

FACULTÉ DES SCIENCES
Département d'Informatique
Service des méthodes formelles
et vérification

UNIVERSITÉ LIBRE DE BRUXELLES
UNIVERSITÉ D'EUROPE

The logo of the University of Brussels (ULB) is a blue square with the white letters "ULB" inside.

Supervisory Control Of Infinite State Systems Under Partial Observation

Gabriel KALYON

Dissertation présentée en vue de
l'obtention du grade de **Docteur en
Sciences**

Année académique 2010-2011

À mon père

Acknowledgments

First of all, I would like to thank my thesis advisor Thierry Massart who has guided me throughout this thesis. I am very grateful to him for his continuous support and his steady confidence in my work. It was a great pleasure to work with him.

I would also like to thank my main co-authors (and friends): Hervé, Tristan and Cédric without which this thesis would not have seen the light of day. Working with all of them was a great pleasure.

I would not forget the whole Computer Science Department and the Verification Group at Université Libre de Bruxelles. I would like to mention a few of them in particular: Mahsa, Vincent, Antoine, Nico, Jean-François, Souhaib, Frédéric, Fréd, Eythan, Gilles, Manu, Aldric, Joël, Jean, Maryka, Pascaline and Véronique. My gratitude also goes to the members of the Vertecs team (INRIA Rennes): Thierry, Nathalie, Camille, Jérémy, Vlad, Flo and Christophe.

I wish to thank the members of the Jury: Prof. Bernard Fortz, Dr. Jan Komenda, Prof. Stéphane Lafortune, Dr. Hervé Marchand, Prof. Jean-François Raskin, Prof. Thierry Massart.

I am grateful to the FNRS (the belgian National Fund for Scientific Research) for the support I received in the form of a research fellow position.

On a personal note, I would also like to thank John, Hakim, Ahmet, David, Yvan, Hadrien, Abdu, Abdeslam, Ouael, Come.

Finally, I would like to thank my family for their endless love and support. Thank you for everything.

Contents

Acknowledgments	i
Introduction	1
Validation Methods	2
Objectives and Structure of this Thesis	7
I Fundamentals and Definitions	13
1 Preliminaries	15
1.1 Basic Notions	15
1.2 Languages and Automata	17
1.3 Lattice Theory	22
1.3.1 Partially Ordered Set and Lattice	22
1.3.2 Fixpoints	23
1.4 Abstract Interpretation	25
1.4.1 The Galois Connection Framework	25
1.4.2 The Representation Framework	28
2 Supervisory Control of Discrete Event Systems	31
2.1 Supervisory Control Theory of Finite Discrete Event Systems	32
2.1.1 Modeling of Discrete Event Systems	32
2.1.2 Event-based Approach	33
2.1.3 State-based Approach	44
2.2 Control of Infinite State Systems	51
2.2.1 Controller Synthesis with Petri Nets	52
2.2.2 Control of Timed Automata	54
2.2.3 Control of Systems using Variables	56

2.3 Conclusion	57
II Control of Infinite State Systems	59
3 Synthesis of Centralized Memoryless Controllers with Partial Observation for Infinite State Systems	61
3.1 Symbolic Transition Systems	62
3.1.1 Model of STS	63
3.1.2 Properties and Operations on STS	66
3.2 State Invariance Properties	69
3.3 Framework and State Avoidance Control Problem	72
3.3.1 Means of Observation	72
3.3.2 Means of Control	73
3.3.3 Controller and Controlled System	73
3.3.4 Definition of the Control Problems	75
3.4 Computation by Means of Abstract Interpretation of a Memoryless Controller for the Basic Problem	82
3.4.1 Semi-algorithm for the Basic Problem	82
3.4.2 Effective Algorithm for the Basic Problem	88
3.4.3 Evaluation and Improvement of the Permissiveness of the Controller	97
3.5 Computation by Means of Abstract Interpretation of a Memoryless Controller for the Deadlock Free Problem	103
3.5.1 Semi-algorithm for the Deadlock Free Problem	103
3.5.2 Effective Algorithm for the Deadlock Free Problem	110
3.6 The Non-blocking Problem	111
3.7 Experimental Results	111
3.7.1 Description of SMACS	112
3.7.2 Empirical Evaluation	113
4 Synthesis of Centralized k-memory and Online Controllers with Partial Observation for Infinite State Systems	127
4.1 Synthesis of k -Memory Controllers	128
4.1.1 Framework and State Avoidance Control Problem	128
4.1.2 Semi-algorithm for the Memory Basic Problem	131

4.1.3	Effective Algorithm for the Memory Basic Problem	137
4.2	Synthesis of Online Controllers	137
4.2.1	Framework and State Avoidance Control Problem	137
4.2.2	Semi-algorithm for the Online Basic Problem	139
4.2.3	Effective Algorithm for the Online Basic Problem	142
4.3	Comparisons between the Memoryless, k -Memory and On-line Controllers	142
5	Synthesis of Decentralized and Modular Controllers for Infinite State Systems	149
5.1	Synthesis of Decentralized Controllers	150
5.1.1	Framework and State Avoidance Control Problem	151
5.1.2	Computation by Means of Abstract Interpretation of a Decentralized Controller for the Basic Decentralized Problem	155
5.1.2.1	Semi-Algorithm for the Basic Decentralized Problem	155
5.1.2.2	Effective Algorithm for the Basic Decentralized Problem	160
5.1.3	Computation by Means of Abstract Interpretation of a Decentralized Controller for the Deadlock Free Decentralized Problem	160
5.1.3.1	Semi-Algorithm for the Deadlock Free Decentralized Problem	160
5.1.3.2	Effective Algorithm for the Deadlock Free Decentralized Problem	166
5.1.4	Comparison Between the Centralized and Decentralized Controllers	167
5.2	Synthesis of Modular Controllers	170
5.2.1	Framework and State Avoidance Control Problem	171
5.2.2	Semi-Algorithm for the Basic Modular Problem	174
5.2.3	Effective Algorithm for the Basic Modular Problem	182
5.3	Experimental Evaluation	182
6	Distributed Supervisory Control of FIFO Systems	189
6.1	Overview of the Method	191
6.2	Model of the System	196
6.3	Framework and State Avoidance Control Problem	200
6.3.1	Means of Control	200

6.3.2	Distributed Controller and Controlled Execution	202
6.3.3	Definition of the Control Problem	203
6.4	State Estimates of Distributed Systems	204
6.4.1	Instrumentation	204
6.4.2	Computation of State Estimates	209
6.4.2.1	State Estimate Algorithm	209
6.4.2.2	Properties	214
6.5	Computation by Means of Abstract Interpretation of Dis- tributed Controllers for the Distributed Problem	217
6.5.1	Semi-algorithm for the Distributed Problem	217
6.5.2	Effective Algorithm for the Distributed Problem	221
6.6	Proofs of Propositions 6.4 and 6.5	225
6.6.1	Lemmas	225
6.6.2	Proof of Proposition 6.4	227
6.6.3	Proof of Proposition 6.5	234
III	Control of Finite State Systems	237
7	Computational Complexity of the Synthesis of Memoryless Controllers with Partial Observation for Finite State Systems	239
7.1	Related Works	240
7.2	Review of Complexity Theory	241
7.3	Framework and Control Problems	242
7.4	Time Complexity of the Control Problems	245
7.4.1	The Basic Case	245
7.4.2	The Non-blocking and Deadlock Free Cases	256
7.5	Discussions	260
	Conclusion	261
	Summary	261
	Future Works	263
A	SMACS	265
A.1	Description of SMACS	265
A.2	Input Language of SMACS	267
A.3	Outputs of SMACS	275

Bibliography 279

Introduction

The real danger is not that computers will begin to think like men, but that men will begin to think like computers.

by *Sydney Harris.*

T is in the nature of humans to make *errors*. Fortunately, humans are able to learn and to reason, and they can then use these capacities to try to fix the errors they made. *Computers* do not have these abilities. Their role consists in executing *programs* or *sequences of instructions* written by humans and they are consequently subject to *human errors*. Since they are used in *critical applications* (e.g., banking systems, telecommunication networks, trains, planes, space rockets, medical equipments), a failure can lead to dramatic damages; let us just mention the following *famous* bugs:

- From 1985, several patients died due to the malfunction of a radiation therapy device called *Therac-25*. This malfunction was caused by a *race* condition in the software that controlled the device and because of this bug, a fast succession of operations could deliver a massive dose of radiations. The use of this device only stopped in 1987.
- The development cost of the *Ariane 5* space rocket is estimated at \$1 billion. The first test of this space rocket took place on June 4th, 1996, and the rocket was destroyed less than a minute after the launch. In fact, the rocket left its trajectory, because of an arithmetic overflow caused by a conversion of data from 64 bits floating point to 16 bits floating point, and as a consequence, the resulting aerodynamic forces caused the disintegration of the rocket.

These examples show that a bug in a computer system can have disastrous human and economical consequences. That is why, we need methods to rigorously validate or synthesize computer systems.

Obviously, humans cannot *manually* validate all computer systems, since the amount of lines of their source code can be huge (e.g., the flight program of Ariane 5 has more than 50000 lines of source code, the kernel of the Linux operating system has more than 1500000 lines of source code, ...). Therefore, since several decades, people work on *validation methods* that allow us to *(semi-)automatically* verify that a system behaves correctly and use computers to perform this verification.

Validation Methods

Validation methods are generally composed of three parts: *modeling*, *specification*, and *validation*:

- The first step consists in *describing* the system that must be verified. Generally, this description is either given by the source code of the system, or by a *formal* (i.e., *mathematical*) model extracted from the system. This model can be an *automaton* [ASU86], a Kripke structure [Kri63], a discrete event system [WR87], a hybrid system [ACH⁺95],...
- The specification step consists in describing the properties that the system must satisfy. These properties can be specified by predicates, by formulas of a *temporal logic* like the *linear time logic* (LTL) [Pnu77] or the *branching time logic* (CTL) [CE82], ...
- The third step is the validation, where the model is checked against the specification. Depending on the validation techniques, that are used, this step may consist in deciding if the model satisfies the specification, or in imposing the specification on the model by restricting the possible behaviors that it describes.

There are a lot of validation methods like *model checking*, *testing*, *controller synthesis*, ... In this thesis, we focus on the development of validation methods and, in particular, we are interested in controller synthesis methods.

Model Checking. A classical and well-known technique to validate a system is given by *model checking*. It has been introduced by Clarke and Emerson [CE82] and also independently by Queille and Sifakis [QS82]. This method works on a *model* of the system which is generally given by a *finite* state transition system like *Kripke Structure* [Kri63]. The *specification*, which gives the properties that

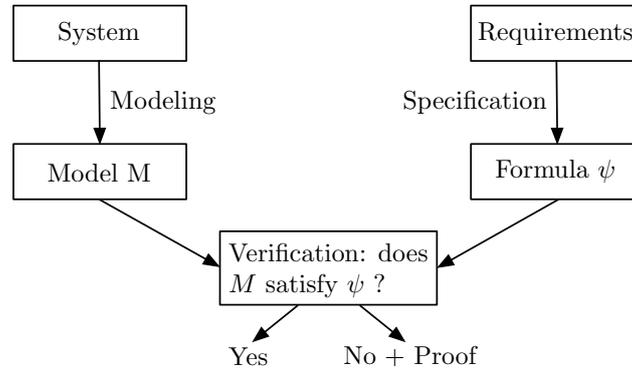


Figure 1 - The model checking process.

the system must satisfy to be correct, is generally specified by a formula of a *temporal logic* like the *linear time logic* (LTL) [VW86] or the *branching time logic* (CTL) [CE82]. Model checking techniques analyze the model of the system and produce an output which says if the system satisfies the given specification (see Figure 1). When the specification is not satisfied, a proof (e.g., a sequence of events leading to the violation of the specification), which shows that the system does not behave correctly, is given. This proof is often given as an error trace and a human assistance is then required to identify the source of the error and to fix it. Several model checking tools have been developed like SPIN [Hol04] and SMV [McM93]. When the model is extracted from a concurrent system, the number of states of the underlying transition system can be huge; this problem is known as the *state explosion problem*. It can then prevent from an exhaustive verification of the system, even with *efficient exploration techniques* like *partial order reduction* [God96, PVK01] or *symbolic model checking* [McM93]¹.

Testing. When model checking techniques cannot be used (for example, due to complexity reasons), we can turn into *testing methods* [Mye79], which also allow us to validate a system. There exist different testing methods, which depend on the assumptions made on the system. For example, if we assume that the source code of the system under test is available, a first step consists in instrumenting this source code to emit relevant events. Next, the system is executed and the events emitted by the system are collected to form a *trace*. Finally, the trace

¹Other techniques like *theorem proving* [Lov78] can be used to check the validity of a model w.r.t. a specification.

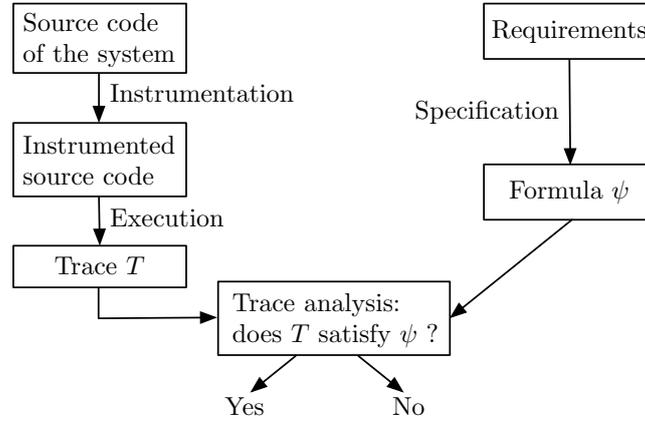


Figure 2 - The testing process.

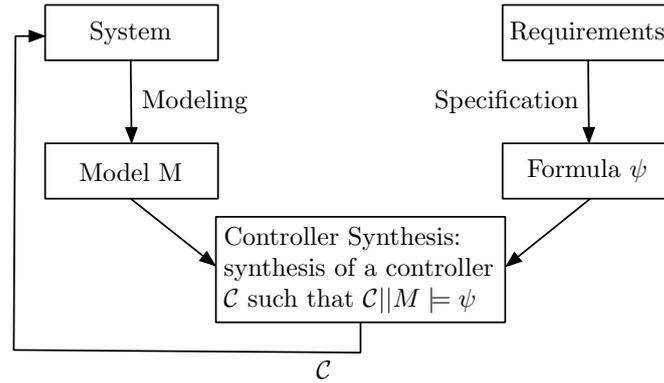


Figure 3 - The controller synthesis process.

is analyzed to determine if it satisfies a given specification (see Figure 2), and when an error is found, a human assistance is required to find the cause of the error and to correct it. The specification can be given by an LTL formula, a CTL formula, ... The testing is not an exhaustive method i.e., in general only a part of the system is analyzed. However, if it is performed on a large number of traces, we can have a reasonable confidence in the correctness of the system.

Controller Synthesis. In this thesis, our aim is to develop *control methods* for *discrete event systems*. A discrete event system is a system whose state space is given by a discrete set and whose state transition mechanism is *event-driven* i.e., its state evolution depends only on the occurrence of discrete events over the

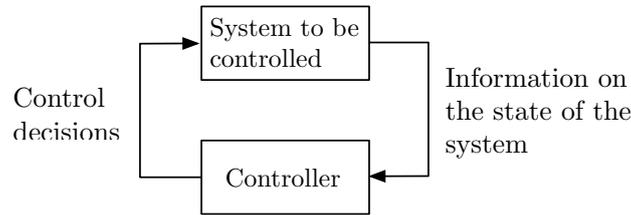


Figure 4 - Feedback interaction between the system to be controlled and the controller.

time. These systems are present in manufacturing production, robotic, vehicle traffic, telecommunication networks, ... Control methods work differently from model checking and testing methods to ensure the correctness of the system: the aim is not to verify that the system satisfies a given specification, but to impose it on the system by means of a *controller* (also called *supervisor*), which runs in parallel with the original system and which restricts its behavior (see Figure 3). The advantage of this method w.r.t. model checking and testing methods is that a human assistance is no more necessary to fix the errors of the system. Indeed, the controller restricts the behavior of the system to prevent it from generating erroneous behaviors. There are mainly two theories on the control of discrete event systems: *game theory* [PR89, Rei84, DDR06, CDHR07] and *supervisory control theory* [WR87, RW89, Won05, CL08]. In this thesis, we focus on the second theory.

- Game theory is the formal study of decision making in a *game* which models an interactive situation between several players, where a choice of a player may potentially affect the interests of the other players. A complete plan of actions of a player is called a *strategy*. This theory is used in several domains like biology, economics, philosophy, computer science (notably, for the controller synthesis), ... In game theory, the controller synthesis problem is often presented as a two-player game graph whose vertices represent the states of the system to be controlled. One of the players is called the *environment* and represents the actions of the system that cannot be inhibited and the other player is called the *controller* and represents the actions that can be forbidden in the system. The aim is then to compute a *winning strategy* which corresponds to a function that tells precisely what the controller has to do in order to satisfy the control requirements.
- In supervisory control theory, the behavior of the discrete event system to be

controlled is modeled by a language: the alphabet of this language models the different actions of the system and a word of this language models an execution of the system. The controller interacts with the system to be controlled in a feedback manner as depicted in Figure 4. More precisely, at each step of the execution of the system, the controller receives an observation from the system and computes, from this information, a set of actions, that the system must not execute in order to satisfy the control requirements. The control requirements are thus achieved by restricting the behavior of the system. In the literature, the supervisory control of *finite* discrete event systems received a lot of attention [WR87, Ush89, RW92b, BL98]. In that case, the language modeling the system is generally represented by a finite state automaton [WR87]. However, in many industrial applications, the state space of the model of the system can be infinite; for example, the state space is infinite when the model uses variables that take their values in a infinite domain. Unfortunately, the works on the control of finite discrete event systems cannot trivially be extended to infinite state systems, because, in this case, the control problems are generally *undecidable*. Most of the works on the control of infinite state systems ensure the termination of the computations by considering particular situations, in which the control problems are *decidable*. That is for example the case for the works on the control of infinite state systems modeled by Petri nets [Pet62, Kro87] or by timed automata [AD94, AMP95]. An alternative approach (that we use in this thesis) to overcome this undecidability consists in using abstract interpretation techniques [CC77]. Indeed, in most cases control problems can be solved by computing the least fixpoint of a function and the theory of abstract interpretation, developed by Cousot, allows us to approximate the evaluation of functions and of their fixpoints in a finite number of steps. Few works use this approach to solve control problems. We can nevertheless mention [LJM05], where abstract interpretation techniques are used to define an *effective algorithm* (i.e., which always terminates) synthesizing *memoryless* controllers (i.e., controllers that define their control decisions from the current observation received from the system) with *full observation* (i.e., the controllers distinguish all states of the system) that prevent the system from reaching a set of forbidden states.

Objectives and Structure of this Thesis

In this thesis, we want to develop control methods based on the supervisory control theory. In this theory, most of the works suppose that the system to be controlled has a *finite* state space. However, when modeling realistic systems, it is often convenient to manipulate state variables instead of simply atomic states, allowing a compact way to specify systems handling data. In that case, to provide a homogeneous treatment of these models, it is convenient to consider variables, whose domain is infinite, and thus systems with an infinite state space. Unfortunately, works on the control of finite state systems cannot trivially be extended to *infinite* state systems for undecidability reasons. Moreover, it is more interesting to consider the synthesis of controllers with *partial observation*. Indeed, controllers interact with the plant through sensors and actuators and, in general, they have only a partial observation of the system, because these materials have not an absolute precision or because some parts of the system are not observed by the controllers.

Therefore, in this thesis, we are interested in the control of infinite state systems with partial observation. This problem is generally undecidable and to overcome this negative result, we use, as in [LJM05], abstract interpretation techniques which ensure that our algorithm always terminate. Moreover, we want to provide the most complete contribution it is possible to bring to this topic. Hence, we consider more and more realistic problems. More precisely, we start our work by considering a *centralized framework* (i.e., the system is controlled by a single controller) and by synthesizing memoryless controllers (i.e., controllers that define their control policy from the current observation received from the system). Next, to obtain better solutions, we consider the synthesis of controllers that record a part or the whole of the execution of the system and use this information to define the control policy. However, in some cases, the nature of the system to be controlled makes unrealistic the use of a centralized framework; for example, in the case of distributed systems. We then consider, more elaborate frameworks, which allow us to control this kind of systems. We first work on a *decentralized framework* which is useful for the control of distributed systems with synchronous communications. In this framework, the system is controlled by several controllers, that are coordinated to jointly ensure the desired properties. After, we consider a *modular framework*, which actually corresponds to a decentralized framework where the structure of the system to be controlled is exploited to perform the computations more efficiently. Unfortunately, these

frameworks cannot be used for the control of an interesting and more complex class of distributed systems: the distributed systems with asynchronous communications. Therefore, we define a *distributed framework* which allows us to control this kind of systems.

This thesis is structured as follows.

Chapter 1 - Preliminaries. In this chapter, we briefly present fundamental concepts and existing results that we will use throughout this thesis.

Chapter 2 - Supervisory Control of Discrete Event Systems. In this chapter, we present a state of the art on supervisory control theory of finite discrete event systems and of infinite state systems.

Chapter 3 - Synthesis of Centralized Memoryless Controllers with Partial Observation for Infinite State Systems. In chapter 3, we are interested in the *state avoidance control problem* for infinite state systems. This problem consists in synthesizing a controller that prevents the system from reaching a set of forbidden states. In our framework, we model infinite state systems by *symbolic transition systems* [HMR05]. This formalism allows us to compactly manipulate infinite state systems. Moreover, we place ourself in the context of controllers that are memoryless and that do not perfectly observe the system to be controlled. This assumption of *partial observation* makes the problem harder, but also more realistic, because the sensors, that the controllers use to observe the system, have generally neither an absolute precision, nor a full observation of the system. We provide algorithms, which synthesize controllers with partial observation for the state avoidance control problem for both cases where the deadlock free property must and must not be ensured. Since the problems, that we want to solve, are undecidable, we use, as in [LJM05], abstract interpretation techniques to obtain effective algorithms at the price, however, of some approximations in the solutions that we compute. The content of this chapter is based on the followings article:

G. Kalyon, T. Le Gall, H. Marchand, T. Massart. Control of Infinite Symbolic Transition Systems under Partial Observation. *In European Control Conference (ECC'09)*, pages 1456-1462, Budapest, Hungary, August 2009.

Chapter 4 - Synthesis of Centralized k -memory and Online Controllers with Partial Observation for Infinite State Systems. In this chapter,

we extend the previous work by synthesizing *k-memory controllers* and *online controllers*. A *k-memory controller* uses the last *k* observations received from the system to define its control decisions and an online controller computes, during the execution of the system, an estimate of the current state of the system that it uses to define its control policy. We also compare the control quality of our memoryless, *k-memory*, and online controllers. The content of this chapter is based on the following article:

G. Kalyon, T. Le Gall, H. Marchand and T. Massart. Symbolic Supervisory Control of Infinite Transition Systems under Partial Observation using Abstract Interpretation. *Submitted to Journal of Discrete Event Dynamical Systems*.

Chapter 5 - Synthesis of Decentralized and Modular Controllers for Infinite State Systems. In this chapter, we provide algorithms computing decentralized controllers for infinite state systems. In a decentralized framework, the system is controlled by *n* controllers, that are coordinated to jointly ensure the desired properties. This framework is useful for the control of more complex systems like the distributed systems with synchronous communications. We consider the state avoidance control problem for both cases where the deadlock free property must and must not be ensured. Again, we use abstract interpretation techniques to ensure the termination of the computations, since these problems are undecidable. However, in this framework, we have no information regarding the structure of the distributed system to be controlled while the computation of the controllers, which implies that these computations are performed on the model of the global system. We then adapt this approach to the case where the structure of the distributed system to be controlled is known. We explain how this structure can be exploited to perform most of the computations locally i.e., on each subsystem which composes the global system to be controlled. The content of this chapter is based on the following articles:

G. Kalyon, T. Le Gall, H. Marchand, and T. Massart. Contrôle Décentralisé de Systèmes Symboliques Infinis sous Observation Partielle. *Journal Européen des Systèmes Automatisés (7ème Colloque Francophone sur la Modélisation des Systèmes Réactifs)*, 43/7-9-2009:805-819, 2009.

