International Conference on Application of Concurrency to System Design, ACSD 2005*, 2005.

[141] D. Potop-Butucaru, B. Caillaud, and A. Benveniste. Concurrency in synchronous systems. In *Proceedings of the International Conference on Application of Concurrency to System Design, ACSD 2004*, 2004.

[142] D. Potop-Butucaru, B. Caillaud, and A. Benveniste. Concurrency in synchronous systems. Research Report 5110, INRIA, feb 2004. Also published as IRISA Internal Publication PI-1605.

[143] D. Potop-Butucaru, B. Caillaud, and A. Benveniste. Concurrency in synchronous systems. *Formal Methods in System Design*, 28(2), mar 2006.

[144] D. Potop-Butucaru, R. de Simone, and J.-P. Talpin. The synchronous hypothesis and synchronous languages. In R. Zurawski, editor, *The Embedded Systems Handbook*, 2005. CRC Press.

[145] Dumitru Potop-Butucaru, Robert de Simone, and Yves Sorel. Necessary and sufficient conditions for deterministic desynchronization. In *Proceedings of the 7th ACM & IEEE international conference on embedded software, EMSOFT 2007*, pages 124–133, 2007.

[146] Dumitru Potop-Butucaru, Robert de Simone, Yves Sorel, and Jean-Pierre Talpin. From concurrent multi-clock programs to deterministic asynchronous implementations. In *Proc. of the ninth international conference on application of concurrency to system design, ACSD 2009*, pages 42–51, 2009.

[147] P. Potop-Butucaru and B. Caillaud. Correct-by-construction asynchronous implementation of modular synchronous specifications. *Fundamenta Informaticae*, 78(1):131–159, 2007.

[148] J-B. Raclet. Residual for component specifications. In *FACS*, 2007.

[149] Jean-Baptiste Raclet. *Quotient de spécifications pour la réutilisation de composants*. PhD thesis, Université de Rennes I, december 2007. (In French).

[150] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoit Caillaud, and Roberto Passerone. Why are modalities good for interface theories? In *Proc. of the 9th International Conference on Application of Concurrency to System Design (ACSD'09)*, pages 119–127. IEEE Computer Society Press, 2009.

[151] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoit Caillaud, and Roberto Passerone. Why are modalities good for interface theories? Research Report 6899, INRIA, 2009.

[152] Jean-Baptiste Raclet, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. Modal interfaces: Unifying interface automata and modal specifications. In *Proc. 9th International Conference on Embedded Software (EMSOFT 2009)*, pages 87–96. IEEE Computer Society, oct 2009.

[153] Jean-Baptiste Raclet, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. A modal interface theory for component-based design. *Fundamenta Informaticae*, 107:1–32, 2011.

[154] P.J. Ramadge and W.M. Wonham. On the supremal controllable language of a given language. *SIAM Journal of Control and Optimization*, 25:637–659, 1987.

[155] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25:206–230, 1987.

[156] Ramine Nikoukhah. Hybrid dynamics in Modelica: Should all events be considered synchronous? In *First Internation Workshop on Equation-Based Object Oriented Languages and Tools (EOOLT 2007)*, pages 37–48, Berlin, Germany, 2007.

[157] W. Reisig. Petri nets. In *EATCS Monographs on Theoretical Computer Science*, volume 4. Springer, 1985.

[158] N. Rezg, X. Xie, and A. Ghaffari. Supervisory control in discrete event systems using the theory of regions. In R. Boel and G. Stremersch, editors, *Discrete Event Systems: Analysis and Control*, pages 391–398. Kluwer Academic Publishers, 2000.

[159] Laurie Ricker and Benoiît Caillaud. Revisiting state-baed models for synthesizing optimal communicating decentralized discrete-event controllers. In *European Control Conference 2009 (ECC'09)*, Budapest, Hungary, aug 2009.

[160] Laurie Ricker and Benoît Caillaud. Mind the gap: Expanding communication options in decentralized discrete-event control. In *46th IEEE Conference on Decision and Control*, New Orleans, LA, USA, 2007.

[161] A. Robinson. *Non-Standard Analysis*. Princeton Landmarks in Mathematics, 1996. ISBN 0-691-04490-2.

[162] Heinrich Rust. *Operational semantics for timed systems: a non-standard approach to uniform modeling of timed and hybrid systems*, volume 3456 of *Lecture notes in computer science*. Springer, 2005.

[163] A. Schrijver. *Theory of linear and integer programming*. Wiley, April 1998.

[164] K. Sen, M. Viswanathan, and G. Agha. Model-checking Markov chains in the presence of uncertainties. In *TACAS*, volume 3920 of *LNCS*, pages 394–410. Springer, 2006.

[165] M. Singh and M. Theobald. Generalized latency insensitive systems for gals architectures. In *Proceedings FMGALS2003*, Pisa, Italy, 2003.

[166] E. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64(3):221–269, 1989.

[167] J.-P. Talpin, P. Le Guernic, S. K. Shukla, R. Gupta, and F. Doucet. Formal refinement checking in a system-level design methodology. *Fundamenta Informaticae*, 62(2):243–273, 2004.

[168] J.-P. Talpin, D. Potop-Butucaru, J. Ouy, and B. Caillaud. Compositional synthesis of latency-insensitive systems from multi-clocked synchronous specifications. Research report 1730, IRISA, jun 2005.

[169] J.-P. Talpin, D. Potop-Butucaru, J. Ouy, and B. Caillaud. From multi-clocked synchronous processes to latency-insensitive modules (short paper). In *Proceedings of the fifth ACM*

International Conference on Embedded Software (Emsoft), pages 282–285, Jersey City, NJ, USA, sep 2005. ACM Press.

[170] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. RAND Corp., 1948.

[171] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[172] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Journal on Software — Concepts and Tools*, 17(3):103–120, 1996.

[173] Jan Tijmen Udding. A formal model for defining and classifying delay-insensitive circuits and systems. *Distributed Computing*, 1(4):197–204, 1986.

[174] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. In *Proc. IFIP Congress*, pages 613–618. North Holland / IFIP, 1989.

[175] W. Vogler. Concurrent implementation of asynchronous transition systems. In *Proceedings of ICATPN'99*, volume 1639 of *LNCS*, pages 284–303. Springer-Verlag, 1999.

[176] Michael Winokur, Susanne Graf, and Bernhard Josko. Contract-based system design - the speeds approach. In *Proceedings of the 2008 INCOSE International Symposium*, 2008.

[177] A. Yakovlev. Designing control logic for counterflow pipeline processor using petri nets. *Formal Methods in System Design*, 12:39–71, 1998.

[178] Hitoshi Yanami and Hirokazu Anai. Synrac: a maple toolbox for solving real algebraic constraints. *ACM Communications in Computer Algebra*, 41(3):112–113, September 2007.

[179] K. Yun and D. Dill. Automatic synthesis of extended burst-mode circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(2):101–132, feb. 1999.