G. Kalyon, T. Le Gall, H. Marchand and T. Massart. Decentralized Control of Infinite State Systems. *Submitted to Journal of Discrete Event Dynamical Systems*.

Chapter 6 - Distributed Supervisory Control of FIFO Systems. In chapter 6, we consider the state avoidance control problem in a distributed framework. In this context, the system to be controlled is defined by n subsystems communicating asynchronously through reliable unbounded FIFO channels. Each subsystem is modeled by a *communicating finite state machine* [BZ83a] and the state space of the global system can be infinite, because the queue are unbounded. We define an algorithm which computes n local controllers (each local controller controls a subsystem) that prevent the global system from reaching a set of forbidden states. The controllers, that we define, can exchange, during the execution of the system, information through reliable unbounded FIFO channels. They exploit this information to compute an estimate of the current state of the system that they use to define their control policy. Once again, since the problem, that we consider, is undecidable, we use abstract interpretation techniques to ensure the termination of the computations. The content of this chapter is based on the following technical report:

B. Jeannet, G. Kalyon, T. Le Gall, H. Marchand and T. Massart. Distributed Supervisory Control of FIFO Systems. *Technical report*, ULB, 2010.

Chapter 7 - Computational Complexity of the Synthesis of Memoryless Controllers with Partial Observation for Finite State Systems. In this chapter, we restrict our attention to the case, where the system to be controlled has a finite state space, and we study the time complexity of some control problems related to the state avoidance control problem. In particular, we show that the problem, which consists in computing a maximal solution for the state avoidance control problem, cannot be solved in polynomial time unless $P = NP$. The content of this chapter is based on the following article:

G. Kalyon, T. Le Gall, H. Marchand, and T. Massart. Computational Complexity for State-feedback Controllers with Partial Observation. In *7th International Conference on Control and Automation (ICCA'09)*, pages 436-441, Christchurch, New Zealand, December 2009.

Appendix A - Smacs (Symbolic MAsked Controller Synthesis). In this chapter, we describe our tool SMACS which implements some control algorithms defined in this thesis. This tool allows us to experimentally evaluate our approach based on the use of abstract interpretation to obtain effective algorithms.

Finally, we conclude this thesis by summarizing our work and by suggesting future works.

Part I

Fundamentals and Definitions

Chapter 1

Preliminaries

 IN this chapter, we recall the key concepts used throughout this thesis. First, in section 1.1, we present some basic notions on sets, functions, predicates,... Next, in section 1.2, we briefly recall the concepts of languages and of automata. In section 1.3, we give a short introduction to lattice theory. Finally, in section 1.4, we briefly present the abstract interpretation framework. Abstract interpretation will be widely used in this thesis to obtain *effective* control algorithms (i.e., control algorithms which always terminate) for infinite state systems.

1.1 Basic Notions

In this section, we recall some basic notions and notations that will be used throughout this thesis.

Sets. \mathbb{N} denotes the set of *natural numbers* $\{1, 2, \dots\}$ and \mathbb{N}_0 denotes the set $\{0, 1, 2, \dots\}$. The set of *integer* numbers $\{0, 1, 2, \dots\} \cup \{-1, -2, \dots\}$ is denoted by \mathbb{Z} . The set of *rational* (resp. *real*) numbers is denoted by \mathbb{Q} (resp. \mathbb{R}). \mathbb{B} denotes the set of boolean values $\{\text{tt}, \text{ff}\}$, where **tt** (resp. **ff**) stands for *true* (resp. *false*). Given two numbers $a, b \in \mathbb{Z} \cup \{-\infty, \infty\}$, the *interval* between a and b is denoted by $[a, b]$ and is defined by $[a, b] \stackrel{\text{def}}{=} \{x \in \mathbb{Z} \cup \{-\infty, \infty\} \mid a \leq x \leq b\}$. Given a set X , 2^X denotes the *power set* of X i.e., the set of all subsets of X ,

and $|X| \in \mathbb{N}_0 \cup \{\infty\}$ denotes the *cardinality* of X i.e., the number of elements in X . Given a set X and a subset $Y \subseteq X$, the *complement* of Y w.r.t. X is denoted by Y^c and is defined by $Y^c \stackrel{\text{def}}{=} X \setminus Y$. Given two sets X and Y , the *cartesian product* of X and Y is denoted by $X \times Y$. Given a set X and a number $n \geq 1$, X^n denotes the cartesian product $\underbrace{X \times \dots \times X}_n$, which gives the set of n -tuples of elements of X . The *empty set* is denoted by \emptyset .

Coverings and Partitions. Given a set X , a *covering* C of X is a set of non-empty subsets of X such that $\bigcup_{Y \in C} Y = X$ and a *partition* P of X is a covering such that $\forall Y, Z \in P : Y \cap Z = \emptyset$. To express that the sets X_1, X_2, \dots, X_n constitute a partition P of X , we write $X = X_1 \uplus X_2 \uplus \dots \uplus X_n$.

Tuples. Given an n -tuple $\vec{a} = \langle a_1, \dots, a_n \rangle$ and two numbers $i, j \in [1, n]$ with $i \leq j$, $\vec{a}|_{[i:j]}$ denotes the tuple made of the components of \vec{a} between a_i and a_j i.e., $\vec{a}|_{[i:j]} \stackrel{\text{def}}{=} \langle a_i, \dots, a_j \rangle$. When $i = j$, $\vec{a}|_{[i:i]}$ is denoted by $\vec{a}|_{[i]}$. This notation is extended to sets of n -tuples as expected: given a set \vec{X} of n -tuples, $\vec{X}|_{[i:j]} \stackrel{\text{def}}{=} \{\vec{x}|_{[i:j]} \mid \vec{x} \in \vec{X}\}$. Given two n -tuples \vec{a}_1 and \vec{a}_2 , the *concatenation* of these two n -tuples is denoted by $\vec{a}_1 \cup \vec{a}_2$ and is defined by the *pair* $\vec{a}_1 \cup \vec{a}_2 \stackrel{\text{def}}{=} \langle \vec{a}_1, \vec{a}_2 \rangle$. The empty tuple is denoted by \emptyset .

Functions. A *function* f from a set X , called the *domain*, to a set Y , called the *codomain*, is denoted by $f : X \rightarrow Y$ and is a relation between X and Y , which associates at most one element $y \in Y$ with each element $x \in X$; this element y is denoted by $f(x)$. A function $f : X \rightarrow Y$ is *total* if, for each $x \in X$, $|f(x)| = 1$; otherwise it is called *partial*. A function $f : X \rightarrow Y$ is (i) *injective* if $\forall x_1, x_2 \in X : (f(x_1) = f(x_2)) \Rightarrow (x_1 = x_2)$, (ii) *surjective* if $\forall y \in Y, \exists x \in X : f(x) = y$, and (iii) *bijective* if it is injective and surjective. Given a function $f : X \rightarrow Y$ and a set $Z \subseteq X$, $f(Z)$ denotes $\bigcup_{z \in Z} f(z)$. Given a set X , $\text{ld}_X : X \rightarrow X$ denotes the *identity function*, which is defined, for each $x \in X$, by $\text{ld}_X(x) \stackrel{\text{def}}{=} x$. The *preimage function* of a function $f : X \rightarrow Y$ is denoted by $f^{-1} : Y \rightarrow 2^X$ and is defined, for each $y \in Y$, by $f^{-1}(y) \stackrel{\text{def}}{=} \{x \in X \mid f(x) = y\}$. The *composition* $g \circ f$ of two functions $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ is a function from the domain X to the codomain Z and is defined, for each $x \in X$, by $(g \circ f)(x) \stackrel{\text{def}}{=} g(f(x))$. Given a function $f : X \rightarrow X$ and a number $n \in \mathbb{N}_0$, the n^{th} *functional power* of f is denoted by f^n and is recursively defined as follows: (i) $f^0 \stackrel{\text{def}}{=} \text{ld}_X$, and (ii) $f^n \stackrel{\text{def}}{=} f^{n-1} \circ f$,

if $n \geq 1$. We sometimes use the Church's notation [Chu36] where a function $f : X \rightarrow Y$ is denoted by $\lambda x.f(x)$.

Predicates. Let X be a set, a *predicate* $P : X \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ over the set X is a function, which associates a *truth value* belonging to $\{\mathbf{tt}, \mathbf{ff}\}$ with each element $x \in X$; this truth value is denoted by $P(x)$. $P(x) = \mathbf{tt}$ and $P(x) = \mathbf{ff}$ are respectively denoted by $P(x)$ and $\neg P(x)$. The predicate $P : X \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ can also be viewed as a subset $Y \subseteq X$ defined by $Y \stackrel{\text{def}}{=} \{x \in X \mid P(x)\}$. Given a set X , $\mathbb{T}_X : X \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ denotes the *true predicate*, which is defined, for each $x \in X$, by $\mathbb{T}_X(x) = \mathbf{tt}$. Similarly, $\mathbb{F}_X : X \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ denotes the *false predicate*, which is defined, for each $x \in X$, by $\mathbb{F}_X(x) = \mathbf{ff}$.

Variables. The domain of a variable v is denoted by \mathcal{D}_v , and the domain of an n -tuple of variables $V = \langle v_1, \dots, v_n \rangle$ is denoted by \mathcal{D}_V and is defined by $\mathcal{D}_V \stackrel{\text{def}}{=} \prod_{i \in [1, n]} \mathcal{D}_{v_i}$. Throughout this thesis, we will frequently use \vec{v} to denote a value $\langle \nu_1, \dots, \nu_n \rangle \in \mathcal{D}_V$.

Relations. A *binary relation* $R \subseteq X \times X$ is (i) *reflexive* if $\forall x \in X : \langle x, x \rangle \in R$, (ii) *symmetric* if $\forall x, y \in X : (\langle x, y \rangle \in R) \Rightarrow (\langle y, x \rangle \in R)$, (iii) *antisymmetric* if $\forall x, y \in X : ((\langle x, y \rangle \in R) \wedge (\langle y, x \rangle \in R)) \Rightarrow (x = y)$, and (iv) *transitive* if $\forall x, y, z \in X : ((\langle x, y \rangle \in R) \wedge (\langle y, z \rangle \in R)) \Rightarrow (\langle x, z \rangle \in R)$. A *partial order* is a reflexive, antisymmetric and transitive binary relation and an *equivalence relation* is a reflexive, symmetric and transitive binary relation.

1.2 Languages and Automata

Definition of a Language. An *alphabet* Σ is a *finite* set of symbols called *letters*. A *word* w over Σ is a sequence of letters of Σ ; the empty word is denoted by ϵ . The set of finite words over Σ is denoted by Σ^* and a *language* over Σ is defined as a subset $\mathcal{L} \subseteq \Sigma^*$ of finite words. The key operation to build words from an alphabet Σ is the *concatenation*: given two words $w_1, w_2 \in \Sigma^*$, the concatenation of w_1 and w_2 is denoted by $w_1.w_2$ (or w_1w_2 for short) and is defined by a new word composed of the letters of w_1 immediately followed by the letters of w_2 . Given a subset $\Sigma' \subseteq \Sigma$, the *projection* of a word $w \in \Sigma^*$ on Σ' is obtained from w by keeping only the symbols of Σ' . This operation is formally

defined by the function $P : \Sigma^* \rightarrow \Sigma'^*$, where, for each $w \in \Sigma^*$:

$$P(w) \stackrel{\text{def}}{=} \begin{cases} \epsilon & \text{if } w = \epsilon \\ P(w').a & \text{if } (w = w'.a) \wedge (a \in \Sigma') \\ P(w') & \text{if } (w = w'.a) \wedge (a \notin \Sigma') \end{cases} \quad (1.1)$$

The *prefix-closure* of a language \mathcal{L} (over Σ) is a language denoted by $\bar{\mathcal{L}}$ and defined by all *prefixes* of all words of \mathcal{L} i.e., $\bar{\mathcal{L}} \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid \exists w' \in \Sigma^* : w.w' \in \mathcal{L}\}$.

Automata. An *automaton* is a well-known formalism which models languages and which corresponds to a directed graph with *actions* (also called *labels*) on the edges. This concept is formally defined as follows:

Definition 1.1 (Automaton)

An *automaton* \mathcal{T} is defined by a 5-tuple $\langle X, X_0, X_m, \Sigma, \Delta \rangle$, where¹:

- X is a countable set of states
- X_0 is the set of *initial* states
- X_m is a set of *marked* (or *final* or *accepting*) states
- Σ is a finite *set of actions* (or *alphabet*)
- $\Delta \subseteq X \times \Sigma \times X$ is the *transition relation*

An automaton $\mathcal{T} = \langle X, X_0, X_m, \Sigma, \Delta \rangle$ is *deterministic*² if (i) $|X_0| = 1$ and (ii) $\forall x_1 \in X, \forall \sigma \in \Sigma : [(\langle x_1, \sigma, x_2 \rangle \in \Delta) \wedge (\langle x_1, \sigma, x'_2 \rangle \in \Delta)] \Rightarrow (x_2 = x'_2)$. Moreover, \mathcal{T} is *finite* if the set X of states is finite. A finite automaton can be transformed into a finite deterministic automaton by using the subset construction [HMU06]. In the sequel, a transition $\langle x_1, \sigma, x_2 \rangle \in \Delta$ is also denoted by $x_1 \xrightarrow{\sigma} x_2$.

Example 1.1

Figure 1.1 shows a graphical representation of a finite automaton $\mathcal{T} = \langle X, X_0, X_m, \Sigma, \Delta \rangle$. In this diagram, states and transitions are respectively represented by nodes and edges. The initial states are identified by a small incoming arrow and the final states are doubly circle nodes. Thus, the automaton \mathcal{T} represented by this diagram is defined as follows:

¹In the sequel, when we will not need the set of marked states X_m , we will define an automaton by a 4-tuple $\langle X, X_0, \Sigma, \Delta \rangle$.

²In some references [ASU86], an equivalent definition is given: (i) $|X_0| = 1$ and (ii) $\forall x_1 \in X, \forall \sigma \in \Sigma : |\{\langle x_1, \sigma, x_2 \rangle \mid \langle x_1, \sigma, x_2 \rangle \in \Delta\}| = 1$.

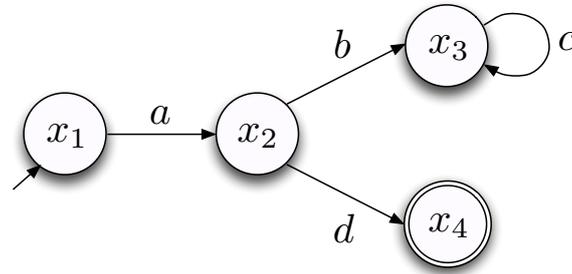


Figure 1.1 - An example of finite deterministic automata.

- $X = \{x_1, x_2, x_3, x_4\}$
- $X_0 = \{x_1\}$
- $X_m = \{x_4\}$
- $\Sigma = \{a, b, c, d\}$
- $\Delta = \{\langle x_1, a, x_2 \rangle, \langle x_2, b, x_3 \rangle, \langle x_2, d, x_4 \rangle, \langle x_3, c, x_3 \rangle\}$

We now define some notions, related to automata, that are often used in the control of discrete event systems. For convenience, we suppose that we work on an automaton $\mathcal{T} = \langle X, X_0, X_m, \Sigma, \Delta \rangle$.

Generated and Marked Languages. There are mainly two languages which are associated to an automaton: the *generated* and *marked* languages.

Definition 1.2 (Generated Language)

A finite word $w = \sigma_1.\sigma_2 \dots \sigma_n$ is *generated* by an automaton \mathcal{T} if there exists a finite sequence of states $x_0, x_1, \dots, x_n \in X$ such that (i) $x_0 \in X_0$, and (ii) $\forall i \in [0, n - 1] : \langle x_i, \sigma_{i+1}, x_{i+1} \rangle \in \Delta$. The *language* $\mathcal{L}(\mathcal{T})$ generated by \mathcal{T} is then defined by the set of finite words w generated by \mathcal{T} .

The generated language thus describes the set of possible behaviors of an automaton and is obviously prefix-closed.

Definition 1.3 (Marked Language)

A finite word $w = \sigma_1.\sigma_2 \dots \sigma_n$ is *marked* by an automaton \mathcal{T} if there exists a finite sequence of states $x_0, x_1, \dots, x_n \in X$ such that (i) $x_0 \in X_0$, (ii) $\forall i \in [0, n-1] : \langle x_i, \sigma_{i+1}, x_{i+1} \rangle \in \Delta$ and (iii) $x_n \in X_m$. The *language* $\mathcal{L}_m(\mathcal{T})$ *marked* by \mathcal{T} is then defined by the set of finite words w marked by \mathcal{T} .

The marked language thus describes the set of possible behaviors of an automaton which end in a final state. For example, if the final states correspond to the states in which a task is accomplished, the marked language then represents the set of behaviors corresponding to the achievement of a task.

Regular Languages. *Regular languages* form an interesting class of languages defined as follows:

Definition 1.4 (Regular Language)

A language \mathcal{L} is *regular* if it is marked by a finite automaton.

Further in this thesis, we will use this concept in the control of distributed systems to abstract the content of the queues of these systems.

Reachable and Coreachable States. The concepts of *reachable* and *coreachable* states are defined in the following way for automata.

Definition 1.5 (Reachable State)

A state x' is *reachable* from a state x in an automaton \mathcal{T} if x' can be reached from x by a *finite* sequence of transitions i.e., there exist a *finite* sequence of states $x_0, x_1, \dots, x_n \in X$ and a finite sequence of actions $\sigma_1, \sigma_2, \dots, \sigma_n \in \Sigma$ such that (i) $x_0 = x$, (ii) $x_n = x'$, and (iii) $\forall i \in [0, n-1] : \langle x_i, \sigma_{i+1}, x_{i+1} \rangle \in \Delta$. A state x is *reachable in the system* \mathcal{T} if it is reachable from an initial state of \mathcal{T} .

Definition 1.6 (Coreachable State)

A state x' is *coreachable* from a state x in an automaton \mathcal{T} if x can be reached from x' by a *finite* sequence of transitions i.e., there exist a *finite* sequence of states $x_0, x_1, \dots, x_n \in X$ and a finite sequence of actions $\sigma_1, \sigma_2, \dots, \sigma_n \in \Sigma$ such that (i) $x_0 = x'$, (ii) $x_n = x$, and (iii) $\forall i \in [0, n-1] : \langle x_i, \sigma_{i+1}, x_{i+1} \rangle \in \Delta$.

Backward and Forward Operators. The backward and forward operators

are defined in the following way:

Definition 1.7 (Backward Operator)

Given a subset of actions $A \subseteq \Sigma$, the *backward operator* $\text{Pre}_A^T : 2^X \rightarrow 2^X$ is defined, for each $B \subseteq X$, by $\text{Pre}_A^T(B) \stackrel{\text{def}}{=} \bigcup_{\sigma \in A} \{x_1 \in X \mid \exists x_2 \in B : \langle x_1, \sigma, x_2 \rangle \in \Delta\}$.

Thus, $\text{Pre}_A^T(B)$ gives the set of states that lead to B by a transition labeled by an action of A .

Definition 1.8 (Forward Operator)

Given a subset of actions $A \subseteq \Sigma$, the *forward operator* $\text{Post}_A^T : 2^X \rightarrow 2^X$ is defined, for each $B \subseteq X$, by $\text{Post}_A^T(B) \stackrel{\text{def}}{=} \bigcup_{\sigma \in A} \{x_1 \in X \mid \exists x_2 \in B : \langle x_2, \sigma, x_1 \rangle \in \Delta\}$.

Thus, $\text{Post}_A^T(B)$ gives the set of states that are reachable from B by a transition labeled by an action of A .

Coreachability and Reachability Operators. The reachability and coreachability operators are defined as follows:

Definition 1.9 (Coreachability Operator)

Given a subset of actions $A \subseteq \Sigma$, the *coreachability operator* $\text{Coreach}_A^T : 2^X \rightarrow 2^X$ is defined, for each $B \subseteq X$, by $\text{Coreach}_A^T(B) \stackrel{\text{def}}{=} \bigcup_{n \geq 0} (\text{Pre}_A^T)^n(B)$.

Thus, $\text{Coreach}_A^T(B)$ gives the set of states that lead to B by a sequence of transitions labeled by actions of A .

Definition 1.10 (Reachability Operator)

Given a subset of actions $A \subseteq \Sigma$, the *reachability operator* $\text{Reachable}_A^T : 2^X \rightarrow 2^X$ is defined, for each $B \subseteq X$, by $\text{Reachable}_A^T(B) \stackrel{\text{def}}{=} \bigcup_{n \geq 0} (\text{Post}_A^T)^n(B)$.

Thus, $\text{Reachable}_A^T(B)$ gives the set of states that are reachable from B by a sequence of transitions labeled by actions of A .

Properties. The *deadlock free* and *non-blocking* properties are two interesting concepts for automata:

Definition 1.11 (Deadlock Free Property)

A state $x_1 \in X$ is *deadlock free* if $\exists \sigma \in \Sigma, \exists x_2 \in X : \langle x_1, \sigma, x_2 \rangle \in \Delta$, and an automaton \mathcal{T} is *deadlock free* if all states, that are *reachable* in this automaton, are deadlock free.

This property means that it is always possible to fire at least one action from any reachable state.

Definition 1.12 (Non-blocking Property)

A state $x \in X$ is *non-blocking* if $(\text{Reachable}_{\Sigma}^{\mathcal{T}}(x)) \cap X_m \neq \emptyset$. An automaton \mathcal{T} is *non-blocking* if all states, that are reachable in this automaton, are non-blocking; in this case, $\overline{\mathcal{L}_m(\mathcal{T})} = \mathcal{L}(\mathcal{T})$.

This property means that a marked state can always be reached from any reachable state. Notice that a non-blocking automaton can have deadlocking marked states.

1.3 Lattice Theory

In this section, we recall some basic notions on lattice theory. First, in section 1.3.1, we introduce the concepts of partial orders and of lattices, and next, in section 1.3.2, we turn our attention to fixpoint computations.

1.3.1 Partially Ordered Set and Lattice

The notion of *partially ordered set* is defined as follows:

Definition 1.13 (Partially Ordered Set)

Let X be a set and $\sqsubseteq \subseteq X \times X$ be a binary relation. The pair $\langle X, \sqsubseteq \rangle$ is a *partially ordered set* if \sqsubseteq is a partial order on X .

For example, $\langle \mathbb{N}_0, \leq \rangle$ and $\langle \mathbb{Z}, \leq \rangle$ are partially ordered sets.

Let $Y \subseteq X$ be a subset of X and $\sqsubseteq \subseteq X \times X$ be a partial order. An element $x \in X$ is an *upper bound* (resp. *lower bound*) of Y if $\forall y \in Y : y \sqsubseteq x$ (resp. $x \sqsubseteq y$). Moreover, x is the *least upper bound* of Y (i) if it is an upper bound of Y , and (ii) if $x \sqsubseteq y$, for every upper bound y of Y . Similarly, x is the *greatest lower bound* of Y (i) if it is a lower bound of Y , and (ii) if $y \sqsubseteq x$, for every lower bound y of Y . The least upper bound and the greatest lower bound of a set Y do

not always exist. When they exist, they are unique (since \sqsubseteq is antisymmetric) and they are respectively denoted by $\bigsqcup Y$ and $\bigsqcap Y$. The operator \bigsqcup (resp. \bigsqcap) is also called the *join operator* (resp. *meet operator*) and $\bigsqcup\{x, y\}$ (resp. $\bigsqcap\{x, y\}$) is written $x \sqcup y$ (resp. $x \sqcap y$).

The notion of *lattice* is defined as follows:

Definition 1.14 (Lattice)

Let X be a set and $\sqsubseteq \subseteq X \times X$ be a partial order. The pair $\langle X, \sqsubseteq \rangle$ is a *lattice* if, for any $x, y \in X$, the least upper bound and the greatest lower bound of the set $\{x, y\}$ are defined.

Definition 1.15 (Complete Lattice)

Let X be a set and $\sqsubseteq \subseteq X \times X$ be a partial order. The element $\langle X, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \perp \rangle$ is a *complete lattice* if every non-empty subset $Y \subseteq X$ has a least upper bound $\bigsqcup Y$ and a greatest lower bound $\bigsqcap Y$. In particular, a complete lattice admits a \sqsubseteq -maximal element (also called *top*) $\top = \bigsqcup X$ and a \sqsubseteq -minimal element (also called *bottom*) $\perp = \bigsqcap X$.

Example 1.2

Let X be a set. The *power set lattice* $\text{PSL}(X) = \langle 2^X, \subseteq, \cup, \cap, X, \emptyset \rangle$ is a complete lattice, which has (i) the union as least upper bound, (ii) the intersection as greatest lower bound, (iii) the set X as \subseteq -maximal element, and (iv) the set \emptyset as \subseteq -minimal element.

1.3.2 Fixpoints

Given a set X and a function $f : X \rightarrow X$ over the lattice $\langle X, \sqsubseteq \rangle$, an element $x \in X$ is a *fixpoint* of f if $x = f(x)$. We define $\text{FP}(f) \stackrel{\text{def}}{=} \{x \in X \mid f(x) = x\}$. The *least fixpoint* of f (denoted by $\text{lfp}^{\sqsubseteq}(f)$) is the greatest lower bound of $\text{FP}(f)$ i.e., $\text{lfp}^{\sqsubseteq}(f) \stackrel{\text{def}}{=} \bigsqcap(\text{FP}(f))$. Similarly, the *greatest fixpoint* of f (denoted by $\text{gfp}^{\sqsubseteq}(f)$) is the least upper bound of $\text{FP}(f)$ i.e., $\text{gfp}^{\sqsubseteq}(f) \stackrel{\text{def}}{=} \bigsqcup(\text{FP}(f))$. An element $x \in X$ is a *post-fixpoint* (resp. *pre-fixpoint*) if $x \sqsupseteq f(x)$ (resp. $x \sqsubseteq f(x)$).

We present two well-known theorems on fixpoints. But before that, we define the concepts of *monotonic* and *continuous* functions.

Definition 1.16 (Monotonic and Continuous Functions)

Let $f : X \rightarrow X$ be a function over the complete lattice $\langle X, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$. This function is:

- *monotonic* if $\forall x, y \in X : (x \sqsubseteq y) \Rightarrow (f(x) \sqsubseteq f(y))$.
- *continuous* if, for every subset $Y = \{y_1, y_2, \dots, y_n, \dots\} \sqsubseteq X$ such that $y_1 \sqsubseteq y_2 \sqsubseteq \dots \sqsubseteq y_n \sqsubseteq \dots$, we have that $f(\sqcup Y) = \sqcup\{f(y) \mid y \in Y\}$ and $f(\sqcap Y) = \sqcap\{f(y) \mid y \in Y\}$.

Note that a continuous function is also monotonic. When X is finite, we have an equivalence between these two concepts.

The Knaster-Tarski's theorem states that each monotonic function f over a complete lattice admits a least fixpoint.

Theorem 1.1 (Knaster-Tarski's Theorem [Kna28, Tar55])

Let $f : X \rightarrow X$ be a monotonic function over the complete lattice $\langle X, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$. Then, the following characterization of the least fixpoint of f holds:

$$\text{lfp}^{\sqsubseteq}(f) \stackrel{\text{def}}{=} \sqcap \{x \in X \mid f(x) \sqsubseteq x\} \quad (1.2)$$

Unlike the Knaster-Tarski's theorem, the Kleene's theorem provides an effective computation of $\text{lfp}^{\sqsubseteq}(f)$:

Theorem 1.2 (Kleene's Theorem [Mar97])

Let $f : X \rightarrow X$ be a continuous function over the complete lattice $\langle X, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$. Then, $\text{lfp}^{\sqsubseteq}(f)$ is the limit of the following sequence:

$$u_i \stackrel{\text{def}}{=} \begin{cases} \perp & \text{if } i = 0 \\ u_{i-1} \sqcup f(u_{i-1}) & \text{if } i > 0 \end{cases}$$

This iterative computation is only a semi-algorithm, because the computation may not terminate when the complete lattice $\langle X, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$ does not satisfy the *ascending chain condition*. A lattice satisfies the *ascending chain condition* if each ascending sequence $u_0 \sqsubseteq u_1 \sqsubseteq \dots \sqsubseteq u_n \sqsubseteq \dots$ stabilizes after a finite number of steps i.e., for each of these sequences, there is some number $i \in \mathbb{N}_0$ such that $u_i = u_j$ for all $j \geq i$. When the termination cannot be ensured, we must find

a way to overapproximate $\text{lfp}^{\sqsubseteq}(f)$. For that, we may use *abstract interpretation techniques* (see section 1.4), which allow us to compute a sequence which always stabilizes after a finite number of steps.

One can note that results similar to Theorems 1.1 and 1.2 exist for $\text{gfp}^{\sqsubseteq}(f)$.

1.4 Abstract Interpretation

In general, verification problems can be solved by computing the least (or greatest) fixpoint of some functions; the computation of the set of reachable states in a system is a classical example. By the Kleene's theorem, these fixpoints can be solved by an iterative algorithm i.e., successive values of the solution are computed until stabilization. Unfortunately, for many interesting lattices, this algorithm may not terminate, because the number of iterations can be infinite or too big to allow an efficient analysis. Abstract interpretation allows us to compute an overapproximation of these fixpoints. For that, it substitutes the computations in the concrete lattice by computations in a simpler *abstract lattice*. Moreover, to ensure the termination of the computations, it provides a powerful tool, the *widening operator*, which attempts to find an invariant in a finite number of iterations.

In the remaining part of this section, we present the general framework of abstract interpretation, the *Galois connection framework*, where the concrete and abstract lattices are supposed to be complete. Next, we present the *representation framework*, which can be used when the abstract lattice is not complete; we will use this framework in this thesis.

1.4.1 The Galois Connection Framework

The Galois connection framework is the general framework of abstract interpretation and it is based on two key concepts: the *Galois connection* and the *widening operator* [CC77].

Galois Connection. The concrete lattice $\langle X, \sqsubseteq \rangle$ is substituted by an *abstract lattice* $\langle \Lambda, \sqsubseteq^{\#} \rangle$ (the abstract lattice can be seen as a simplification of the concrete lattice, which allows us to describe some particular subsets of the state space only). Both lattices must be complete and they are linked by a *Galois connection* $\langle \alpha, \gamma \rangle$, where the monotonic *abstraction function* $\alpha : X \rightarrow \Lambda$ and the monotonic

concretization function $\gamma : \Lambda \rightarrow X$ satisfy the following property [CC77]:

$$\forall x \in X, \forall \ell \in \Lambda : \alpha(x) \sqsubseteq^\# \ell \Leftrightarrow x \sqsubseteq \gamma(\ell) \quad (1.3)$$

The abstraction function can be seen as a function which *attempts to approximate* the sets of concrete states and the concretization function can be seen as a function which *gives a meaning* to these approximate sets.

The Galois connection between the complete lattices $\langle X, \sqsubseteq \rangle$ and $\langle \Lambda, \sqsubseteq^\# \rangle$ is generally denoted by $\langle X, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \Lambda, \sqsubseteq^\# \rangle$ and ensures the computation of an overapproximation of the least fixpoint $\text{lfp}^\square(f)$ of a function f . Indeed, instead of computing the least fixpoint of the function f (concrete lattice), we compute the least fixpoint of the function $f^\# \stackrel{\text{def}}{=} \alpha \circ f \circ \gamma$, which always satisfies the following property: $\text{lfp}^\square(f) \sqsubseteq \gamma(\text{lfp}^{\square^\#}(f^\#))$. Thus, the concretization of the least fixpoint $\text{lfp}^{\square^\#}(f^\#)$ gives an overapproximation of $\text{lfp}^\square(f)$.

Example 1.3

Let us consider the power set lattice $\text{PSL}(\mathbb{Z}) = \langle 2^{\mathbb{Z}}, \subseteq, \cup, \cap, \mathbb{Z}, \perp \rangle$ and the lattice of *intervals* $\langle \mathcal{I}, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$, where (i) $\mathcal{I} = \{[a, b] \mid (a \in \mathbb{Z} \cup \{-\infty\}) \wedge (b \in \mathbb{Z} \cup \{\infty\}) \wedge (a \leq b)\} \cup \{\emptyset\}$, (ii) $\sqsubseteq, \sqcup, \sqcap$ are respectively the classical inclusion, union and intersection on intervals, (iii) $\top = [-\infty, \infty]$, and (iv) $\perp = \emptyset$. Both lattices are complete and they can be linked by the following Galois connection $\langle 2^{\mathbb{Z}}, \subseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{I}, \sqsubseteq \rangle$, where (i) the concretization function $\gamma : \mathcal{I} \rightarrow 2^{\mathbb{Z}}$ is defined, for each $I \in \mathcal{I}$, by $\gamma(I) = \{i \mid i \in I\}$, and (ii) the abstraction function $\alpha : 2^{\mathbb{Z}} \rightarrow \mathcal{I}$ is defined, for each $Z \in 2^{\mathbb{Z}}$, by:

$$\alpha(Z) = \begin{cases} \perp & \text{if } Z = \emptyset \\ [\min(Z), \max(Z)] & \text{otherwise} \end{cases}$$

Widening Operator. When the abstract lattice does not satisfy the ascending chain condition, the computation of $\text{lfp}^{\square^\#}(f^\#)$ may not terminate. To ensure the feasibility of this fixpoint computation, we can use a *widening operator* ∇ , which tries to *guess* the limit of an ascending sequence of elements of a lattice in a finite number of steps [CC77]. This concept is formally defined as follows:

Definition 1.17 (Widening Operator)

Let $\langle \Lambda, \sqsubseteq^\#, \sqcup, \sqcap, \top, \perp \rangle$ be a lattice. $\nabla : \Lambda \times \Lambda \rightarrow \Lambda$ is a *widening operator* if:

- $\forall x, y \in \Lambda : (x \sqsubseteq^\# x \nabla y) \wedge (y \sqsubseteq^\# x \nabla y)$ (or equivalently $x \sqcup y \sqsubseteq^\# x \nabla y$).

- for each ascending sequence $x_0 \sqsubseteq^\# x_1 \sqsubseteq^\# \dots \sqsubseteq^\# x_n \sqsubseteq^\# \dots$, the ascending sequence $(y_i)_{i \in \mathbb{N}_0}$, defined as follows, stabilizes after a finite number of steps:

$$y_i \stackrel{\text{def}}{=} \begin{cases} x_0 & \text{if } i = 0 \\ y_{i-1} \nabla x_i & \text{if } i > 0 \end{cases}$$

The first condition ensures that the widening operator is an overapproximation of the least upper bound and the second one ensures the termination of the computation of the fixpoints.

Example 1.4

A classical widening operator ∇ for the lattice of intervals $\langle \mathcal{I}, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$ is defined, for each $[\ell_1, r_1], [\ell_2, r_2] \in \mathcal{I}$, by $[\ell_1, r_1] \nabla [\ell_2, r_2] \stackrel{\text{def}}{=} [\ell, r]$, where:

$$\ell = \begin{cases} \ell_1 & \text{if } \ell_2 \geq \ell_1 \\ -\infty & \text{otherwise} \end{cases} \quad r = \begin{cases} r_1 & \text{if } r_1 \geq r_2 \\ \infty & \text{otherwise} \end{cases}$$

We can iteratively compute an overapproximation of the least fixpoint of a function by using the widening operator ∇ as follows:

Theorem 1.3 ([CC77])

Let $f^\# : \Lambda \rightarrow \Lambda$ be a function over the *complete* abstract lattice $\langle \Lambda, \sqsubseteq^\#, \sqcup, \sqcap, \top, \perp \rangle$. The sequence $(u_i)_{i \in \mathbb{N}_0}$ defined by:

$$u_i \stackrel{\text{def}}{=} \begin{cases} \perp & \text{if } i = 0 \\ u_{i-1} & \text{if } (i > 0) \wedge (f^\#(u_{i-1}) \sqsubseteq^\# u_{i-1}) \\ u_{i-1} \nabla f^\#(u_{i-1}) & \text{otherwise} \end{cases}$$

stabilizes after a finite number of steps and its limit u_∞ is such that $\text{lfp}^{\sqsubseteq^\#}(f^\#) \sqsubseteq^\# u_\infty$.

Example 1.5

Let us consider the following part of a C++ program where x is an integer variable and C_i ($\forall i \in [0, 3]$) is a control point which represents the possible values of the variable x after the execution of the instruction which follows this control point. We want to compute the values of C_0 , C_1 , C_2 and C_3 :

```

C0: int x;
C1: x = 0;
C2: while(x >= 0)
    {
C3:   x = x + 1;
    }

```

The semantics of this program is defined by the following equation which gives the value of each control point:

$$\begin{cases} C0 = \top \\ C1 = 0 \\ C2 = (C1 \cup C3) \cap [0, \infty[\\ C3 = \text{incr}(C2) \end{cases}$$

where \top represents the entire state space and *incr* is a function which increments its parameter by one unit. To obtain the values of C0, C1, C2 and C3, we define the function $f(C0, C1, C2, C3)$ by $\langle \top, 0, (C1 \cup C3) \cap [0, \infty[, \text{incr}(C2) \rangle$ and we compute its least fixpoint. This computation is performed in the lattice of *intervals* $\langle \mathcal{I}, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$ by using Theorem 1.3. We then obtain the following sequence of values: (i) $u_0 = \langle \perp, \perp, \perp, \perp \rangle$, (ii) $u_1 = \langle [\top, [0, 0], \perp, \perp \rangle$, (iii) $u_2 = \langle [\top, [0, 0], [0, 0], \perp \rangle$, (iv) $u_3 = \langle [\top, [0, 0], [0, 0], [1, 1] \rangle$, (v) $u_4 = \langle [\top, [0, 0], [0, 0] \nabla [0, 1], [1, 1] \rangle = \langle [\top, [0, 0], [0, \infty], [1, 1] \rangle$, (vi) $u_5 = \langle [\top, [0, 0], [0, \infty], [1, \infty] \rangle$. Theorem 1.3 ensures that u_5 is a post-fixpoint of f and we remark that it is also the least fixpoint of this function.

1.4.2 The Representation Framework

The Galois connection framework supposes that the abstract and concrete lattices are *complete*. However, many interesting abstract lattices, like the lattice of *convex polyhedra* or the lattice of *regular languages* (that we will use in this thesis) are not complete and cannot thus be used as abstract lattice in the Galois connection framework. To overcome this obstacle, an alternative framework, called the *representation framework*, has been developed in [Bou92]. This framework proceeds quite similarly to the Galois connection framework to formalize the abstractions and to compute a safe overapproximation of the least fixpoint of a function. The main difference concerns the assumptions imposed on the ele-

ments used to formalize the abstractions, which are *weaker* in the representation framework. In particular, the abstract lattice must only be a partially ordered set.

More precisely, the concrete lattice and the abstract lattice are linked by a *representation* [Bou92], which ensures the computation of an overapproximation of the least fixpoint of a function in a finite number of steps:

Definition 1.18 (Representation [Bou92])

Let $\langle X, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$ be a complete lattice, $\langle \langle \Lambda, \sqsubseteq^\#, \sqcup^\#, \sqcap^\#, \top^\#, \perp^\# \rangle, \gamma, \nabla \rangle$ is a *representation* of X if:

- $\langle \Lambda, \sqsubseteq^\#, \sqcup^\#, \sqcap^\#, \top^\#, \perp^\# \rangle$ is a *partially ordered set*.
- $\gamma : \Lambda \rightarrow X$ is a *monotonic* concretization function, which associates a concrete element with each abstract element. This function thus gives a *meaning* to the abstract elements.
- ∇ is a widening operator, which satisfies the following constraint: $\forall l_1, l_2 \in \Lambda : \gamma(l_2) \sqsubseteq^\# \gamma(l_1 \nabla l_2)$. This widening operator ensures that the computation of an overapproximation of the least fixpoint of a function always terminates.

To correctly perform the computations in the abstract lattice, there must exist an abstraction function $\alpha : X \rightarrow \Lambda$ which *safely* represents each element $x \in X$ by an element $\alpha(x) \in \Lambda$ i.e., $\gamma(\alpha(x)) \sqsupseteq x$. Thus, α can be seen as a function, which *approximates* the elements of the concrete lattice.

In a sense, the concept of representation replaces the concepts of Galois connection and of widening by imposing weaker conditions on the elements used to formalize the abstractions.

We can iteratively compute an overapproximation of the least fixpoint of a function by using the concept of representation as follows:

Theorem 1.4 ([Bou92])

Let $\langle X, \sqsubseteq \rangle$ be a *complete* concrete lattice, $\langle \langle \Lambda, \sqsubseteq^\#, \perp \rangle, \gamma, \nabla \rangle$ be a representation of X , and $f : X \rightarrow X$, $f^\# : \Lambda \rightarrow \Lambda$ be *monotonic* functions such that $\gamma \circ f^\# \sqsupseteq f \circ \gamma$, the sequence $(a_i)_{i \in \mathbb{N}_0}$ (where $\forall i \in \mathbb{N}_0 : a_i \in \Lambda$) defined by:

$$a_i \stackrel{\text{def}}{=} \begin{cases} \perp & \text{if } i = 0 \\ a_{i-1} \nabla f^\sharp(a_{i-1}) & \text{if } i > 0 \end{cases}$$

has a greatest element a_∞ , which is obtained after a finite number of steps and which is a post-fixpoint of f^\sharp . Moreover, $\gamma(a_\infty)$ is a post-fixpoint of f .

An example illustrating this theory is given in section 3.4.2.

Chapter 2

Supervisory Control of Discrete Event Systems

 IN this chapter, we present a state of the art on *supervisory control theory* of *discrete event systems* (*DES* for short) and introduce the main problems, related to this domain, that we will study in this thesis. A discrete event system is a system whose state space is given by a discrete set and whose state transition mechanism is *event-driven* i.e., its state evolution depends only on the occurrence of discrete events over the time. Supervisory control of discrete event systems consists in imposing a given specification on a system by means of a *supervisor* (also called *controller*) which runs in parallel with the original system and which restricts its behavior.

The remainder of this chapter is structured as follows. In section 2.1, we present a state of the art on *supervisory control theory* of *finite* discrete event systems. The control of infinite state systems also received attention in the literature and we present, in section 2.2, some results on the control of *Petri nets*, *timed automata*, and *systems with variables*. In this chapter, we only focus on supervisory control theory, except for the works on the control of timed automata (see section 2.2.2), which are generally based on *game theory*.

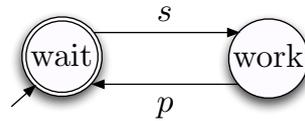


Figure 2.1 - Automaton modeling the behavior of a discrete event system composed of a machine producing parts.

2.1 Supervisory Control Theory of Finite Discrete Event Systems

Supervisory control theory [WR87, RW89, Won05] provides methods to automatically synthesize supervisors that restrict the behavior of a discrete event system in order to fulfill a given specification. In this section, we outline the basic concepts of this theory and present fundamental results. Most of the works on supervisory control theory of finite discrete event systems use the model of automata to formalize the system to be controlled and, in the remainder of this section, we always suppose to work with a finite automaton $\mathcal{T} = \langle X, X_0, X_m, \Sigma, \Delta \rangle$ (see Definition 1.1).

2.1.1 Modeling of Discrete Event Systems

A first task in studying discrete event systems is to develop appropriate models, which adequately describe the behavior of these systems. Ramadge and Wonham [WR87, RW89] proposed to model a discrete event system by two languages L and L_m over an alphabet Σ , where the actions of Σ represent the events of the system and the words of L and L_m represent sequences of events (and thus behaviors) of the system. The language L is prefix-closed and represents all possible behaviors of the system, whereas the language $L_m \subseteq L$ represents all behaviors of the system that correspond to the completion of a task (L_m is generally not prefix-closed). In general, one focuses on regular languages to have effective algorithms. To manipulate more easily these two languages, we can then represent them by a finite automaton \mathcal{T} such that $\mathcal{L}(\mathcal{T}) = L$ and $\mathcal{L}_m(\mathcal{T}) = L_m$. With this approach, the language $\mathcal{L}(\mathcal{T})$ generated by \mathcal{T} represents the possible behaviors of the system and the marked language $\mathcal{L}_m(\mathcal{T})$ of \mathcal{T} models the behaviors of \mathcal{T} which correspond to the completion of a task.

Example 2.1

Let us consider a discrete event system composed of a machine producing parts. This machine can start to work in order to produce a part (action s) and can make a pause when the production of this part is finished (action p). This system can be modeled by the language $L = \{\epsilon, s, sp, sps, spsp, spsps, \dots\}$ describing all possible behaviors of the system and by the language $L_m = \{\epsilon, sp, spsp, \dots\}$ describing the completion of tasks. The automaton \mathcal{T} representing these two languages is depicted in Figure 2.1, where *wait* is the only initial and marked state.

In supervisory control theory, there are mainly two approaches to control a system: the *event-based approach* and the *state-based approach*. Roughly, in the event-based approach, the controller observes the actions fired by the system, whereas, in the state-based approach, it observes the states of the system. These two methods are detailed below.

2.1.2 Event-based Approach

The *event-based* approach [WR87, CL08] has been widely studied in the literature; we present some important results and refer the reader to [CL08] for further details.

Supervisory Control under Full Observation. The discrete event system to be controlled is modeled by a finite automaton \mathcal{T} ; its behavior may violate some *control objectives* modeled by a language $K \subseteq \mathcal{L}_m(\mathcal{T})$ (K is generally not prefix-closed). The aim is to restrict the behavior of this system to satisfy these objectives. For that, a particular device, called *supervisor* (or *controller*), is designed to interact with the system. This supervisor \mathcal{S} can be seen as a function which, from the sequence of actions executed by the system, defines a set of actions that must be forbidden to prevent the system from violating the control objectives. However, the supervisor is not able to control some actions. Indeed, the set of actions Σ is partitioned into the subset of *controllable* actions Σ_c and the subset of *uncontrollable* actions Σ_{uc} . The uncontrollable actions, which model for example failures, data from sensors or clock ticks, cannot be forbidden by the supervisor.

More precisely, a supervisor is a function $\mathcal{S} : \mathcal{L}(\mathcal{T}) \rightarrow 2^{\Sigma_c}$ which interacts with the system \mathcal{T} in a feedback manner as illustrated in Figure 2.2: after a

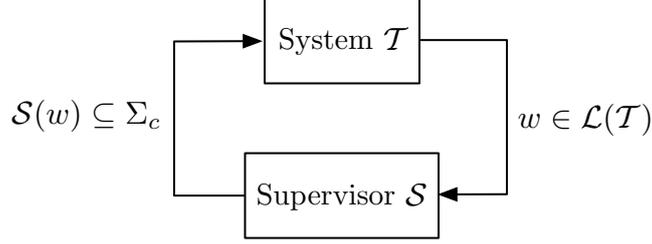


Figure 2.2 - Feedback interaction between the system \mathcal{T} to be controlled and the supervisor \mathcal{S} in an event-based approach under full observation.

sequence w of actions executed by the system, the supervisor computes a set $\mathcal{S}(w)$ of controllable actions that the system must not execute. The *closed loop* system \mathcal{T}/\mathcal{S} resulting from this interaction is called the *controlled system* and must satisfy the control objectives. The *generated language* $\mathcal{L}(\mathcal{T}/\mathcal{S})$ of this system is recursively defined as the smallest set which satisfies the following conditions:

- $\epsilon \in \mathcal{L}(\mathcal{T}/\mathcal{S})$
- $\forall w \in \mathcal{L}(\mathcal{T}), \forall \sigma \in \Sigma : [(w \in \mathcal{L}(\mathcal{T}/\mathcal{S})) \wedge (w\sigma \in \mathcal{L}(\mathcal{T})) \wedge (\sigma \notin \mathcal{S}(w))] \Leftrightarrow (w\sigma \in \mathcal{L}(\mathcal{T}/\mathcal{S}))$

In other words, a sequence $w\sigma$ is a behavior of the controlled system if and only if (i) w is a behavior of the controlled system, (ii) $w\sigma$ is a possible behavior of the system \mathcal{T} , and (iii) the supervisor \mathcal{S} allows the system to fire σ after the sequence w .

The *marked language* $\mathcal{L}_m(\mathcal{T}/\mathcal{S})$ of the controlled system is defined by $\mathcal{L}_m(\mathcal{T}/\mathcal{S}) \stackrel{\text{def}}{=} \mathcal{L}(\mathcal{T}/\mathcal{S}) \cap \mathcal{L}_m(\mathcal{T})$. A supervisor \mathcal{S} is *non-blocking* if its corresponding controlled system \mathcal{T}/\mathcal{S} is non-blocking i.e., $\overline{\mathcal{L}_m(\mathcal{T}/\mathcal{S})} = \mathcal{L}(\mathcal{T}/\mathcal{S})$.

A fundamental result for the supervisory control in the presence of uncontrollable actions is given by the following proposition:

Proposition 2.1 ([WR87])

Let \mathcal{T} be the system to be controlled, and $K \subseteq \mathcal{L}_m(\mathcal{T})$ (where $K \neq \emptyset$) be the control objective, there exists a *non-blocking* supervisor \mathcal{S} such that (i) $\mathcal{L}_m(\mathcal{T}/\mathcal{S}) = K$ and (ii) $\mathcal{L}(\mathcal{T}/\mathcal{S}) = \overline{K}$ if and only if:

- K is $\mathcal{L}_m(\mathcal{T})$ -closed i.e., $K = \overline{K} \cap \mathcal{L}_m(\mathcal{T})$, and
- K is *controllable* w.r.t. Σ_{uc} and $\mathcal{L}(\mathcal{T})$ i.e., $\overline{K}\Sigma_{uc} \cap \mathcal{L}(\mathcal{T}) \subseteq \overline{K}$.

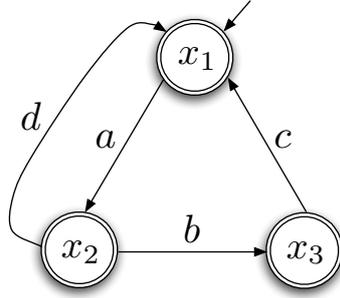


Figure 2.3 - Illustration of the controllability condition.

Moreover, if the controllability and $\mathcal{L}_m(\mathcal{T})$ -closure conditions are satisfied, a supervisor \mathcal{S} , which satisfies the specification, is defined as follows: for each $w \in \mathcal{L}(\mathcal{T})$, $\mathcal{S}(w) \stackrel{\text{def}}{=} \{\sigma \in \Sigma_c \mid w\sigma \notin \overline{K}\}$.

Thus, a controlled system, which exactly achieves the specification, can be computed if and only if the controllability and $\mathcal{L}_m(\mathcal{T})$ -closure conditions hold. The first property means that if a behavior of \overline{K} is extended with an uncontrollable action and gives a behavior of $\mathcal{L}(\mathcal{T})$, then this behavior must be included in \overline{K} . Consequently, firing an uncontrollable action always gives a *good* behavior. The second condition ensures that the controlled system satisfies the non-blocking property. The time complexity to test the controllability and $\mathcal{L}_m(\mathcal{T})$ -closure properties is polynomial [WR87, RW89]. The following example illustrates the concept of controllability:

Example 2.2

The system to be controlled is defined by the automaton \mathcal{T} depicted in Figure 2.3 where (i) $X = \{x_1, x_2, x_3\}$, (ii) $X_0 = \{x_1\}$, (iii) $X_m = X$, and (iv) $\Sigma = \{a, b, c, d\}$ with $\Sigma_c = \{a, c\}$ and $\Sigma_{uc} = \{b, d\}$. The language $K_1 = \{a, ab, abca\}$ is not controllable, whereas $K_2 = \{a, ab, ad\}$ is controllable. The language K_1 is not controllable, because the uncontrollable action d , which can occur after a , gives a sequence ad which is not in K_1 .

When the non-blocking property is not under consideration, a result similar to Proposition 2.1 can be obtained by ensuring only the controllability condition i.e., there exists a supervisor \mathcal{S} such that $\mathcal{L}(\mathcal{T}/\mathcal{S}) = \overline{K}$ (where $K \neq \emptyset \subseteq \mathcal{L}(\mathcal{T})$) if and only if K is *controllable* w.r.t. Σ_{uc} and $\mathcal{L}(\mathcal{T})$ [WR87].

When there exists no supervisor, which can restrict the behavior of \mathcal{T} to K ,

we would like to compute a supervisor such that the resulting controlled system achieves the *largest* sublanguage of K .

Problem 2.1 ([WR87])

Let \mathcal{T} be the system to be controlled, and $K \subseteq \mathcal{L}_m(\mathcal{T})$ (where K is $\mathcal{L}_m(\mathcal{T})$ -closed) be the control objective, does there exist a *non-blocking* supervisor \mathcal{S} such that (i) $\mathcal{L}_m(\mathcal{T}/\mathcal{S}) \subseteq K$ and (ii) \mathcal{S} is optimal i.e., for each supervisor \mathcal{S}' such that $\mathcal{L}_m(\mathcal{T}/\mathcal{S}') \subseteq K$, we have that $\mathcal{L}_m(\mathcal{T}/\mathcal{S}') \subseteq \mathcal{L}_m(\mathcal{T}/\mathcal{S})$?

This problem can be solved thanks to the concept of *supremal controllable sublanguage*. The supremal controllable sublanguage of K is denoted by $K^{\uparrow C}$ and is defined by $K^{\uparrow C} \stackrel{\text{def}}{=} \bigcup_{L \in C_{in}(K)} L$, where $C_{in}(K) \stackrel{\text{def}}{=} \{L \subseteq K \mid \overline{L}\Sigma_{uc} \cap \mathcal{L}(\mathcal{T}) \subseteq \overline{L}\}$ gives the set of controllable sublanguages of K . The element $K^{\uparrow C}$ always belongs to $C_{in}(K)$ (one can note that $K^{\uparrow C} = \emptyset$, when $C_{in}(K) = \emptyset$), because the class $C_{in}(K)$ of controllable sublanguages of K is closed under union i.e., if $K_1, K_2 \in C_{in}(K)$, then $K_1 \cup K_2 \in C_{in}(K)$ [WR87, RW89]. When $K^{\uparrow C} = \emptyset$, there exists no supervisor \mathcal{S} such that $\mathcal{L}_m(\mathcal{T}/\mathcal{S}) \subseteq K$. Otherwise ($K^{\uparrow C} \neq \emptyset$), there exists a solution to Problem 2.1 and it is given by the supervisor \mathcal{S} which satisfies the property $\mathcal{L}(\mathcal{T}/\mathcal{S}) = \overline{K^{\uparrow C}}$ and $\mathcal{L}_m(\mathcal{T}/\mathcal{S}) = K^{\uparrow C}$. This supervisor can be obtained by Proposition 2.1, because $K^{\uparrow C}$ is $\mathcal{L}_m(\mathcal{T})$ -closed and controllable w.r.t. Σ_{uc} and $\mathcal{L}(\mathcal{T})$ [WR87, RW89, CL08].

When the non-blocking property is not under consideration, an optimal supervisor \mathcal{S} ensuring the specification $\mathcal{L}(\mathcal{T}/\mathcal{S}) \subseteq K$ (where $K = \overline{K} \subseteq \mathcal{L}(\mathcal{T})$) can also be computed thanks to the concept of supremal controllable sublanguage. Note that, if $K^{\uparrow C} \neq \emptyset$, the supervisor \mathcal{S} , which satisfies the property $\mathcal{L}(\mathcal{T}/\mathcal{S}) = K^{\uparrow C}$, is actually a solution to this problem [WR87].

Supervisory Control under Partial Observation. In practice, the supervisor interacts with the plant through *sensors* and *actuators* and it has only a *partial observation* of the system, because these materials have not an absolute precision or some parts of the plant might not be observed by the supervisor. In the event-based approach, this *partial observation* means that the supervisor does not observe some actions. This partitions the set of actions Σ into the set Σ_o of *observable actions* (the actions that the supervisor observes) and the set Σ_{uo} of *unobservable actions* (the actions that the supervisor does not observe). This partial observation is formally defined by a projection $P : \Sigma^* \rightarrow \Sigma_o^*$ which maps a sequence $w \in \Sigma^*$ of actions into the sequence $P(w) \in \Sigma_o^*$ of observable actions. The feedback interaction between the system to be controlled and

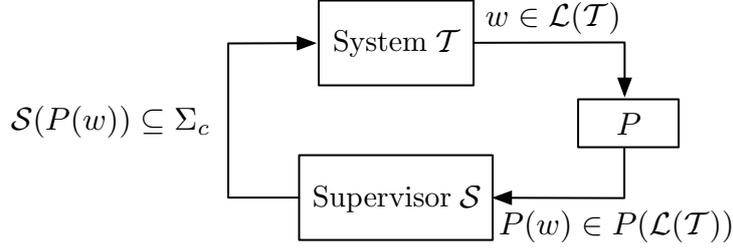


Figure 2.4 - Feedback interaction between the system \mathcal{T} to be controlled and the supervisor \mathcal{S} in an event-based approach under partial observation.

the supervisor with partial observation is depicted in Figure 2.4 and is defined as follows: after a sequence w of actions executed by the system, the supervisor computes a set $\mathcal{S}(P(w))$ of controllable actions that the system must not execute (see Figure 2.4).

Proposition 2.1, which allows us to synthesize a supervisor such that the resulting controlled system exactly achieves a specification K , can be extended to the framework with partial observation by adding an additional condition called *observability* [LW88, CL08]:

Proposition 2.2 ([LW88])

Let \mathcal{T} be the system to be controlled, $K \subseteq \mathcal{L}_m(\mathcal{T})$ (where $K \neq \emptyset$) be the control objective, and $P : \Sigma^* \rightarrow \Sigma_o^*$ be a projection, there exists a non-blocking supervisor \mathcal{S} with partial observation such that (i) $\mathcal{L}(\mathcal{T}/\mathcal{S}) = \overline{K}$ and (ii) $\mathcal{L}_m(\mathcal{T}/\mathcal{S}) = K$ if and only if:

- K is *controllable* w.r.t. Σ_{uc} and $\mathcal{L}(\mathcal{T})$,
- K is $\mathcal{L}_m(\mathcal{T})$ -closed, and
- K is *observable* w.r.t. Σ_c , Σ_o and $\mathcal{L}(\mathcal{T})$ i.e., $\forall w \in \overline{K}, \forall \sigma \in \Sigma_c : [(w\sigma \in \mathcal{L}(\mathcal{T}) \setminus \overline{K}) \Rightarrow [\{P^{-1}(P(w)).\sigma\} \cap \overline{K} = \emptyset]]$.

Moreover, if all conditions are satisfied, a supervisor \mathcal{S} , which satisfies the specification, is defined as follows: for each $w \in P(\mathcal{L}(\mathcal{T}))$, $\mathcal{S}(w) = \{\sigma \in \Sigma_c \mid \nexists w'\sigma \in \overline{K} : P(w') = w\}$.

Intuitively, the observability condition means that if an action σ must be disabled after a sequence w , then this action must also be disabled after each sequence which is undistinguishable from w . This property thus ensures that the supervi-

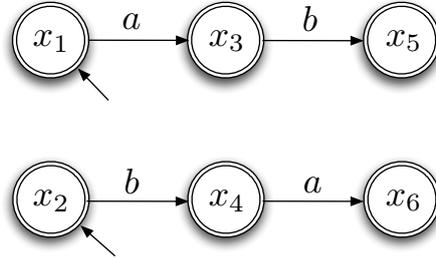


Figure 2.5 - System without an optimal supervisor.

sor is coherent w.r.t. the actions that it forbids, since it inhibits the same set of actions after the sequences of actions that it does not distinguish. Therefore, a correct supervisor must satisfy the observability condition or at least a stronger property to define a coherent control policy w.r.t. its partial observation. The time complexity to test the observability property is polynomial [Tsi89].

A similar result also exists for the case where the non-blocking property must not be ensured [LW88].

When the conditions of Proposition 2.2 do not hold, the computation of an optimal supervisor \mathcal{S} such that $\mathcal{L}_m(\mathcal{T}/\mathcal{S}) \subseteq K$ is not always possible, because the class of observable sublanguages of K is not closed under union and hence there does not always exist a supremal observable and controllable sublanguage of K [LW88]. The following example illustrates this property:

Example 2.3

In the system \mathcal{T} depicted in Figure 2.5, (i) the set of states $X = \{x_i \mid i \in [1, 6]\}$, (ii) the set of initial states $X_0 = \{x_1, x_2\}$, (iii) the set of final states $X_m = X$, and (iv) all transitions are controllable and unobservable. The specification K is equal to $\{\epsilon, a, b\}$. There are three supervisors which satisfy this specification:

- the supervisor \mathcal{S}_1 which always forbids the actions a and b : $\mathcal{L}(\mathcal{T}/\mathcal{S}_1) = \{\epsilon\}$.
- the supervisor \mathcal{S}_2 which always forbids the action a : $\mathcal{L}(\mathcal{T}/\mathcal{S}_2) = \{\epsilon, b\}$.
- the supervisor \mathcal{S}_3 which always forbids the action b : $\mathcal{L}(\mathcal{T}/\mathcal{S}_3) = \{\epsilon, a\}$.

Therefore, there is no optimal solution, because the supervisors \mathcal{S}_2 and \mathcal{S}_3 are incomparable.

Various approaches have then been developed to overcome this problem and we

mention two of them:

- *Considering a weaker problem by computing a maximal supervisor.* A supervisor \mathcal{S} is *maximal* if, for each supervisor \mathcal{S}' such that $\mathcal{L}_m(\mathcal{T}_{/\mathcal{S}'}) \subseteq K$, we have that $\mathcal{L}_m(\mathcal{T}_{/\mathcal{S}'}) \not\subseteq \mathcal{L}_m(\mathcal{T}_{/\mathcal{S}})$. In [RP05], Riedweg and Pinchinat work on this problem. They prove that it is decidable and they propose an algorithm solving it with a $2\text{EXP-TIME}(|\psi| \times |P|)$ complexity (where $|\psi|$ is the size of the specification and $|P|$ is the size of the system).
- *Considering some situations where the supremal observable and controllable sublanguage of K exists.* That is, for example, the case when $\Sigma_c \subseteq \Sigma_o$. Indeed, under this assumption, the controllability and observability properties imply the normality property [KGM91], which is formally defined as follows:

Definition 2.1 (Normality)

Let L (with $L = \bar{L}$) and K (with $K \subseteq L$) be two languages over Σ and $P : \Sigma^* \rightarrow \Sigma_o^*$ be a projection, K is *normal* w.r.t. P and L if $\bar{K} = P^{-1}(P(\bar{K})) \cap L$.

In other words, each behavior of \bar{K} can be recovered from L and from the behaviors $P^{-1}(P(\bar{K}))$ that are undistinguishable from \bar{K} . This property is interesting, because the class of normal sublanguages of a language K is closed under union and hence there always exists a supremal controllable and normal sublanguage (denoted by $K^{\uparrow(CN)}$) of K . Consequently, when $\Sigma_c \subseteq \Sigma_o$, an optimal supervisor \mathcal{S} such that $\mathcal{L}_m(\mathcal{T}_{/\mathcal{S}}) \subseteq K$ always exists, since, in this case, the controllability and observability properties imply the normality property. It is obtained by computing a supervisor such that the resulting controlled system achieves exactly $K^{\uparrow(CN)}$ [KGM91].

Decentralized Supervisory Control. In the previous approaches, the system is controlled by a single supervisor, but this framework is not suitable for the control of some systems like the distributed systems. Indeed, these systems are generally composed of several subsystems acting in parallel and located in different sites. It is thus more appropriate to have several supervisors that control the system to ensure the desired properties. The decentralized framework [RW92b, YL02, CL08] follows this approach. It has been widely studied in the past years and has shown its usefulness in several domains like manufacturing systems, communication network protocols, . . . [RW92a]. In this approach, the system is controlled by n local supervisors. Each supervisor has its own

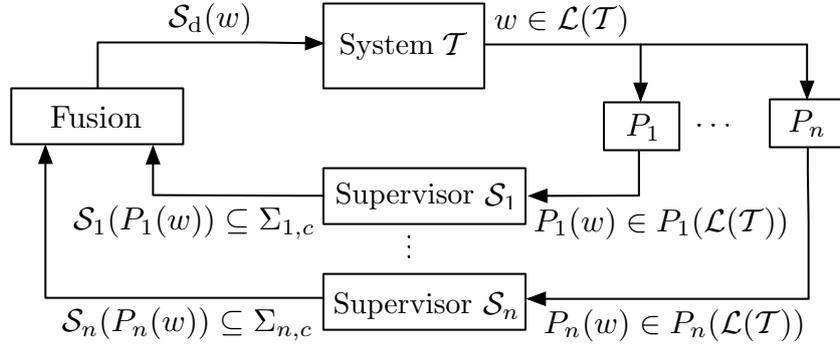


Figure 2.6 - Feedback interaction between the system \mathcal{T} to be controlled and the n local supervisors \mathcal{S}_i ($\forall i \in [1, n]$) in a decentralized event-based approach.

view of the system and its own set of controllable actions. Indeed, due to the distributed nature of the system to be controlled, the supervisors observe the system through different sets of sensors (these sets can overlap) and control it by means of different sets of actuators (these sets can also overlap). Moreover, there is a *coordination mechanism* (called *fusion rule*) defining the global control to be applied to the system. This global control is computed from the control decisions of each local supervisor [CL08]. In the decentralized framework, we have no information regarding the structure of the distributed system to be controlled while the computation of the supervisors. Therefore, the computation of the local supervisors must be performed on the model of the global system resulting from the parallel composition of each subsystem.

More precisely, each local supervisor \mathcal{S}_i ($\forall i \in [1, n]$) can control the set of actions $\Sigma_{i,c}$ and cannot control the set of actions $\Sigma_{i,uc}$. The subsets $\Sigma_{1,c}, \dots, \Sigma_{n,c}$ are not necessarily disjoint. The set of actions that can be controlled by at least one controller is denoted by Σ_c and is defined by $\Sigma_c \stackrel{\text{def}}{=} \bigcup_{i=1}^n \Sigma_{i,c}$ and the set of actions that cannot be controlled is denoted by Σ_{uc} and is defined by $\Sigma_{uc} \stackrel{\text{def}}{=} \Sigma \setminus \Sigma_c$. Moreover, each local supervisor \mathcal{S}_i can observe the set of actions $\Sigma_{i,o}$ and cannot observe the set of actions $\Sigma_{i,uo}$ (the subsets $\Sigma_{1,o}, \dots, \Sigma_{n,o}$ are not necessarily disjoint). The partial observation of the supervisor \mathcal{S}_i ($\forall i \in [1, n]$) is modeled by the projection $P_i : \Sigma^* \rightarrow \Sigma_{i,o}^*$. The feedback interaction between the system and the local supervisors is depicted in Figure 2.6 and works as follows. After a sequence w of actions executed by the system, each supervisor \mathcal{S}_i computes its own set $\mathcal{S}_i(P_i(w))$ of actions that it proposes to forbid. Next, the

global control applied to the system is the fusion, according to some rule to be defined, of the control decisions of each supervisor. There exist different fusion rules in the literature and we mention two of them:

- *the conjunctive architecture*: the fusion rule is defined by the *union of forbidden actions* and is given by the function $\mathcal{R}_c : \mathcal{L}(\mathcal{T}) \rightarrow 2^{\Sigma_c}$, which is defined, for each $w \in \mathcal{L}(\mathcal{T})$, by:

$$\mathcal{R}_c(w) \stackrel{\text{def}}{=} \bigcup_{i=1}^n \mathcal{S}_i(P_i(w)) \quad (2.1)$$

This architecture is called conjunctive, because it is defined by the intersection of enabled actions¹.

- *the disjunctive architecture*: the fusion rule is defined by the *intersection of forbidden actions* and is given by the function $\mathcal{R}_d : \mathcal{L}(\mathcal{T}) \rightarrow 2^{\Sigma_c}$, which is defined, for each $w \in \mathcal{L}(\mathcal{T})$, by:

$$\mathcal{R}_d(w) \stackrel{\text{def}}{=} \bigcap_{i=1}^n \mathcal{S}_i(P_i(w)) \quad (2.2)$$

This architecture is called disjunctive, because it is defined by the union of enabled actions.

A *decentralized supervisor* \mathcal{S}_d is then defined by a pair $\langle (\mathcal{S}_i)_{i=1}^n, \mathcal{R} \rangle$ i.e., by n local supervisors \mathcal{S}_i coordinated by a fusion rule \mathcal{R} . The controlled system, that it defines, is denoted by $\mathcal{T}_{/\mathcal{S}_d}$.

A fundamental result for the decentralized control (with a conjunctive architecture) is given by the following proposition:

Proposition 2.3 ([RW92b, RDFV88])

Let \mathcal{T} be the system to be controlled, $K \subseteq \mathcal{L}_m(\mathcal{T})$ (where $K \neq \emptyset$) be the control objective, $\Sigma_{i,c}$ and $\Sigma_{i,uc}$ ($\forall i \in [1, n]$) be a partition of Σ , $\Sigma_c = \bigcup_{i=1}^n \Sigma_{i,c}$, $\Sigma_{uc} = \Sigma \setminus \Sigma_c$, $\Sigma_{i,o}$ and $\Sigma_{i,uo}$ ($\forall i \in [1, n]$) be a partition of Σ , $\Sigma_o = \bigcup_{i=1}^n \Sigma_{i,o}$, $\Sigma_{uo} = \Sigma \setminus \Sigma_o$, and $P_i : \Sigma^* \rightarrow \Sigma_{i,o}^*$ ($\forall i \in [1, n]$) be a projection, there exists a non-blocking decentralized supervisor $\mathcal{S}_d = \langle (\mathcal{S}_i)_{i=1}^n, \mathcal{R}_c \rangle$ such that (i) $\mathcal{L}(\mathcal{T}_{/\mathcal{S}_d}) = \overline{K}$, and (ii) $\mathcal{L}_m(\mathcal{T}_{/\mathcal{S}_d}) = K$ if and only if:

- K is *controllable* w.r.t. Σ_{uc} and $\mathcal{L}(\mathcal{T})$,

¹In the literature, a supervisor generally gives the actions that are enabled.

- K is $\mathcal{L}_m(\mathcal{T})$ -closed, and
- K is *CP-coobservable* w.r.t. $\Sigma_{i,c}, \Sigma_{i,o}$ ($\forall i \in [1, n]$) and $\mathcal{L}(\mathcal{T})$ i.e., $\forall w \in \overline{K}, \forall \sigma \in \Sigma_c : [w\sigma \in \mathcal{L}(\mathcal{T}) \setminus \overline{K}] \Rightarrow [\exists i \in [1, n] : (\sigma \in \Sigma_{i,c}) \wedge (\{P_i^{-1}(P_i(w)).\sigma\} \cap \overline{K} = \emptyset)]$.

Thus, a decentralized supervisor such that the resulting controlled system exactly achieves the specification, can be computed if and only if the controllability, $\mathcal{L}_m(\mathcal{T})$ -closure and CP-coobservability conditions hold. This last condition means that if an action σ must be disabled after a sequence w , then at least one of the supervisors, that control σ , proposes to forbid this action after each sequence that it does not distinguish from w . The time complexity to test the CP-coobservability property is polynomial [RW95].

A similar result exists for the disjunctive architecture [YL02].

One can note that the centralized control under partial observation is a particular case of the decentralized control. It corresponds to the case where the number n of local supervisors is equal to 1.

When a specification K is not CP-coobservable, we would like to compute a non-blocking and correct supervisor such that the resulting controlled system achieves a subset of K . But, this is, in general, not possible:

Proposition 2.4 ([Thi05])

Let \mathcal{T} be the system to be controlled, $K \subseteq \mathcal{L}_m(\mathcal{T})$ (where K is not CP-coobservable) be the control objective, $\Sigma_{i,c}$ and $\Sigma_{i,uc}$ ($\forall i \in [1, n]$) be a partition of Σ , $\Sigma_c = \bigcup_{i=1}^n \Sigma_{i,c}$, $\Sigma_{uc} = \Sigma \setminus \Sigma_c$, $\Sigma_{i,o}$ and $\Sigma_{i,uo}$ ($\forall i \in [1, n]$) be a partition of Σ , $\Sigma_o = \bigcup_{i=1}^n \Sigma_{i,o}$, $\Sigma_{uo} = \Sigma \setminus \Sigma_o$, and $P_i : \Sigma^* \rightarrow \Sigma_{i,o}^*$ ($\forall i \in [1, n]$) be a projection, the problem, which consists in deciding if there exists a non-blocking decentralized supervisor $\mathcal{S}_d = \langle (\mathcal{S}_i)_{i=1}^n, \mathcal{R}_c \rangle$ such that $\mathcal{L}_m(\mathcal{T}_{/\mathcal{S}_d}) \subseteq K$ is undecidable.

However, when the non-blocking requirement is relaxed, the problem becomes decidable [RW95].

Several approaches have been developed to improve the control policy in the decentralized framework and we mention two of them:

- *Adding communication between the supervisors* [BL98, BL00, XK09, Tri02]. In [BL00], Barrett and Lafortune formalize a decentralized framework with

communication by specifying the communication architecture and the kind of information exchanged. They assume that there are no communication delays and notably give a necessary and sufficient condition for the existence of solutions achieving exactly a control specification (defined by a language). In [XK09], Kumar and Xu propose an algorithm, which computes, for each local supervisor, an estimate of the current state of the system. To compute a *good* state estimate, the local supervisors exchange information: they send, to the other local supervisors, the actions that they receive from the system. In this framework, the communication delays are unbounded. In [Tri02], Tripakis studies the time complexity of some problems related to the decentralized control with communication. The specification is given by a set ϕ of *responsiveness* properties over an alphabet Σ . A responsiveness property is a formula of the form $a \rightsquigarrow b$ (where $a, b \in \Sigma$) and it is satisfied by a (finite or infinite) string ℓ over Σ if an action b eventually occurs after every action a in ℓ . The problem under consideration consists in deciding if there exists a decentralized supervisor \mathcal{S}_d such that the *maximal* language L_{max} of the controlled system $\mathcal{T}_{/\mathcal{S}_d}$ satisfies ϕ . This language L_{max} is defined by all infinite strings of the controlled system and all finite strings that cannot be extended by an action of Σ (i.e., the system is in a deadlocking state). The author proves that the problem is *undecidable* when there is no communication or when the communication delays are unbounded. He conjectures that the problem is *decidable* when the communication delays are bounded.

- *Defining local supervisors with conditional control decision.* Usually, in the decentralized framework, the local supervisors propose unconditional decisions i.e., they propose either to forbid or to allow an action. In more elaborate architectures, they may propose *conditional* decisions i.e., control decisions which depend on the decisions of the other supervisors. For example, a supervisor can propose to allow an action if no supervisor proposes to forbid it. In [YL04], Yoo and Lafortune propose a decentralized framework where there are four control decisions: enable, disable, disable if nobody enables and enable if nobody disables. They notably give a necessary and sufficient condition for the existence of a decentralized supervisor such that the resulting controlled system exactly achieves a desired specification (defined by a language). In [Ric03], Ricker and Rudie propose a decentralized framework with four possible control decisions: enable, disable, conditional disable, do not know. A local supervisor makes its control decision based on its own

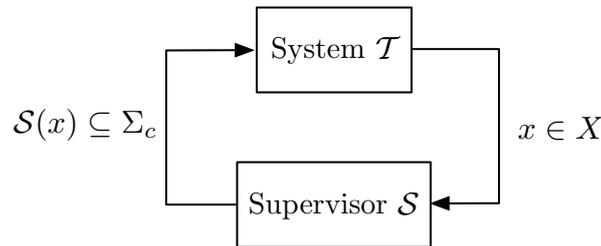


Figure 2.7 - Feedback interaction between the system \mathcal{T} to be controlled and the memoryless supervisor \mathcal{S} in a state-based approach under full observation.

knowledge of the system and if it cannot make a definitive control decision, it reasons about whether other local supervisors have sufficiently knowledge to make the correct control decision. More precisely, if the local supervisor \mathcal{S}_i cannot decide (based on its observation of the system) whether an action must be forbidden or allowed, it tries to guess (based on its *own* observation of the system) whether another local supervisor \mathcal{S}_j could allow this action. If it thinks that a local supervisor could allow it, it proposes the *conditional disable* decision; otherwise, it proposes the *do not know* decision. The authors give a necessary and sufficient condition for the existence of a decentralized supervisor, which prevents the system from firing a set of forbidden actions when they can occur.

2.1.3 State-based Approach

Specifying the control objectives by a language is not always the more appropriate approach. For example, for the *state avoidance control problem* [TK97, KGM93], where the aim is to prevent the system from reaching a set *Bad* of *forbidden states*, it seems more pertinent to specify the control objectives by a predicate giving the forbidden states rather than by a language K giving the set of behaviors which do not lead to the forbidden states (as in section 2.1.2). For this kind of problems, it can be shown that a *memoryless* supervisor (i.e., a supervisor that observes the *states* of the system and that does not need to keep in memory the past of the execution of the system) is sufficient to define the control policy.

Supervisory Control under Full Observation. Under full observation, a

supervisor \mathcal{S} is a function $X \rightarrow 2^{\Sigma_c}$ which interacts with the system \mathcal{T} in a feedback manner as illustrated in Figure 2.7: when the system \mathcal{T} reaches a state x , the supervisor observes that \mathcal{T} is in x and it computes, from this information, a set of controllable actions that the system cannot execute. One can note that the supervisor \mathcal{S} is *memoryless* i.e., its control decisions are only based on the current information received from the system. The controlled system $\mathcal{T}/\mathcal{S} = \langle X, X_0, X_m, \Sigma, \Delta_{/\mathcal{S}} \rangle$ must satisfy the control objectives and its transition relation $\Delta_{/\mathcal{S}}$ is defined as follows: $\Delta_{/\mathcal{S}} \stackrel{\text{def}}{=} \{ \langle x_1, \sigma, x_2 \rangle \mid (\langle x_1, \sigma, x_2 \rangle \in \Delta) \wedge (\sigma \notin \mathcal{S}(x_1)) \}$. Thus, $\Delta_{/\mathcal{S}}$ corresponds to a restriction of Δ .

A well-known result in this framework is given by the following proposition [Ush89]:

Proposition 2.5 ([Ush89])

Let \mathcal{T} be the system to be controlled, and $Bad \subseteq X$ be the set of forbidden states, there exists a memoryless supervisor \mathcal{S} such that²:

- $\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{S}}(X_0) \cap Bad = \emptyset$, and
- \mathcal{S} is optimal i.e., for each supervisor \mathcal{S}' such that $\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{S}'}(X_0) \cap Bad = \emptyset$, we have that $\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{S}'}(X_0) \subseteq \text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{S}}(X_0)$.

if and only if $X_0 \cap \text{Coreach}_{\Sigma_{uc}}^{\mathcal{T}}(Bad) = \emptyset$.

Moreover, if this condition is satisfied, a supervisor \mathcal{S} , which satisfies the specification, is defined as follows: for each $x \in X \setminus \text{Coreach}_{\Sigma_{uc}}^{\mathcal{T}}(Bad)$, $\mathcal{S}(x) \stackrel{\text{def}}{=} \{ \sigma \in \Sigma_c \mid \exists x' \in \text{Coreach}_{\Sigma_{uc}}^{\mathcal{T}}(Bad) : \langle x, \sigma, x' \rangle \in \Delta \}$.

This property means that there exists an optimal supervisor which prevents the system from reaching Bad if and only if a forbidden state cannot be reached from an initial state of the system \mathcal{T} by a sequence of uncontrollable transitions. The condition in Proposition 2.5 can be tested in polynomial time.

An optimal supervisor can also be computed (when a solution exists) when the deadlock free or non-blocking property must be ensured. For that, we first compute the set of states leading uncontrollably to Bad and we add them to the set of forbidden states. Next, if we make unreachable these states, there may be blocking (resp. deadlocking) states in the resulting controlled system. We thus add these blocking (resp. deadlocking) states to the set of forbidden

²The reachability and coreachability operators are respectively defined in Definitions 1.10 and 1.9.

states. But, there can then be new states leading uncontrollably to the set of forbidden states. We thus repeat these two steps until stabilization and we then obtain a set denoted by $I_{bl}(Bad)$ (resp. $I_{df}(Bad)$). If $I_{bl}(Bad) \cap X_0 = \emptyset$ (resp. $I_{df}(Bad) \cap X_0 = \emptyset$), then there exists an optimal supervisor which prevents the system from reaching Bad and which ensures the non-blocking (resp. deadlock free) property. This supervisor is defined as follows: for each $x \in X \setminus I_{bl}(Bad)$, $\mathcal{S}(x) = \{\sigma \in \Sigma_c \mid \exists x' \in I_{bl}(Bad) : \langle x, \sigma, x' \rangle \in \Delta\}$ (resp. for each $x \in X \setminus I_{df}(Bad)$, $\mathcal{S}(x) = \{\sigma \in \Sigma_c \mid \exists x' \in I_{df}(Bad) : \langle x, \sigma, x' \rangle \in \Delta\}$).

Remark 2.1

In [GM05], Gaudin and Marchand consider the problem described in Proposition 2.5 in a *modular* framework where the system to be controlled is given by a collection of n subsystems. They provide an algorithm which computes an optimal supervisor for this problem by performing the computations locally on each subsystem. These local computations allow them to avoid the *combinatorial explosion of the number of states* resulting from the parallel composition of the subsystems.

Let us now explain how the event-based approach and the state-based approach can be linked.

The event-based approach can be used to compute an optimal supervisor which prevents the system \mathcal{T} from reaching a set Bad of forbidden states. For that, we first define the automaton $\mathcal{T}' = \langle X \setminus Bad, X_0 \setminus Bad, X_m \setminus Bad, \Sigma, \Delta \rangle$. Next, we compute a supervisor such that the resulting controlled system achieves the supremal controllable sublanguage of $\mathcal{L}(\mathcal{T}')$ w.r.t. Σ_{uc} and $\mathcal{L}(\mathcal{T}')$, and this supervisor is actually an optimal solution which prevents the system from reaching Bad ³.

Inversely, the state-based approach can be used to compute a supervisor such that the resulting controlled system achieves the supremal controllable sublanguage of a specification K . For that, we compute the *synchronous product* of the automaton \mathcal{T} and of the automaton $\mathcal{T}_k = \langle X_k, (X_0)_k, (X_m)_k, \Sigma, \Delta_k \rangle$, which generates the language K . That gives an automaton $\mathcal{T}' = \langle X \times X_k, X_0 \times (X_0)_k, X_m \times (X_m)_k, \Sigma, \Delta' \rangle$ such that $\mathcal{L}(\mathcal{T}') = \mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\mathcal{T}_k)$. Next, the set Bad of forbidden states is defined by $Bad = \{\langle x, x_k \rangle \in X \times X_k \mid \exists \sigma \in \Sigma_{uc}, \exists x' \in X : [(\langle x, \sigma, x' \rangle \in \Delta) \wedge (\nexists x'_k \in X_k : \langle \langle x, x_k \rangle, \sigma, \langle x', x'_k \rangle \rangle \in \Delta')]\}$. Thus, $\langle x, x_k \rangle$ is considered as a forbidden state if

³The time complexity is then in $O(X^2)$ instead of $O(X)$.

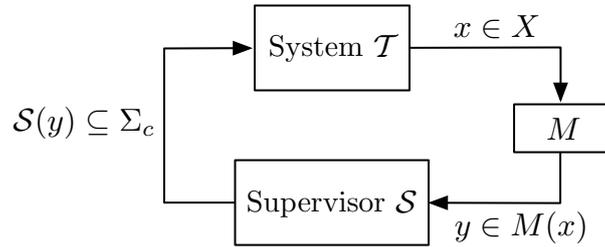


Figure 2.8 - Feedback interaction between the system \mathcal{T} to be controlled and the supervisor \mathcal{S} in a state-based approach under partial observation.

there exists an uncontrollable transition σ that can be fired from x in \mathcal{T} and that cannot be fired from $\langle x, x_k \rangle$ in \mathcal{T}' (i.e., that produces a behavior which does not belong to K). Finally, we use Proposition 2.5 to compute an optimal supervisor, which prevents the system \mathcal{T}' from reaching Bad , and the controlled system defined by this supervisor achieves the supremal controllable sublanguage of K .

Supervisory Control under Partial Observation. In the framework with partial observation, the supervisor cannot distinguish some states⁴. This partial observation is formally defined by a mask $M : X \rightarrow 2^Y$ which corresponds to a mapping from the state space X to the power set of an *observation space* Y and which can also be seen as a *covering* of the state space [HTL08, DDR06]. When the system arrives in a state x , the supervisor receives an observation $y \in M(x)$ as information and it defines its control decisions from this information (see Figure 2.8). In the sequel, we say that a state x is *compatible* with an observation y if $y \in M(x)$.

In [TUK95], Takai *et al.* consider a framework where the system to be controlled is deterministic and the partial observation is modeled by a mask $M : X \rightarrow Y$ corresponding to a *partition* of the state space i.e., each state is compatible with *exactly* one observation (one can note that this definition of M is a particular case of the definition where it is defined as a covering of the state space). They provide a necessary and sufficient condition for the existence of a supervisor that allows the system to exactly reach the set of states Bad^c (where $Bad^c = X \setminus Bad$ and Bad is the set of forbidden states). This necessary and sufficient condition, given in Proposition 2.6, uses the following memoryless

⁴This has to be opposed to the event-based approach where the partial observation is due to some actions that cannot be observed by the supervisor.

supervisor $\tilde{\mathcal{S}}$ with partial observation, which prevents the system from reaching Bad :

$$\forall y \in Y, \tilde{\mathcal{S}}(y) \stackrel{\text{def}}{=} \{ \sigma \in \Sigma_c \mid \exists x_1 \notin Bad, \exists x_2 \in Bad : (y = M(x_1)) \wedge \langle x_1, \sigma, x_2 \rangle \in \Delta \} \quad (2.3)$$

Thus, the supervisor $\tilde{\mathcal{S}}$ forbids an action σ in the observation state y if there exists a state $x_1 \notin Bad$ compatible with y , which leads to a forbidden state by σ .

Proposition 2.6 ([TUK95])

Let $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$ be the *deterministic* system to be controlled, $Bad \subseteq X$ be the set of forbidden states, and $M : X \rightarrow Y$ be a mask, there exists a memoryless supervisor \mathcal{S} such that $\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{S}}(x_0) = Bad^c$ if and only if Bad^c is M -controllable⁵ i.e., the supervisor $\tilde{\mathcal{S}}$, defined in (2.3), satisfies the property $\text{Reachable}_{\Sigma}^{\mathcal{T}/\tilde{\mathcal{S}}}(x_0) = Bad^c$.

Obviously, if the M -controllability condition holds, the supervisor $\tilde{\mathcal{S}}$ is a solution to this problem. This proposition means that if the system resulting from the supervision of $\tilde{\mathcal{S}}$ does not exactly reach Bad^c , then no supervisor can satisfy this requirement. The time complexity to test the M -controllability property is polynomial [TK97, TK98].

When the M -controllability condition does not hold, Takai and Kodama provide an algorithm computing a memoryless supervisor such that the resulting controlled system reaches a subset of Bad^c [TK97, TK98]. This algorithm is based on a *forward approach* and is composed of two parts; in this algorithm, the set Bad^c of good states is denoted by Q :

- the computation of the set of *safe* states $Q^\uparrow \subseteq Q$ where $Q^\uparrow \stackrel{\text{def}}{=} \bigcap_{j=0}^{\infty} Q_j$, with Q_j defined recursively as follows:

$$Q_j \stackrel{\text{def}}{=} \begin{cases} Q & \text{if } j = 0 \\ g(Q_{j-1}) & \text{otherwise} \end{cases}$$

where, for each $Q' \subseteq Q$, the function $g(Q') \stackrel{\text{def}}{=} Q \cap \left(\bigcap_{\sigma \in \Sigma_{uc}} \{ x_1 \in X \mid \forall x_2 \in X : (\langle x_1, \sigma, x_2 \rangle \in \Delta) \Rightarrow (x_2 \in Q') \} \right)$. Thus, Q^\uparrow gives the set of states that do not lead to Bad by a sequence of uncontrollable transitions.

⁵We give a simplified (and equivalent) version of the definition of M -controllability given in [TUK95].

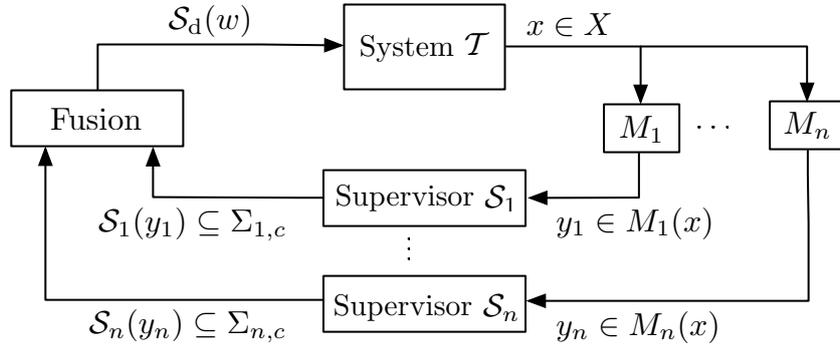


Figure 2.9 - Feedback interaction between the system \mathcal{T} to be controlled and the n local supervisor \mathcal{S}_i ($\forall i \in [1, n]$) in a decentralized state-based approach.

- the computation of the supervisor \mathcal{S} , which gives the actions that are forbidden by the supervisor. This function is defined, for each $y \in Y$, by $\mathcal{S}(y) \stackrel{\text{def}}{=} \{\sigma \in \Sigma_c \mid \exists x_1 \in Q^\uparrow, \exists x_2 \notin Q^\uparrow : (y = M(x_1)) \wedge (\langle x_1, \sigma, x_2 \rangle \in \Delta)\}$. This supervisor forbids an action σ in the observation y if there exists a safe state $x_1 \in Q^\uparrow$ compatible with y , which leads to a bad state $x_2 \notin Q^\uparrow$ by σ .

Since the system is finite and the function g is monotonic, all computations terminate; in particular, there exists an integer n such that $Q^\uparrow = \bigcap_{j=0}^n Q_j$.

However, this method does not always give a maximal supervisor and no information is given by the authors regarding the computation of such a solution. To our knowledge, the deadlock free or non-blocking cases have not been considered.

In [KGM93], Kumar *et al.* also provide an algorithm which synthesizes a supervisor (when a solution exists) such that the resulting controlled system exactly achieves the set Bad^c of good states. But their supervisor uses the entire sequence of observations received from the system to define its control policy (this supervisor has thus a memory).

In [HTL08], Hill *et al.* extend the results in [TK97, TK98] by considering non-deterministic systems to be controlled and masks corresponding to coverings of the state space.

Decentralized Supervisory Control. In the decentralized state-based approach, the system is controlled by n local supervisors \mathcal{S}_i ($\forall i \in [1, n]$). Each supervisor \mathcal{S}_i controls the set of actions $\Sigma_{i,c}$ and cannot control the set of actions

$\Sigma_{i,uc}$. The subsets $\Sigma_{1,c}, \dots, \Sigma_{n,c}$ are not necessarily disjoint. The set of actions that can be controlled by at least one controller is denoted by Σ_c and is defined by $\Sigma_c \stackrel{\text{def}}{=} \bigcup_{i=1}^n \Sigma_{i,c}$. The set of actions that cannot be controlled is denoted by Σ_{uc} and is defined by $\Sigma_{uc} \stackrel{\text{def}}{=} \Sigma \setminus \Sigma_c$. The partial observation of \mathcal{S}_i is modeled by the mask $M_i : X \rightarrow 2^{Y_i}$. The feedback interaction between the system to be controlled and the n local supervisors is depicted in Figure 2.9 and works as follows. When the system arrives in a state $x \in X$, each supervisor receives an observation $y_i \in M_i(x)$. From this information, \mathcal{S}_i defines a set $\mathcal{S}_i(y_i)$ of actions, that it proposes to forbid, and the global control applied to the system is the fusion, according to some rule to be defined, of the control decisions $\mathcal{S}_i(y_i)$ of all supervisors.

In [TKU94], Takai *et al.* work on a decentralized framework, where the fusion rule \mathcal{R}_c is given by the intersection of the actions allowed by each supervisor \mathcal{S}_i . It is thus a *conjunctive* architecture and their decentralized supervisor \mathcal{S}_d is defined by the pair $\langle (\mathcal{S}_i)_{i=1}^n, \mathcal{R}_c \rangle$. In their framework, the system $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$ to be controlled is deterministic and the partial observation is modeled, for each supervisor \mathcal{S}_i , by a mask $M_i : X \rightarrow Y_i$ corresponding to a *partition* of the state space. They provide a necessary and sufficient condition for the existence of a balanced decentralized supervisor \mathcal{S}_d that allows the system to exactly reach the set of states Bad^c (where Bad is the set of forbidden states). The decentralized supervisor \mathcal{S}_d is balanced if $\forall \sigma \in \Sigma, \forall x, x' \in X : [(x, x' \in \text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{S}_d}(x_0)) \wedge (\langle x, \sigma, x' \rangle \in \Delta)] \Rightarrow [\forall i \in [1, n] : \sigma \notin \mathcal{S}_i(M_i(x))]$, where $\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{S}_d}(x_0)$ denotes the set of states that are reachable in the system \mathcal{T} under the control of \mathcal{S}_d . Thus, for each pair of states x, x' reachable in the controlled system, if there is an action σ which can be fired from x to x' , then this action must be allowed by each local supervisor \mathcal{S}_i in the observation state $M_i(x)$. Their necessary and sufficient condition, given in Proposition 2.7, uses the following decentralized supervisor:

$$\tilde{\mathcal{S}}_d = \langle (\tilde{\mathcal{S}}_i)_{i=1}^n, \mathcal{R}_c \rangle \quad (2.4)$$

which prevents the system from reaching Bad . For each $i \in [1, n]$, the local supervisor $\tilde{\mathcal{S}}_i$ with partial observation is defined as follows:

$$\forall y \in Y_i, \tilde{\mathcal{S}}_i(y) \stackrel{\text{def}}{=} \{ \sigma \in \Sigma_c \mid \exists x_1 \notin Bad, \exists x_2 \in Bad : (y = M_i(x_1)) \wedge (\langle x_1, \sigma, x_2 \rangle \in \Delta) \} \quad (2.5)$$

Thus, the supervisor $\tilde{\mathcal{S}}_i$ forbids an action σ in the observation y if there exists a state $x_1 \notin Bad$ compatible with y , which leads to a forbidden state by σ .

Proposition 2.7 ([TKU94])

Let $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$ be the *deterministic* system to be controlled, $\Sigma_{i,c}$ and $\Sigma_{i,uc}$ ($\forall i \in [1, n]$) be a partition of Σ , $M_i : X \rightarrow Y_i$ ($\forall i \in [1, n]$) be a mask, and $Bad \subseteq X$ be the set of forbidden states, then there exists a decentralized supervisor $\mathcal{S}_d = \langle (\mathcal{S}_i)_{i=1}^n, \mathcal{R}_c \rangle$ such that $\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{S}_d}(x_0) = Bad^c$ if and only if Bad^c is n -observable⁶ i.e., the decentralized supervisor $\tilde{\mathcal{S}}_d$, defined in (2.4), satisfies the property $\text{Reachable}_{\Sigma}^{\mathcal{T}/\tilde{\mathcal{S}}_d}(x_0) = Bad^c$.

Obviously, if the n -observability condition holds, the decentralized supervisor $\tilde{\mathcal{S}}_d$ is a solution to this problem. This proposition means that if the system resulting from the supervision of $\tilde{\mathcal{S}}_d$ does not exactly reach Bad^c , then no decentralized supervisor can satisfy this requirement.

To our knowledge, the problem of computing a solution that allows the system to reach a subset of Bad^c when the n -observability property does not hold and also the problem which consists in ensuring the deadlock free or non-blocking property have not been studied in the literature. Moreover, we do not know works where the local supervisors can communicate or provide conditional decisions.

2.2 Control of Infinite State Systems

In the previous section, we have presented control methods for systems modeled by regular languages or equivalently by *finite automata*. However, in many industrial applications, the state space of the model of the system can be infinite. For example, the state space is infinite, when the model of the system uses variables that take their value in an infinite domain. Unfortunately, the control theory presented in section 2.1 cannot trivially be extended to infinite state systems, because most of the control problems become *undecidable*. In the literature, there are mainly two approaches to overcome this undecidability: either we consider subcases, in which the problem we want to solve is *decidable*, or we use approximation techniques to compute a *correct approximation* of the solution in a finite number of steps.

The formalism of automata is not suitable for the representation of infinite state systems. Indeed, they cannot always represent these systems with a finite structure. In the literature, there exist several formalisms to efficiently han-

⁶We give a simplified (and equivalent) version of the definition of n -observability given in [TKU94].

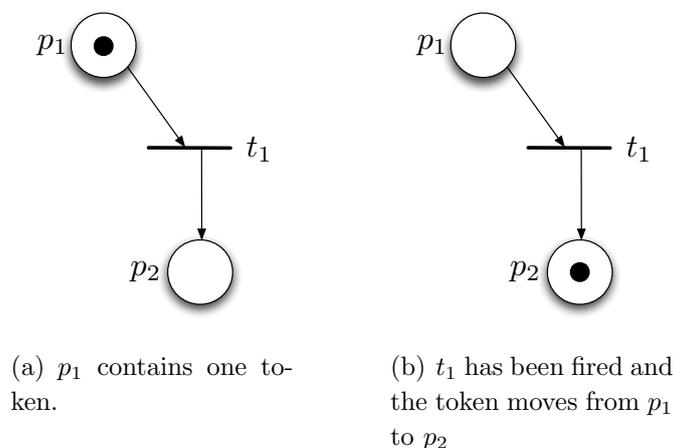


Figure 2.10 - A Petri net with two places and one transition.

dle infinite state systems. We can, for example, mention *Petri nets* [Pet62], *timed automata* [AD90], *extended automata* [SAF09], *symbolic transitions systems* [HMR05],... In the remainder of this section, we present works related to the control of infinite state systems modeled by these formalisms.

2.2.1 Controller Synthesis with Petri Nets

An alternative formalism for discrete event systems is provided by Petri nets [Pet62]. Petri nets are well suitable for the modeling of the concurrent behavior of distributed systems. They allow us to represent more easily complex systems that cannot compactly be represented by automata, because of the *combinatorial explosion* of the number of states. Petri nets are more expressive than finite automata and they can also represent some infinite state systems.

A Petri net has a finite set of *places*, a finite set of *transitions*, and an *incidence relation* which indicates the evolution of the system w.r.t. the places and the transitions. The places and the transitions are connected by *directed arcs*. An arc can be directed from a place to a transition or from a transition to a place. For a given transition t , the *input* (resp. *output*) places of t are the places that lead to (resp. are reachable from) t by a directed arc. In Petri nets, the mechanism, that indicates whether a transition can be fired, is provided by *tokens*, which are assigned to places. A transition t may be fired whenever there is a token in all input places of t . When it is fired, it consumes these tokens and places tokens in all output places. Figure 2.10(a) illustrates a Petri net with two places p_1

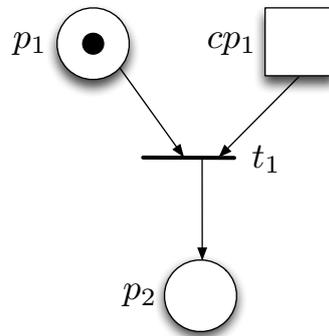


Figure 2.11 - A controlled Petri net with a control place cp_1 which controls the transition t_1 .

and p_2 (p_1 contains one token) and with one transition t_1 ; when t_1 is fired (see Figure 2.10(b)) the token moves from p_1 to p_2 . At each step of the execution, the places have a certain number of tokens and the *marking*, which indicates the number of tokens in each place, defines the state of the system at this step. One can note that a Petri net has an *initial* marking and can have an infinite number of markings. A marking m_1 covers a marking m_2 if, for each place p of the Petri net, the number of tokens in p for m_1 is greater than or equal to the number of tokens in p for m_2 . A set M of markings is *right-closed* if, for each marking $m \in M$, all states, that cover m , are in M .

The control of Petri nets has been developed by Holloway and Krogh [Kro87, HKG97, KH91]. To control the behavior of Petri nets, they add *control places* to the system; this extended model is called *controlled Petri net*. A control place is connected to one or several transitions and it inhibits them, when it contains no token. Figure 2.11 illustrates this control mechanism: the control place cp_1 controls the transition t_1 and this transition cannot be fired, because cp_1 contains no token.

In [KG05], Garg and Kumar prove that the state avoidance control problem for Petri nets, which consists in preventing the system from reaching a set of forbidden states (markings), is undecidable. They also prove that this problem is decidable when the set of forbidden markings is right-closed and they provide an algorithm which computes the optimal control policy. In [Kro87, KH91, KH90], Holloway and Krogh define several assumptions which imply that the Petri nets they consider have a finite number of different markings and are equivalent to automata. They give a necessary and sufficient condition for the existence of an

optimal control policy which satisfies the control objective. They also provide this control policy, when this condition is satisfied. In [HB91], Baosheng and Hoaxun introduce the concept of partial observation and define a distributed control for controlled Petri nets where the aim is to prevent the system from reaching a set of forbidden markings. Each local controller has only a partial observation of the system, which is modeled by a function mapping a marking onto an observation class. A controller is unable to distinguish between the markings of a same observation class and it must then produce the same control for these markings. They solve the problem by adding *coordination places* to the system in order to synchronize the controllers and by using the approach of Holloway and Krogh [KH90] to prevent the system from reaching the forbidden states.

When the specification is given by a language generated by a Petri net (note that this language is not necessarily regular), it has been shown in [Sre93] that deciding if this language is controllable is undecidable. However, the problem is decidable when the specification language is generated by a deterministic Petri net [Sre93]. In [KH96], Holloway and Kumar present an algorithm computing the optimal control policy when the system to be controlled is a deterministic Petri net and the specification is a regular language. The main idea of their algorithm consists in restricting this problem into a state avoidance control problem, which is obtained by computing the synchronous product of the system to be controlled and of the specification.

Works ensuring the deadlock free property and the non-blocking property have also been developed (e.g., [VNJ90, KH92]).

2.2.2 Control of Timed Automata

The formalism of timed automata is a widely used model for representing real-time systems. A timed automaton [AD90] has a finite set of locations, a finite set of actions, a finite set of real-valued clocks and a finite set of transitions. Each transition is labeled by an action, has a guard which depends on the clocks, and can reset clocks.

Figure 2.12 illustrates a timed automaton which has two locations ℓ_1 and ℓ_2 , one clock x and two actions a and b . The transitions δ_1 can be fired when the value of the clock x is less than or equal to 4. When it is fired, this clock is reset.

A state of the underlying labeled transition system of a timed automaton \mathcal{T} is given by a pair $\langle x, t \rangle$ where x is a location of \mathcal{T} and t is the instant in which the

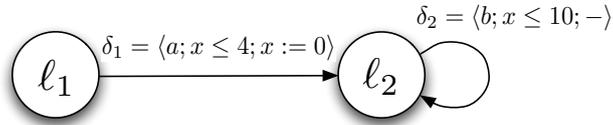


Figure 2.12 - An example of timed automata.

system is in x (more precisely, t gives the value of the clocks). Since the clocks have real values, the number of states of this labeled transition system can be infinite. However, a timed automaton can be translated into a finite automaton, called *region automaton*, which abstracts timed behaviors of the timed automaton into untimed behaviors and which preserves the semantics of the system. This abstraction has been proposed by Alur and Dill [AD90, AD94] and is the core of several decidability results (e.g. reachability, coreachability, emptiness, ...) for the model of timed automata.

The control mechanism for timed automata consists in partitioning the set of actions into the subset of *controllable* actions and the subset of *uncontrollable* actions. Most of the works on the control of timed automata are based on *game theory* where the aim is to compute a *winning strategy*. A winning strategy is a function that tells precisely what the controller has to do in order to satisfy the requirements [MPS95].

In [AMPS98], Asarin *et al.* consider a specification defined by a set of forbidden states and they provide an algorithm which computes the set of winning strategies for this specification. These strategies can be computed thanks to the concept of region automaton. In [HK99], Henzinger and Kopke prove that the safety control problem, which consists in deciding if there is a winning strategy for a specification given by a set of forbidden states, is EXPTIME-complete. In [CDF⁺05], an efficient approach computing a solution for this problem is defined and implemented in the tool Uppaal TiGA.

If the specification is defined by a timed automaton over infinite words, the control problem, which consists in deciding if there is a winning strategy for this specification, is undecidable [DM02]. However, the problem is decidable when the specification is given by a deterministic finite automaton; the problem is actually 2EXPTIME-complete.

The concept of partial observation has also been developed for the control of timed automata. This partial observation means that the controller does not observe certain actions and it is modeled by a partition of the set of actions

into the subset of observable actions and the subset of unobservable actions. In [BDMP03], Bouyer *et al.* suppose that only uncontrollable actions can be unobservable. They prove that the safety control problem under partial observation is undecidable. They also prove that the control problem under partial observation for specifications defined by deterministic timed automata over infinite words is undecidable.

2.2.3 Control of Systems using Variables

For practical reasons, complex systems are generally modeled by using variables. Finite automata are not suitable enough to represent these systems, specially when the domain of the variables is infinite. There exist several formalisms using variables and we mention some of them below.

In [KG05], Garg and Kumar use the *assignment program framework* to model infinite state systems [KGM93]. This formalism is composed of variables and of a finite sequence of instructions. Each instruction is labeled by an action (the set of actions is finite), and has a guard (which depends on the variables) and an assignment statement (which assigns new values to the variables). They prove that the problem (called *state avoidance control problem*) that consists in deciding if there is a controller with full observation, which prevents the system from reaching a set of forbidden states, is undecidable.

In [SAF09], Sköldstam *et al.* use the model of *Extended Automata*. This model is similar to the formalism of automata except that it has variables and each transition is composed of an action (the set of actions is finite), a guard (which depends on the variables), and an assignment (which assigns new values to the variables). They provide a method which takes a collection of extended automata as input and which efficiently translates them into an automaton to use the classical supervisory control theory on this system. To ensure the termination of the computations, they suppose that the domain of the variables is finite.

In [LJM05], Le Gall *et al.* use the model of *symbolic transition systems* [HMR05]. This model is defined by a finite set of variables, an infinite set of actions, and a finite set of symbolic transitions. Each transition is labeled by an action, and has a guard and an update function (a detailed description of this model is given in chapter 3). They use *symbolic techniques* to tackle the state avoidance control problem and use *abstract interpretation techniques* [CC77] to overcome the undecidability of this problem. These techniques ensure that their algorithm, which synthesizes controllers for the state avoidance control prob-

lem, always terminates, but they may overapproximate some computations. The computed controllers are *memoryless* and have a *full observation* of the system.

This work has been extended in [GD07] to tackle the control of concurrent systems modeled by extended automata (the specification is defined by a language). The domain of the variables can be infinite, which explains the use of abstract interpretation to ensure the termination of the computations.

2.3 Conclusion

In this chapter, we have outlined the basic concepts of supervisory control theory for finite discrete event systems by presenting some important results. The control of infinite state systems also received attention in the literature and we have presented interesting results in this domain. This topic is harder, because most of the control problems are undecidable. To define *effective* algorithms (i.e., which always terminate), the authors generally restrict their attention to particular cases in which the problems are decidable. Few works use abstract interpretation techniques to ensure the termination of the algorithms. We can, nevertheless, mention [LJM05], where this approach is used to compute memoryless controllers with full observation for the state avoidance control problem.

As explained in the introduction of this thesis, we are interested in the control of infinite state systems with partial observation. Our aim is to define effective algorithms, which synthesize controllers for these systems. To ensure the termination of our algorithms, we use, as in [LJM05], abstract interpretation techniques.

Part II

Control of Infinite State Systems

Chapter 3

Synthesis of Centralized Memoryless Controllers with Partial Observation for Infinite State Systems

 IN this chapter, we address the *state avoidance control problem* where the aim is to synthesize controllers which prevent the system from reaching a set of forbidden states. We place ourself in the context of infinite state systems to be controlled that are partially observed by the controller. Indeed, to model realistic systems, it is often convenient to manipulate state variables instead of simply atomic states, allowing a compact way to specify systems handling data. In that case, to provide a homogeneous treatment of these models, it is convenient to consider variables, whose domain is infinite, and thus systems with an *infinite state space*. Moreover, from a control point of view, the controller interacts with the plant through *sensors* and *actuators* and, in general, it has only a *partial observation* of the system, because these materials have not an absolute precision or some parts of the plant might not be observed by the controller.

Our aim is to propose *effective* algorithms (i.e., which always terminate) for the control of infinite state systems with partial observation; one can note that this problem is generally *undecidable*. We use the notion of mask defined in [DDR06, HTL08] to model partial observation. In these works, a mask is de-

defined as a *covering* of the state space and this definition thus generalizes the one given in [KGM93], where a mask corresponds to a *partition* of the state space. The state avoidance control problem that we want to solve is undecidable. To overcome this negative result, we follow the approach defined in [LJM05] where an effective algorithm synthesizing memoryless controllers with full observation is obtained by using abstract interpretation techniques. These techniques over-approximate some computations to ensure their termination. In this chapter, we provide effective algorithms computing memoryless controllers with partial observation. In the next chapter, we extend our method to synthesize controllers with memory and online controllers. Our algorithms have been implemented to be evaluated experimentally.

An infinite state system can obviously be modeled by an automaton, but the lack of structure of this formalism makes very hard the manipulation of such systems. To work with these systems, it is more convenient to use the formalism of *symbolic transition systems* (or *STS* for short), which allow us to represent infinite state systems with a finite syntax; we define this model in section 3.1. In section 3.2, we explain how the state avoidance control problem can be used to solve safety properties. Next, in section 3.3, we formally define the control mechanisms, that we use, and the state avoidance control problem. In section 3.4, we define a *semi-algorithm* (i.e., which does not always terminate) which synthesizes controllers for the state avoidance control problem and then we explain how to use abstract interpretation to obtain an *effective* algorithm (i.e., which always terminates). In section 3.5, we extend our method to synthesize controllers which ensure that the controlled system is deadlock free and, in section 3.6, we explain why our approach cannot be used for the non-blocking case. Finally, in section 3.7, we describe the experimental results obtained with our method.

3.1 Symbolic Transition Systems

To model realistic systems, it is often convenient to manipulate state variables instead of simply atomic states. The techniques and associated tools derived from the labeled transition systems do not explicitly take into account the data as the underlying model of transition systems which implies that the variables must be instantiated during the state space exploration and analysis. This enumeration of the possible values of the variables leads to the classical state space explosion when these variables take their values in a finite set, but may also render the computations infeasible whenever the domain of the variables is infinite. In this

thesis, we model the systems to be controlled by *symbolic transition systems*, which offer a compact way to specify systems handling data. STS is a model of (infinite) systems defined over variables, whose domain can be infinite, and composed of a finite set of *symbolic transitions*. Each transition has a *guard*, which indicates when it can be fired, and an *update* function, which indicates the evolution of the variables when it is fired. Furthermore, transitions are labeled by symbols (called *actions*) taken from a finite alphabet. With this formalism infinite state systems (note that the system is infinite whenever the domain of the variables is infinite) can be represented in a finite way.

In the remainder of this section, we formally define the model of STS and we present some operations and properties related to this formalism.

3.1.1 Model of STS

Symbolic Transition Systems. A symbolic transition system is formally defined as follows:

Definition 3.1 (Symbolic Transition System)

A *symbolic transition system* (or *STS* for short) is defined by a 5-tuple $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ where¹:

- $V \stackrel{\text{def}}{=} \langle v_1, \dots, v_n \rangle$ is an n -tuple of variables (n is constant).
- $\Theta_0 \subseteq \mathcal{D}_V$ is a predicate on V , which defines the set of *initial states*.
- $\Theta_m \subseteq \mathcal{D}_V$ is a predicate on V , which defines the set of *marked (or final) states*.
- Σ is a finite alphabet of *actions*.
- Δ is a finite set of *symbolic transitions* $\delta = \langle \sigma_\delta, G_\delta, A_\delta \rangle$ where (i) $\sigma_\delta \in \Sigma$ is the action of δ , (ii) $G_\delta \subseteq \mathcal{D}_V$ is a predicate on V , which defines the *guard* of δ , and (iii) $A_\delta : \mathcal{D}_V \rightarrow \mathcal{D}_V$ is the *update function* (or *assignment function*) of δ .

Throughout this thesis, we suppose that the update functions A_δ of an STS are *continuous* to ensure the existence of a least fixpoint in the fixpoint equations

¹When we will not need the set of final states Θ_m , we will define an STS by a 4-tuple $\langle V, \Theta_0, \Sigma, \Delta \rangle$. In some works (e.g., [LJM05]), STS can have an infinite set Σ of actions. Here we assume that Σ is finite to ensure that our controllers can be computed (see section 3.4.1).

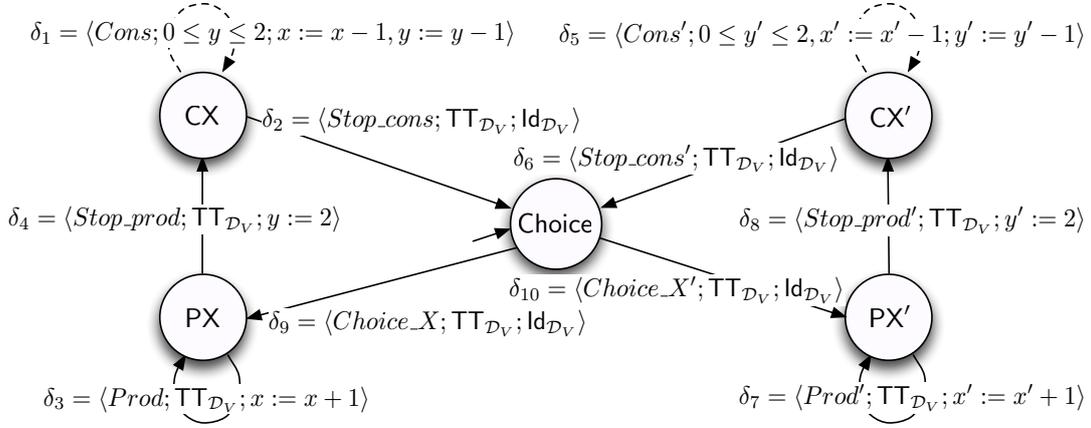


Figure 3.1 - Producer and consumer example.

that we will define.

The semantics of an STS is defined by a possibly infinite labeled transition system (or LTS for short) whose states are valuations of the variables:

Definition 3.2 (Semantics of an STS)

The semantics of an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ is defined by an LTS $\llbracket \mathcal{T} \rrbracket \stackrel{\text{def}}{=} \langle X, X_0, X_m, \Sigma, \Delta_{\llbracket \mathcal{T} \rrbracket} \rangle$ where:

- $X \stackrel{\text{def}}{=} \mathcal{D}_V$ is the set of states
- $X_0 \stackrel{\text{def}}{=} \Theta_0$ is the set of initial states
- $X_m \stackrel{\text{def}}{=} \Theta_m$ is the set of final states
- Σ is the set of labels
- $\Delta_{\llbracket \mathcal{T} \rrbracket} \subseteq X \times \Sigma \times X$ is the transition relation defined by $\Delta_{\llbracket \mathcal{T} \rrbracket} \stackrel{\text{def}}{=} \{ \langle \vec{v}_1, \sigma, \vec{v}_2 \rangle \mid \exists \langle \sigma_\delta, G_\delta, A_\delta \rangle \in \Delta : (\sigma_\delta = \sigma) \wedge (\vec{v}_1 \in G_\delta) \wedge (\vec{v}_2 = A_\delta(\vec{v}_1)) \}$.

An STS thus begins its execution in an initial state. Then, in any state, a transition can be fired only if its guard is satisfied and, in this case, the variables are updated according to the update function.

Remark 3.1

The formalism presented in Definition 3.1 does not allow us to *explicitly* define locations. But, this model is actually equivalent to a model with explicit locations [JMR06], because a variable of enumerated type can always be added to encode the locations. To have clearer illustrations in our examples, the figures, that depict STS, will have explicit locations that will be encoded by this mechanism.

Example 3.1

The STS² of Figure 3.1 is a system of stock management. Two units produce and send (consume) two kinds of parts X and X' . The STS has a tuple $V = \langle \ell, x, x', y, y' \rangle$ of five variables, where (i) $\ell \in \{\text{CX}, \text{PX}, \text{Choice}, \text{CX}', \text{PX}'\}$ gives the current location of the system, (ii) x (resp. x') gives the number of parts X (resp. X') and (iii) y (resp. y') gives the number of parts X (resp. X') that can be produced. The initial state is $\langle \text{Choice}, 0, 0, 0, 0 \rangle$. The choice of the kind of parts to produce is performed in the location **Choice**: the action Choice_X (resp. $\text{Choice}_{X'}$) allows the system to produce parts X (resp. X'). In the location **PX** (resp. **PX'**), the action Prod (resp. Prod') produces a part X (resp. X') whereas the action Stop_prod (resp. $\text{Stop_prod}'$) stops the production process. In the location **CX** (resp. **CX'**), the action Cons (resp. Cons') consumes a part X (resp. X') and the action Stop_cons (resp. $\text{Stop_cons}'$) stops the consumption process. The variables y and y' ensure that at most three parts can be consumed in each consumption cycle.

Generated and Marked Languages. The language $\mathcal{L}(\mathcal{T})$, which represents the set of finite words w generated by \mathcal{T} , is defined by $\mathcal{L}(\mathcal{T}) \stackrel{\text{def}}{=} \mathcal{L}(\llbracket \mathcal{T} \rrbracket)$ and the language $\mathcal{L}_m(\mathcal{T})$, which represents the set of finite words w marked by \mathcal{T} , is defined by $\mathcal{L}_m(\mathcal{T}) \stackrel{\text{def}}{=} \mathcal{L}_m(\llbracket \mathcal{T} \rrbracket)$.

Reachability and Coreachability. A state \vec{v}_1 is *reachable* (resp. *coreachable*) from a state \vec{v}_2 in an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ if and only if it is reachable (resp. coreachable) from \vec{v}_2 in the LTS $\llbracket \mathcal{T} \rrbracket$ i.e., it is reachable (resp. coreach-

²For convenience, in the guards and the update functions of the transitions of the system, we omit the conditions and assignments related to the locations. For example, the transition δ_2 is defined by $\langle \text{Stop_cons}; \text{TT}_{\mathcal{D}_V}; \text{ld}_{\mathcal{D}_V} \rangle$, whereas it should be defined by $\langle \text{Stop_cons}; \ell = \text{CX}; \ell := \text{Choice} \rangle$.

able) from \vec{v}_2 by a finite sequence of transitions³.

Deterministic Symbolic Transition System. An STS \mathcal{T} is *deterministic* if and only if the LTS $\llbracket \mathcal{T} \rrbracket$ is deterministic⁴. But deciding if \mathcal{T} is *deterministic* is generally *undecidable* [Dub06]. Since it is not always possible to check the determinism of an STS, we consider a stronger notion of determinism called *structural determinism* and defined as follows:

Definition 3.3 (Structural Determinism)

An STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ is *structurally deterministic* if and only if $\forall \langle \sigma_1, G_1, A_1 \rangle \neq \langle \sigma_2, G_2, A_2 \rangle \in \Delta : (\sigma_1 = \sigma_2) \Rightarrow (G_1 \cap G_2 = \emptyset)$.

The following property shows that the structural determinism implies the determinism:

Proposition 3.1

If an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ is structurally deterministic, then it is deterministic.

Proof

Let us suppose that \mathcal{T} is structurally deterministic and non-deterministic. The non-determinism of \mathcal{T} implies that there exist transitions $\langle \sigma_1, G_1, A_1 \rangle, \langle \sigma_2, G_2, A_2 \rangle \in \Delta$ and states $\vec{v}_1, \vec{v}_2, \vec{v}_3 \in \mathcal{D}_V$ such that (i) $\sigma_1 = \sigma_2$, (ii) $\vec{v}_1 \in G_1$, (iii) $\vec{v}_2 = A_1(\vec{v}_1)$, (iv) $\vec{v}_1 \in G_2$, (v) $\vec{v}_3 = A_2(\vec{v}_1)$, and (vi) $\vec{v}_2 \neq \vec{v}_3$. Therefore, \mathcal{T} is not structurally deterministic, because the transitions $\langle \sigma_1, G_1, A_1 \rangle$ and $\langle \sigma_2, G_2, A_2 \rangle$ are such that (i) $\sigma_1 = \sigma_2$ and (ii) $G_1 \cap G_2 \neq \emptyset$, which gives a contradiction.

The structural determinism can be checked on an STS if the class of formulae used to define the predicates allows us to check the satisfiability of the formulae.

3.1.2 Properties and Operations on STS

In this subsection, we recall some properties and operations on STS. This recall is not exhaustive; we only present the notions that we use in this thesis.

³The formal definition of a reachable (resp. coreachable) state in an LTS is given in Definition 1.5 (resp. Definition 1.6)).

⁴The formal definition of a deterministic LTS is given in section 1.2.

Notations. Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ and an action $\sigma \in \Sigma$, the notation $\text{Trans}(\sigma) \stackrel{\text{def}}{=} \{ \langle \sigma_\delta, G_\delta, A_\delta \rangle \in \Delta \mid \sigma_\delta = \sigma \}$ gives the set of transitions labeled by σ . Given a transition $\delta \in \Delta$ and two states $\vec{v}_1, \vec{v}_2 \in \mathcal{D}_V$, $\vec{v}_1 \xrightarrow{\delta} \vec{v}_2$ denotes the fact that \vec{v}_2 is reachable from \vec{v}_1 by the transition δ .

Backward and Forward Operators. Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, a subset of transitions $D \subseteq \Delta$, and a subset of actions $A \subseteq \Sigma$, we define the following backward and forward operators:

- the function $\text{Pre}_D^{\mathcal{T}} : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ is defined, for each $B \subseteq \mathcal{D}_V$, by $\text{Pre}_D^{\mathcal{T}}(B) \stackrel{\text{def}}{=} \bigcup_{\langle \sigma, G, A \rangle \in D} (G \cap A^{-1}(B)) \stackrel{\text{def}}{=} \bigcup_{\langle \sigma, G, A \rangle \in D} \{ \vec{v}_1 \in \mathcal{D}_V \mid \exists \vec{v}_2 \in B : (\vec{v}_1 \in G) \wedge (\vec{v}_2 = A(\vec{v}_1)) \}$. Thus, $\text{Pre}_D^{\mathcal{T}}(B)$ gives the set of states that lead to B by a transition of D .
- the function $\text{Post}_D^{\mathcal{T}} : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ is defined, for each $B \subseteq \mathcal{D}_V$, by $\text{Post}_D^{\mathcal{T}}(B) \stackrel{\text{def}}{=} \bigcup_{\langle \sigma, G, A \rangle \in D} (A(G \cap B)) \stackrel{\text{def}}{=} \bigcup_{\langle \sigma, G, A \rangle \in D} \{ \vec{v}_1 \in \mathcal{D}_V \mid \exists \vec{v}_2 \in B : (\vec{v}_2 \in G) \wedge (\vec{v}_1 = A(\vec{v}_2)) \}$. Thus, $\text{Post}_D^{\mathcal{T}}(B)$ gives the set of states that are reachable from B by a transition of D .
- the function $\text{Pre}_A^{\mathcal{T}} : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ is defined, for each $B \subseteq \mathcal{D}_V$, by $\text{Pre}_A^{\mathcal{T}}(B) \stackrel{\text{def}}{=} \bigcup_{\sigma \in A} \text{Pre}_{\text{Trans}(\sigma)}^{\mathcal{T}}(B)$. Thus, $\text{Pre}_A^{\mathcal{T}}(B)$ gives the set of states that lead to B by a transition labeled by an action of A .
- the function $\text{Post}_A^{\mathcal{T}} : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ is defined, for each $B \subseteq \mathcal{D}_V$, by $\text{Post}_A^{\mathcal{T}}(B) \stackrel{\text{def}}{=} \bigcup_{\sigma \in A} \text{Post}_{\text{Trans}(\sigma)}^{\mathcal{T}}(B)$. Thus, $\text{Post}_A^{\mathcal{T}}(B)$ gives the set of states that are reachable from B by a transition labeled by an action of A .

Coreachability and Reachability Operators. Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, a subset of transitions $D \subseteq \Delta$, and a subset of actions $A \subseteq \Sigma$, we define the following functions:

- the function $\text{Coreach}_D^{\mathcal{T}} : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ is defined, for each $B \subseteq \mathcal{D}_V$, by $\text{Coreach}_D^{\mathcal{T}}(B) \stackrel{\text{def}}{=} \bigcup_{n \geq 0} (\text{Pre}_D^{\mathcal{T}})^n(B)$. Thus, $\text{Coreach}_D^{\mathcal{T}}(B)$ gives the set of states that lead to B by a sequence of transitions of D .
- the function $\text{Reachable}_D^{\mathcal{T}} : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ is defined, for each $B \subseteq \mathcal{D}_V$, by $\text{Reachable}_D^{\mathcal{T}}(B) \stackrel{\text{def}}{=} \bigcup_{n \geq 0} (\text{Post}_D^{\mathcal{T}})^n(B)$. Thus, $\text{Reachable}_D^{\mathcal{T}}(B)$ gives the set of states that are reachable from B by a sequence of transitions of D .

- the function $\text{Coreach}_A^{\mathcal{T}} : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ is defined, for each $B \subseteq \mathcal{D}_V$, by $\text{Coreach}_A^{\mathcal{T}}(B) \stackrel{\text{def}}{=} \text{Coreach}_{\text{Trans}(A)}^{\mathcal{T}}(B)$. Thus, $\text{Coreach}_A^{\mathcal{T}}(B)$ gives the set of states that lead to B by a sequence of transitions labeled by actions of A .
- the function $\text{Reachable}_A^{\mathcal{T}} : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ is defined, for each $B \subseteq \mathcal{D}_V$, by $\text{Reachable}_A^{\mathcal{T}}(B) \stackrel{\text{def}}{=} \text{Reachable}_{\text{Trans}(A)}^{\mathcal{T}}(B)$. Thus, $\text{Reachable}_A^{\mathcal{T}}(B)$ gives the set of states that are reachable from B by a sequence of transitions labeled by actions of A .

In this thesis, our algorithms, which synthesize controllers for STS, are based on reachability and coreachability operations i.e., the main operations consist in computing the set of states that are reachable or coreachable from a given set of states. Therefore, our main operations on STS work on sets of states. In our tool SMACS (see appendix A), the sets of states are symbolically represented by predicates and the operations on these sets are performed by *predicate transformers*, which are functions mapping a predicate to another predicate. For example, in our tool we compute $\text{Pre}_\delta^{\mathcal{T}}(B)$ by mapping the predicate $B : \mathcal{D}_V \rightarrow \{\text{tt}, \text{ff}\}$ (which is defined, for each $\vec{v} \in \mathcal{D}_V$, by $B(\vec{v}) \stackrel{\text{def}}{=} \text{tt}$ if and only if $\vec{v} \in B$) into the predicate $\text{Pre}_\delta^{\mathcal{T}}(B) : \mathcal{D}_V \rightarrow \{\text{tt}, \text{ff}\}$ which is defined, for each $\vec{v}_1 \in \mathcal{D}_V$, by $\text{Pre}_\delta^{\mathcal{T}}(B)(\vec{v}_1) \stackrel{\text{def}}{=} \exists \vec{v}_2 \in B : (\vec{v}_1 \in G) \wedge (\vec{v}_2 = A(\vec{v}_1))$ (where G and A are respectively the guard and the update function of δ).

Synchronous Product Operator. The *synchronous product* consists in computing, from two STS \mathcal{T}_1 and \mathcal{T}_2 , a new STS whose behavior corresponds to the intersection of the behaviors of \mathcal{T}_1 and \mathcal{T}_2 . This operation is formally defined as follows:

Definition 3.4 (Synchronous Product)

Given two STS $\mathcal{T}_1 = \langle V_1, \Theta_{0,1}, \Theta_{m,1}, \Sigma, \Delta_1 \rangle$ and $\mathcal{T}_2 = \langle V_2, \Theta_{0,2}, \Theta_{m,2}, \Sigma, \Delta_2 \rangle$ such that $V_1 \cap V_2 = \emptyset$, the *synchronous product* of \mathcal{T}_1 and \mathcal{T}_2 , denoted by $\mathcal{T}_1 \times \mathcal{T}_2$, is defined by an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ where:

- $V \stackrel{\text{def}}{=} V_1 \cup V_2$
- $\Theta_0 \stackrel{\text{def}}{=} \Theta_{0,1} \times \Theta_{0,2}$
- $\Theta_m \stackrel{\text{def}}{=} \Theta_{m,1} \times \Theta_{m,2}$

- the transition relation Δ is defined as follows: for each pair of transitions $\langle \sigma, G_1, A_1 \rangle \in \Delta_1$ and $\langle \sigma, G_2, A_2 \rangle \in \Delta_2$, we define a new transition $\langle \sigma, G_1 \cap G_2, A \rangle \stackrel{\text{def}}{\in} \Delta$ where the update function $A : \mathcal{D}_{V_1} \times \mathcal{D}_{V_2} \rightarrow \mathcal{D}_{V_1} \times \mathcal{D}_{V_2}$ is defined, for each $\vec{v}_1 \in \mathcal{D}_{V_1}$ and $\vec{v}_2 \in \mathcal{D}_{V_2}$, by $A(\langle \vec{v}_1, \vec{v}_2 \rangle) \stackrel{\text{def}}{=} \langle A_1(\vec{v}_1), A_2(\vec{v}_2) \rangle$.

The synchronous product satisfies the two following properties: (i) $\mathcal{L}(\mathcal{T}) = \mathcal{L}(\mathcal{T}_1) \cap \mathcal{L}(\mathcal{T}_2)$ and (ii) $\mathcal{L}_m(\mathcal{T}) = \mathcal{L}_m(\mathcal{T}_1) \cap \mathcal{L}_m(\mathcal{T}_2)$. Moreover, the synchronous product of two deterministic (resp. structurally deterministic) STS is a deterministic (resp. structurally deterministic) STS.

Properties. The *deadlock free* and *non-blocking* properties are defined as follows on STS:

Definition 3.5 (Deadlock Free Property)

An STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ is deadlock free if and only if the LTS $\llbracket \mathcal{T} \rrbracket$ is deadlock free. In other words, \mathcal{T} is deadlock free if and only if, for each state \vec{v} reachable in this system, there exists at least one transition that can be fired from this state (i.e., $\exists \langle \sigma, G, A \rangle \in \Delta : \vec{v} \in G$).

Definition 3.6 (Non-blocking Property)

An STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ is non-blocking if and only if the LTS $\llbracket \mathcal{T} \rrbracket$ is non-blocking.

3.2 State Invariance Properties

As mentioned above, we are interested in the *state avoidance control problem*, where the aim is to control the system to satisfy a *state invariance property*, which is defined by a set *Bad* of states that the system must avoid. In this section, we explain how a *safety property* can be reduced to a state invariance property by means of an *STS-observer*.

State Invariance Property and Safety Property. We first define the concept of *state invariance property*:

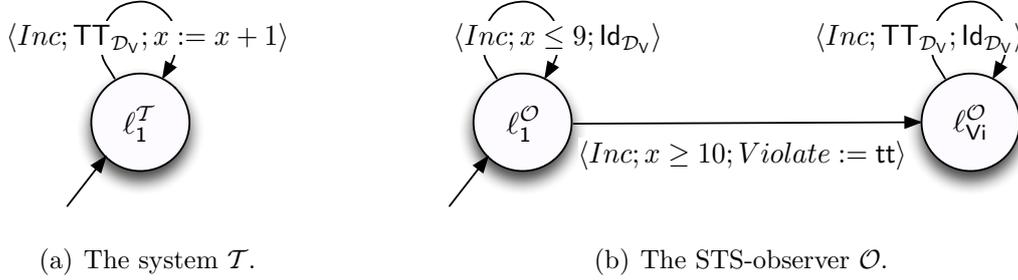


Figure 3.2 - An STS \mathcal{T} and an STS-observer \mathcal{O} for \mathcal{T} .

Definition 3.7 (State Invariance Property)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, a *state invariance property* is defined by a set of states $Bad \subseteq \mathcal{D}_V$ and the STS \mathcal{T} satisfies this state invariance property if Bad cannot be reached along the execution of \mathcal{T} i.e., $\text{Reachable}_{\Delta}^{\mathcal{T}}(\Theta_0) \cap Bad = \emptyset$.

Now, one can also want to specify more general properties like *safety properties*. In this setting a safety property ψ over an alphabet Σ is defined by a set of finite sequences of actions such that $\forall w_1 \in \Sigma^* : (w_1 \notin \psi) \Rightarrow (\forall w_2 \in \Sigma^* : w_1.w_2 \notin \psi)$. In other words, if a finite sequence violates the property ψ , no extension of this sequence will satisfy it. The absence of failure is a classical example of safety properties.

STS-observer. The concept of *STS-observer* is used to reduce a safety property to a state invariance property. Given an STS \mathcal{T} , an *STS-observer* of \mathcal{T} is an STS \mathcal{O} which is *non-intrusive* i.e., \mathcal{O} satisfies the following property: $\mathcal{L}(\mathcal{T} \times \mathcal{O}) = \mathcal{L}(\mathcal{T})$. The STS \mathcal{O} can share some variables with the STS \mathcal{T} to observe this system, but it cannot modify them⁵. Moreover, this STS has a particular boolean variable *Violate* and its set of marking states is given by the set $Violate^{\mathcal{O}}$ which denotes the set of states in which the variable *Violate* is equal to **tt**; one can note that once *Violate* becomes true, then so is forever. The language $\mathcal{L}_m(\mathcal{O})$ marked by \mathcal{O} corresponds to the set of sequences of actions which lead to a state where the variable *Violate* is equal to **tt**. In fact, an STS-observer encodes the negation of a safety property ψ and $\mathcal{L}_m(\mathcal{O})$ then represents the set of sequences that violate ψ .

⁵The synchronous product used to compute $\mathcal{T} \times \mathcal{O}$ is slightly different from the one given in Definition 3.4, because \mathcal{T} and \mathcal{O} can share variables.

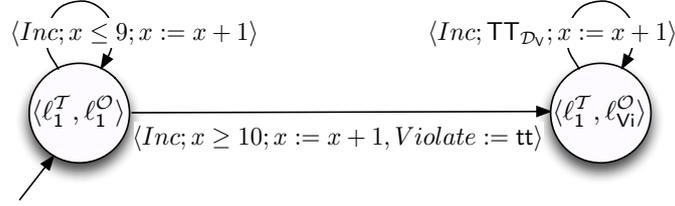


Figure 3.3 - Synchronous product $\mathcal{T} \times \mathcal{O}$.

Example 3.2

Let us consider the system $\mathcal{T} = \langle V_{\mathcal{T}}, (\Theta_0)_{\mathcal{T}}, (\Theta_m)_{\mathcal{T}}, \Sigma, \Delta_{\mathcal{T}} \rangle$ depicted in Figure 3.2(a) which increments a variable x . This STS has a tuple $V_{\mathcal{T}} = \langle \ell_{\mathcal{T}}, x \rangle$ of two variables where (i) $\ell_{\mathcal{T}} \in \{\ell_1^T\}$ gives the location of the system, and (ii) $x \in \mathbb{N}_0$. The set of initial states $(\Theta_0)_{\mathcal{T}} = \{\langle \ell_1^T, 0 \rangle\}$ and the set of marked states $\Theta_m = \mathcal{D}_{V_{\mathcal{T}}}$. An STS-observer, which encodes the negation of the safety property $\psi = \Box(x \leq 10)$ (i.e., x must always be less than 10), can be defined by the STS $\mathcal{O} = \langle V_{\mathcal{O}}, (\Theta_0)_{\mathcal{O}}, (\Theta_m)_{\mathcal{O}}, \Sigma, \Delta_{\mathcal{O}} \rangle$ depicted in Figure 3.2(b). This STS has a tuple $V_{\mathcal{O}} = \langle \ell_{\mathcal{O}}, x, Violate \rangle$ of three variables where (i) $\ell_{\mathcal{O}} \in \{\ell_1^O, \ell_{Vi}^O\}$ gives the current location of the system, (ii) $x \in \mathbb{N}_0$, and (iii) $Violate \in \{tt, ff\}$. The set of initial states $(\Theta_0)_{\mathcal{O}} = \{\langle \ell_1^O, 0, ff \rangle\}$ and the set of marked states $(\Theta_m)_{\mathcal{O}} = \{\langle \ell_{Vi}^O, x, tt \rangle \mid x \in \mathbb{N}_0\}$. The synchronous product $\mathcal{T} \times \mathcal{O}$ is depicted in Figure 3.3. In this system, the states, which are in the location $\langle \ell_1^T, \ell_{Vi}^O \rangle$, violate the property ψ .

Safety Property Reduced to a State Invariance Property. The use of an STS-observer allows us to reduce, thanks to the following proposition, a safety property on the system \mathcal{T} to a state invariance property on the system $\mathcal{T} \times \mathcal{O}$:

Proposition 3.2 (Safety Property and State Invariance Property)

Let \mathcal{T} be an STS and \mathcal{O} be an STS-observer for \mathcal{T} defining the safety property ψ . Then, \mathcal{T} satisfies ψ if and only if $\mathcal{T} \times \mathcal{O}$ satisfies the state invariance property $\mathcal{D}_V \times Violate^{\mathcal{O}}$.

Based on Proposition 3.2, we may only consider state invariance properties. Indeed, controlling \mathcal{T} to ensure the safety property ψ modeled by \mathcal{O} is equivalent to controlling $\mathcal{T} \times \mathcal{O}$ to ensure the state invariance property $\mathcal{D}_V \times Violate^{\mathcal{O}}$.

3.3 Framework and State Avoidance Control Problem

In this section, we define our used framework and the problems we are interested in. More precisely, in section 3.3.1, we define the kind of information available from the observation of the system. Next, in section 3.3.2, we define the control mechanisms. In section 3.3.3, we formally define the concepts of controller and of controlled system. Finally, in section 3.3.4, we define the state avoidance control problem and some of its variants.

3.3.1 Means of Observation

The controller interacts with the plant through *sensors* and *actuators* and, in general, it has only a *partial observation* of the system, because these materials have not an absolute precision or some parts of the system might not be observed by the controller. Since we consider a state-based approach, we use the concept of *observer* to model this partial observation:

Definition 3.8 (Observer)

An *observer* of the state space \mathcal{D}_V is a pair $\langle Obs, M \rangle$, where (i) *Obs* is a variable⁶, whose domain \mathcal{D}_{Obs} represents an observation space which can be *infinite*, and (ii) $M : \mathcal{D}_V \rightarrow 2^{\mathcal{D}_{Obs}}$ is a *mask* which gives, for each state $\vec{v} \in \mathcal{D}_V$, the set $M(\vec{v})$ (which can be *infinite*) of possible observations that the controller can receive when the system arrives in the state \vec{v} . Moreover, each state $\vec{v} \in \mathcal{D}_V$ must have at least one observation (i.e., $\forall \vec{v} \in \mathcal{D}_V : M(\vec{v}) \neq \emptyset$).

One can note that when the system reaches a state $\vec{v} \in \mathcal{D}_V$, the controller only receives *an* observation among the set $M(\vec{v})$ of possible observations. We say that a state $\vec{v} \in \mathcal{D}_V$ is *compatible with* an observation $obs \in \mathcal{D}_{Obs}$, if $obs \in M(\vec{v})$. In this definition, a mask M represents a *covering* of the state space. This definition generalizes the one given in [TUK95, KGM93], where a mask corresponds to a *partition* of the state space (i.e., each state $\vec{v} \in \mathcal{D}_V$ is compatible with exactly one observation). An observer $\langle Obs, M \rangle$ defines a *full observation* of the system when $Obs = V$ and $M = \text{Id}_{\mathcal{D}_V}$.

Throughout this thesis, we suppose that the mask M and its preimage function M^{-1} are continuous.

⁶We have chosen to define the observation space \mathcal{D}_{Obs} as the domain of the variable *Obs* to remain coherent with the formalization of the state space \mathcal{D}_V .

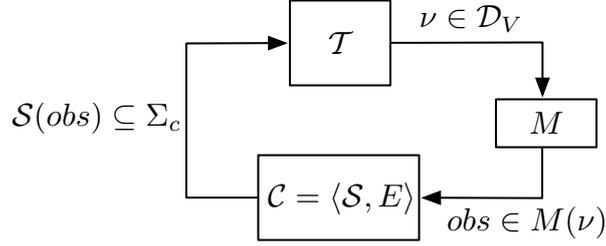


Figure 3.4 - Feedback interaction between the system \mathcal{T} and the controller \mathcal{C} .

Example 3.3

For the system depicted in Figure 3.1, an example of partial observation is given by the observer $\langle Obs, M \rangle$ where (i) Obs is a variable such that $\mathcal{D}_{Obs} = 2^{Loc \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}}$, and (ii) $M : Loc \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \rightarrow 2^{Loc \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}}$ is a mask such that, for each state $\vec{\nu} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, the set of states, that are undistinguishable from $\vec{\nu}$, is defined by $\{\langle \ell, x_1, x', y, y' \rangle \mid x_1 \in [x - 1, x + 1]\}$. This mask models the fact that there is an imprecision regarding the number of parts X .

3.3.2 Means of Control

Following the Ramadge & Wonham’s theory [RW89], we associate a controller \mathcal{C} with the system \mathcal{T} to be controlled. This controller interacts with \mathcal{T} in a *feedback manner* as illustrated in Figure 3.4: the controller observes the system and computes, from this observation, a set of actions that the system cannot execute in order to ensure the desired properties. Moreover, the alphabet Σ is partitioned into the set of *controllable* actions Σ_c and the set of *uncontrollable* actions Σ_{uc} ; only controllable actions can be forbidden by the controller. That also induces a partition of the set of symbolic transitions Δ into the set Δ_c of controllable transitions and the set Δ_{uc} of uncontrollable transitions: a transition $\langle \sigma, G, A \rangle \stackrel{\text{def}}{\in} \Delta_c$ (resp. Δ_{uc}) if its action $\sigma \in \Sigma_c$ (resp. Σ_{uc}).

3.3.3 Controller and Controlled System

The controller aims at restricting the behavior of the system to ensure a given property and it is formally defined as follows:

Definition 3.9 (Controller)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ and an observer $\langle Obs, M \rangle$, a *controller with partial observation* for the system \mathcal{T} is a pair $\mathcal{C} \stackrel{\text{def}}{=} \langle \mathcal{S}, E \rangle$, where:

- $\mathcal{S} : \mathcal{D}_{Obs} \rightarrow 2^{\Sigma^c}$ is a *supervisory function* which defines, for each observation $obs \in \mathcal{D}_{Obs}$, a set $\mathcal{S}(obs)$ of controllable actions that are forbidden when obs is observed by the controller.
- $E \subseteq \mathcal{D}_V$ is a set of states which restricts the set of initial states of \mathcal{T} .

This controller is *memoryless*, since its control decisions are only based on the current observation received from the system. We recall that when the system reaches a state $\vec{v} \in \mathcal{D}_V$, the controller receives, as information, an observation $obs \in M(\vec{v})$.

The system resulting from the feedback interaction between the system \mathcal{T} and the controller \mathcal{C} is called the *controlled system* and is defined by an STS having an additional variable to represent the observations that the controller receives from the system. It is formally defined as follows:

Definition 3.10 (Controlled System)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$, and a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, the system \mathcal{T} controlled by \mathcal{C} , denoted by $\mathcal{T}_{/\mathcal{C}}$, is an STS including observations which is defined by $\mathcal{T}_{/\mathcal{C}} \stackrel{\text{def}}{=} \langle V_{/\mathcal{C}}, (\Theta_0)_{/\mathcal{C}}, (\Theta_m)_{/\mathcal{C}}, \Sigma, \Delta_{/\mathcal{C}} \rangle$, where:

- $V_{/\mathcal{C}} \stackrel{\text{def}}{=} V \cup \langle Obs \rangle$ is a *pair* composed of the tuple of variables of V and of the variable Obs .
- $(\Theta_0)_{/\mathcal{C}} \stackrel{\text{def}}{=} \{ \langle \vec{v}, obs \rangle \mid (\vec{v} \in (\Theta_0 \setminus E)) \wedge (obs \in M(\vec{v})) \}$ is the set of initial states.
- $(\Theta_m)_{/\mathcal{C}} \stackrel{\text{def}}{=} \{ \langle \vec{v}, obs \rangle \mid (\vec{v} \in \Theta_m) \wedge (obs \in M(\vec{v})) \}$ is the set of marked states.
- $\Delta_{/\mathcal{C}}$ is defined from Δ as follows: for each transition $\langle \sigma, G, A \rangle \in \Delta$, we define a new transition $\langle \sigma, G_{/\mathcal{C}}, A_{/\mathcal{C}} \rangle \in \Delta_{/\mathcal{C}}$, where (i) $G_{/\mathcal{C}} \stackrel{\text{def}}{=} \{ \langle \vec{v}, obs \rangle \mid (\vec{v} \in G) \wedge (obs \in M(\vec{v})) \wedge (\sigma \notin \mathcal{S}(obs)) \}$ and (ii) $A_{/\mathcal{C}}$ is defined, for each $\vec{v}_1 \in \mathcal{D}_V$ and for each $obs_1 \in M(\vec{v}_1)$, by $A_{/\mathcal{C}}(\langle \vec{v}_1, obs_1 \rangle) \stackrel{\text{def}}{=} \{ \langle \vec{v}_2, obs_2 \rangle \mid (\vec{v}_2 = A(\vec{v}_1)) \wedge (obs_2 \in M(\vec{v}_2)) \}$.

A state $\langle \vec{v}, obs \rangle \in \mathcal{D}_{V/C}$ (with $obs \in M(\vec{v})$) of the controlled system models the fact that when the system is in the state \vec{v} , the controller receives the observation obs as information. The guards and the update functions of the controlled system $\mathcal{T}_{/C}$ are defined from the ones of the system \mathcal{T} by taking into account the control decisions of the controller \mathcal{C} and the observations that it receives from the system:

- **Guards:** A transition $\langle \sigma, G_{/C}, A_{/C} \rangle \in \Delta_{/C}$, defined from the transition $\langle \sigma, G, A \rangle \in \Delta$, can be fired from a state $\langle \vec{v}, obs \rangle \in \mathcal{D}_{V/C}$ in the controlled system if and only if (i) $\langle \sigma, G, A \rangle$ can be fired from \vec{v} in \mathcal{T} , (ii) \vec{v} is compatible with obs , and (iii) σ is not forbidden by the controller \mathcal{C} in obs .
- **Update functions:** The update function $A_{/C}$ of the transition $\langle \sigma, G_{/C}, A_{/C} \rangle \in \Delta_{/C}$, defined from the transition $\langle \sigma, G, A \rangle \in \Delta$, allows the controlled system to reach the state $\langle \vec{v}_2, obs_2 \rangle \in \mathcal{D}_{V/C}$ from a $\langle \vec{v}_1, obs_1 \rangle \in \mathcal{D}_{V/C}$ if and only if (i) the update function A allows the system to reach \vec{v}_2 from \vec{v}_1 , and (ii) \vec{v}_2 is compatible with obs_2 . These update functions $A_{/C}$ satisfy the following property: $(\langle \vec{v}_2, obs_2 \rangle \in A_{/C}(\langle \vec{v}_1, obs_1 \rangle)) \Rightarrow (\forall obs_3 \in M(\vec{v}_2) : (\langle \vec{v}_2, obs_3 \rangle \in A_{/C}(\langle \vec{v}_1, obs_1 \rangle)))$. This property means that when the system arrives in a state \vec{v}_1 , the controller can receive any observation of \vec{v}_1 .

3.3.4 Definition of the Control Problems

In this section, we are interested in three distinct versions of the state avoidance control problem that consists in preventing the system from reaching some particular states, either because some properties are not satisfied in these states (e.g., the states where two state variables are equal or more generally the states in which some particular state predicates are satisfied) or because they are deadlocking or blocking states. We also prove some properties regarding these problems.

Control Problems. The first problem is defined as follows:

Problem 3.1 (Basic State Avoidance Control Problem)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the *basic state avoidance control problem* (*basic problem* for short) consists in computing a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ such that (i) $\text{Reachable}_{\Delta_{/C}}^{\mathcal{T}_{/C}}((\Theta_0)_{/C}) \neq \emptyset$ and (ii) $\left(\text{Reachable}_{\Delta_{/C}}^{\mathcal{T}_{/C}}((\Theta_0)_{/C}) \right]_{[1]} \cap Bad = \emptyset$.

We recall (see section 1.1) that the projection $\text{Reachable}_{\Delta/c}^{\mathcal{T}/c}((\Theta_0)/c) \downarrow_{[1]} = \{\vec{v} \mid \exists \langle \vec{v}, obs \rangle \in \text{Reachable}_{\Delta/c}^{\mathcal{T}/c}((\Theta_0)/c)\}$. Therefore, the basic problem consists in synthesizing a controller which is (i) *non-trivial* (i.e., the behavior of the resulting controlled system is not empty) and (ii) *valid* (i.e., it prevents the system from reaching a forbidden state).

A solution to the basic problem does not ensure that the controlled system is deadlock free (i.e., it is not ensured that the controlled system has always the possibility to make a move) or non-blocking (i.e., it is not ensured that a marked state can always be reached in the controlled system). Since most of the time a deadlocking system or a blocking system cannot be tolerated, we extend our problem to these cases:

Problem 3.2 (Deadlock Free State Avoidance Control Problem)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the *deadlock free state avoidance control problem* (*deadlock free problem* for short) consists in computing a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ such that (i) $\text{Reachable}_{\Delta/c}^{\mathcal{T}/c}((\Theta_0)/c) \neq \emptyset$, (ii) $\left(\text{Reachable}_{\Delta/c}^{\mathcal{T}/c}((\Theta_0)/c) \downarrow_{[1]} \right) \cap Bad = \emptyset$, and (iii) \mathcal{T}/\mathcal{C} is deadlock free.

In other words, the controller must be (i) *non-trivial* and (ii) *valid*, and (iii) it must define a *deadlock free* controlled system.

Problem 3.3 (Non-blocking State Avoidance Control Problem)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the *non-blocking state avoidance control problem* (*non-blocking problem* for short) consists in computing a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ such that (i) $\text{Reachable}_{\Delta/c}^{\mathcal{T}/c}((\Theta_0)/c) \neq \emptyset$, (ii) $\left(\text{Reachable}_{\Delta/c}^{\mathcal{T}/c}((\Theta_0)/c) \downarrow_{[1]} \right) \cap Bad = \emptyset$, and (iii) \mathcal{T}/\mathcal{C} is non-blocking.

In other words, the controller must be (i) *non-trivial* and (ii) *valid*, and (iii) it must define a *non-blocking* controlled system.

Maximal Controller. These problems can be solved by several controllers and we would like to determine the *best* (or *maximal*) solutions. For that, let us consider a relation \preceq , which provides a comparison criterion among the controllers. The concept of \preceq -*maximal controller* then allows us then to define the best solutions (w.r.t. the criterion \preceq) to these problems:

Definition 3.11 (\preceq -Maximal Controller)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a partial order \preceq defining a relation between the controllers, we say that a controller \mathcal{C}_1 is \preceq -*maximal* if and only if, for each controller \mathcal{C}_2 , either \mathcal{C}_1 and \mathcal{C}_2 are incomparable (w.r.t. the relation \preceq), or $\mathcal{C}_2 \preceq \mathcal{C}_1$.

In the literature, there exist several permissiveness criteria. We can, for example, mention:

- the \preceq_{ℓ_g} criterion (resp. \preceq_{ℓ_m} criterion): if the language generated (resp. marked) by the controlled system defined by the controller \mathcal{C}_1 is included in the language generated (resp. marked) by the controlled system defined by the controller \mathcal{C}_2 , then $\mathcal{C}_1 \preceq_{\ell_g} \mathcal{C}_2$ (resp. $\mathcal{C}_1 \preceq_{\ell_m} \mathcal{C}_2$).
- the \preceq_s criterion: if, for each observation obs , the set of actions allowed by the controller \mathcal{C}_1 in obs is included in the set of actions allowed by the controller \mathcal{C}_2 in obs , then $\mathcal{C}_1 \preceq_s \mathcal{C}_2$.
- the \preceq_p criterion: if the set of reachable states in the controlled system defined by the controller \mathcal{C}_1 , is included in the set of reachable states in the controlled system defined by the controller \mathcal{C}_2 , then $\mathcal{C}_1 \preceq_p \mathcal{C}_2$.

In this thesis, we use the *permissiveness criterion* \preceq_p to compare the quality of the different solutions of a problem. It is formally defined as follows:

Definition 3.12 (Permissiveness)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ and an observer $\langle Obs, M \rangle$, a controller $\mathcal{C}_1 = \langle \mathcal{S}_1, E_1 \rangle$ is more *permissive* than a controller $\mathcal{C}_2 = \langle \mathcal{S}_2, E_2 \rangle$ (denoted by $\mathcal{C}_2 \preceq_p \mathcal{C}_1$) if and only if $\text{Reachable}_{\Delta/\mathcal{C}_2}^{\mathcal{T}/\mathcal{C}_2}((\Theta_0)_{/\mathcal{C}_2}) \subseteq \text{Reachable}_{\Delta/\mathcal{C}_1}^{\mathcal{T}/\mathcal{C}_1}((\Theta_0)_{/\mathcal{C}_1})$. When the inclusion is strict, we say that \mathcal{C}_1 is *strictly more permissive* than \mathcal{C}_2 (denoted by $\mathcal{C}_2 \prec_p \mathcal{C}_1$).

Indeed, we are interested in the state avoidance control problem, where the aim is to prevent the system from reaching a set Bad of forbidden states. The *best* controller that can be computed is the one which allows the controlled system to exactly reach the set of good states Bad^c (as a reminder, $Bad^c = \mathcal{D}_V \setminus Bad$). When such a controller does not exist, we would like to compute a controller such that the resulting controlled system achieves the *largest* possible subset of Bad^c . Therefore, it is more appropriate to define our comparison criterion w.r.t. the set of states that are reachable in the controlled system. We can also note that

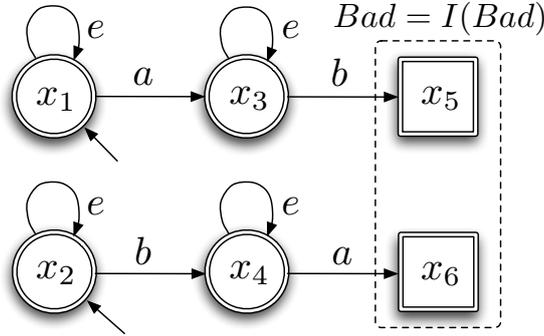


Figure 3.5 - System with two \preceq_p -maximal controllers.

two controlled systems with the same set of reachable states can have different enabled transitions⁷.

We have the following property:

Proposition 3.3

There does not always exist a unique \preceq_p -maximal controller solving the basic (resp. deadlock free and non-blocking) problem.

Proof

We show that the system⁸ depicted in Figure 3.5 admits two \preceq_p -maximal controllers as solution. In this system, (i) the set of states $X = \{x_i \mid i \in [1, 6]\}$, (ii) the set of initial states $X_0 = \{x_1, x_2\}$, (iii) the set of marked states $X_m = X$, and (iv) all transitions are controllable.

Moreover, the set $Bad = \{x_5, x_6\}$ (a *bad state* is represented by a *square*) and the observer $\langle Obs, M \rangle$ is defined as follows: $\mathcal{D}_{Obs} = \{obs_1, obs_2, obs_3\}$ and the mask M is defined in the following way:

$$M(x) = \begin{cases} obs_1 & \text{if } x \in \{x_1, x_4\} \\ obs_2 & \text{if } x \in \{x_2, x_3\} \\ obs_3 & \text{if } x \in \{x_5, x_6\} \end{cases}$$

There are two \preceq_p -maximal controllers which solve the basic (resp. deadlock free and non-blocking) problem:

⁷We could have used an extended definition of permissiveness where the inclusion of the actions, that can be fired from the reachable states, is also taken into account, when two controlled systems have the same set of reachable states.

- the controller \mathcal{C}_1 which forbids the action a in obs_1 :
 $\text{Reachable}_{\Delta/\mathcal{C}_1}^{\mathcal{T}/\mathcal{C}_1}((\Theta_0)_{/\mathcal{C}_1}) = \{\langle x_1, obs_1 \rangle, \langle x_2, obs_2 \rangle, \langle x_4, obs_1 \rangle\}$.
- the controller \mathcal{C}_2 which forbids the action b in obs_2 :
 $\text{Reachable}_{\Delta/\mathcal{C}_2}^{\mathcal{T}/\mathcal{C}_2}((\Theta_0)_{/\mathcal{C}_2}) = \{\langle x_1, obs_1 \rangle, \langle x_2, obs_2 \rangle, \langle x_3, obs_2 \rangle\}$.

These controllers \mathcal{C}_1 and \mathcal{C}_2 are also maximal w.r.t. the \preceq_{ℓ_g} , \preceq_{ℓ_m} and \preceq_s criteria and thus Proposition 3.3 also holds for the concepts of \preceq_{ℓ_g} , \preceq_{ℓ_m} and \preceq_s -maximal controllers.

Undecidability Results. Now, we prove, for any notion of \preceq -maximality, that we cannot define, for the basic, deadlock free and non-blocking problems, *effective* algorithms (i.e., which always terminate) which compute \preceq -maximal controllers. For that, we show that the related decision problems are *undecidable* (see Proposition 3.8) and we use the four following results (Propositions 3.4, 3.5, 3.6 and 3.7):

Proposition 3.4 ([HMR05])

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ and a state $\vec{v} \in \mathcal{D}_V$, the *reachability decision problem*, which consists in deciding if the state \vec{v} is reachable in \mathcal{T} , is undecidable.

Proposition 3.5

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the *basic decision problem*, which consists in deciding if there exists a controller solving the basic problem, is undecidable.

Proof

The proof consists in a reduction from the reachability decision problem (see Proposition 3.4) to the basic decision problem. For that, we build an instance of the basic decision problem from an instance of the reachability decision problem in such a way that there exists a controller solving the basic problem if and only if the state \vec{v} is not reachable in the STS \mathcal{T} .

⁸Automata are particular cases of STS. When the system to be controlled is sufficiently simple, we represent it directly by the corresponding automaton.

Polynomial transformation from the reachability decision problem to the basic decision problem: Let $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ and $\vec{v} \in \mathcal{D}_V$ be the elements defining an instance of the reachability decision problem. The instance of the basic decision problem is then defined by:

- the system $\mathcal{T}' = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$. All actions of Σ are uncontrollable.
- the observer $\langle Obs, M \rangle$ where $Obs = V$ and $M = \text{Id}_{\mathcal{D}_V}$.
- the set $Bad = \{\vec{v}\}$

Correctness of the reduction: Since all actions are uncontrollable, the only controller \mathcal{C} , that can be defined, is the one which forbids no action and this controller is valid if and only if $\text{Reachable}_{\Delta}^{\mathcal{T}}(\Theta_0) \cap Bad = \emptyset$. Therefore, there exists a controller solving the basic problem if and only if $\vec{v} \notin \text{Reachable}_{\Delta}^{\mathcal{T}}(\Theta_0)$ (i.e., \vec{v} is not reachable in \mathcal{T}).

Proposition 3.6

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the *deadlock free decision problem*, which consists in deciding if there exists a controller solving the deadlock free problem, is undecidable.

Proof

The proof consists in a reduction from the basic decision problem to the deadlock free decision problem. For that, we build an instance of the deadlock free decision problem from an instance of the basic decision problem in such a way that a controller \mathcal{C} is a solution of the basic problem if and only if it is a solution of the deadlock free problem.

Polynomial transformation from the basic decision problem to the deadlock free decision problem: Let $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, $\langle Obs, M \rangle$ and Bad be the elements defining an instance of the basic decision problem. The instance of the deadlock free decision problem is then defined by $\mathcal{T}' = \langle V, \Theta_0, \Theta_m, \Sigma', \Delta' \rangle$, $\langle Obs, M \rangle$ and Bad where:

- $\Sigma' = \Sigma \uplus \{\tau\}$. The new action τ does not belong to Σ and is uncontrollable.
- $\Delta' = \Delta \uplus \{\langle \tau, \text{TT}_{\mathcal{D}_V}, \text{Id}_{\mathcal{D}_V} \rangle\}$. This new transition can be fired from any state $\vec{v} \in \mathcal{D}_V$ and loops on \vec{v} .

Correctness of the reduction: The system \mathcal{T}' is deadlock free, because the uncontrollable transition $\langle \tau, \mathbb{T}_{\mathcal{D}_V}, \text{ld}_{\mathcal{D}_V} \rangle$ can be fired from any state of this system. Moreover, given a controller \mathcal{C} , the controlled system $\mathcal{T}'_{\mathcal{C}}$ is always deadlock free, because the transition $\langle \tau, \mathbb{T}_{\mathcal{D}_V}, \text{ld}_{\mathcal{D}_V} \rangle$ cannot be forbidden by \mathcal{C} . Therefore, a controller \mathcal{C} is a solution of the deadlock free problem if and only if it is a solution of the basic problem.

Proposition 3.7

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle \text{Obs}, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the *non-blocking decision problem*, which consists in deciding if there exists a controller solving the non-blocking problem, is undecidable.

Proof

The proof consists in a reduction from the basic decision problem to the non-blocking decision problem. For that, we build an instance of the non-blocking decision problem from an instance of the basic decision problem in such a way that a controller \mathcal{C} is a solution of the basic problem if and only if it is a solution of the non-blocking problem.

Polynomial transformation from the basic decision problem to the non-blocking decision problem: Let $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, $\langle \text{Obs}, M \rangle$ and Bad be the elements defining an instance of the basic decision problem. The instance of the non-blocking decision problem is then defined by $\mathcal{T}' = \langle V, \Theta_0, \Theta'_m, \Sigma, \Delta \rangle$, $\langle \text{Obs}, M \rangle$ and Bad where $\Theta'_m = \mathcal{D}_V$.

Correctness of the reduction: The system \mathcal{T}' is non-blocking, because all states of \mathcal{T}' are marked. Moreover, given a controller \mathcal{C} , the controlled system $\mathcal{T}'_{\mathcal{C}}$ is also non-blocking. Therefore, a controller \mathcal{C} is a solution of the non-blocking problem if and only if it is a solution of the basic problem.

With the previous three properties, we can prove the following result of undecidability:

Proposition 3.8

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle \text{Obs}, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the *maximal basic* (resp. *deadlock free* and *non-blocking*) *decision problem*, which consists in deciding if the basic (resp. deadlock free and non-blocking) problem admits at least one \preceq -maximal controller as solution, is undecidable.

Proof

The proof is straightforward from Propositions 3.5, 3.6 and 3.7.

To overcome this negative result, we propose an approach, based on abstract interpretation, which approximates some computations to ensure the termination of the algorithms. Our aim, in the next sections, is to prove that this approach gives correct controllers and to experimentally show that these solutions are of good quality.

3.4 Computation by Means of Abstract Interpretation of a Memoryless Controller for the Basic Problem

In this section, we present our method, based on abstract interpretation, which synthesizes memoryless controllers for the basic problem. First, in section 3.4.1, we present a *semi-algorithm* (i.e., which does not always terminate) which computes a memoryless controller for the basic problem. Next, in section 3.4.2, we explain how to extend it to obtain, at the price of some overapproximations, an *effective algorithm* (i.e., which always terminates). Finally, in section 3.4.3, we theoretically evaluate the permissiveness of our solutions and we explain how it can be improved.

3.4.1 Semi-algorithm for the Basic Problem

Our algorithm, which synthesizes controllers for the basic problem, is composed of two parts:

- we compute, using a fixpoint equation, the set $I(Bad)$ of states that can lead to Bad by triggering only uncontrollable transitions.
- we forbid, in each state \vec{v} of the system, all controllable actions that can lead to $I(Bad)$ from this state. Moreover, to take into account the partial observation, we must also forbid these actions in the states that are undistinguishable from \vec{v} .

These two steps are formalized as follows:

Computation of $I(Bad)$. The set $I(Bad)$ of states leading uncontrollably to Bad is given by $\text{Coreach}_{\Delta_{uc}}^T(Bad)$ which, as a reminder, is defined by $\text{Coreach}_{\Delta_{uc}}^T(Bad) = \bigcup_{n \geq 0} (\text{Pre}_{\Delta_{uc}}^T)^n(Bad)$. This set of states cannot always be computed, because the coreachability is *undecidable* in the model of STS, and, in section 3.4.2, we will use abstract interpretation techniques to overapproximate it. However, to compute this overapproximation, we must characterize the set $I(Bad)$ by a fixpoint equation. This equation is defined as follows over the complete lattice $\langle 2^{\mathcal{D}^V}, \subseteq, \cup, \cap, \mathcal{D}^V, \emptyset \rangle$:

$$I(Bad) \stackrel{\text{def}}{=} \text{lfp}^{\subseteq}(\lambda B. Bad \cup \text{Pre}_{\Delta_{uc}}^T(B)) \quad (3.1)$$

Since the function $\lambda B. Bad \cup \text{Pre}_{\Delta_{uc}}^T(B)$ is monotonic, the Knaster-Tarski's theorem [Tar55] ensures that the least fixpoint of this function exists and the following proposition proves that this fixpoint equation gives the set of states leading uncontrollably to Bad :

Proposition 3.9

The function $\lambda B. Bad \cup \text{Pre}_{\Delta_{uc}}^T(B)$ defined over the complete lattice $\langle 2^{\mathcal{D}^V}, \subseteq, \cup, \cap, \mathcal{D}^V, \emptyset \rangle$ is such that $\text{lfp}^{\subseteq}(\lambda B. Bad \cup \text{Pre}_{\Delta_{uc}}^T(B)) = \bigcup_{n \geq 0} (\text{Pre}_{\Delta_{uc}}^T)^n(Bad)$.

Proof

By the Knaster-Tarski and Kleene's theorems, it suffices to show that the function $\text{Pre}_{\Delta_{uc}}^T$ is continuous to prove this proposition. We must thus prove that, for each increasing sequence of sets $B_1 \subseteq B_2 \subseteq \dots \subseteq B_n \subseteq \dots$ (where $\forall i \geq 1 : B_i \in 2^{\mathcal{D}^V}$), the equality $\text{Pre}_{\Delta_{uc}}^T(\bigcup_{i \geq 1} B_i) = \bigcup_{i \geq 1} \text{Pre}_{\Delta_{uc}}^T(B_i)$ holds. For that, we first prove that, for each $\delta = \langle \sigma, G, A \rangle \in \Delta$, $\text{Pre}_{\delta}^T(\bigcup_{i \geq 1} B_i) = \bigcup_{i \geq 1} \text{Pre}_{\delta}^T(B_i)$:

$$\begin{aligned} \text{Pre}_{\delta}^T\left(\bigcup_{i \geq 1} B_i\right) &= G \cap A^{-1}\left(\bigcup_{i \geq 1} B_i\right), \text{ by definition of } \text{Pre}_{\delta}^T \\ &= G \cap \left(\bigcup_{i \geq 1} A^{-1}(B_i)\right), \text{ because the function } A^{-1} \text{ is} \\ &\quad \text{continuous by assumption.} \\ &= \bigcup_{i \geq 1} (G \cap A^{-1}(B_i)), \text{ because } a \cap \left(\bigcup_i b_i\right) = \bigcup_i (a \cap b_i). \\ &= \bigcup_{i \geq 1} \text{Pre}_{\delta}^T(B_i) \end{aligned}$$

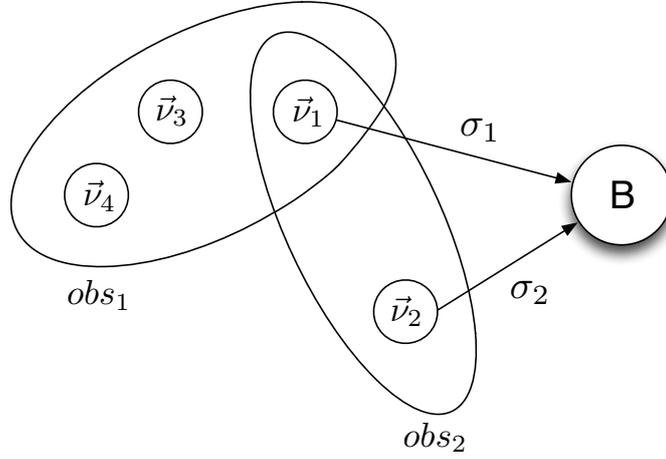


Figure 3.6 - Computation of $\mathcal{F}(\sigma_i, B)$ (for $i = 1, 2$).

Next, we prove that, for each $D \subseteq \Delta$, $\text{Pre}_D^T(\bigcup_{i \geq 1} B_i) = \bigcup_{i \geq 1} \text{Pre}_D^T(B_i)$:

$$\begin{aligned}
 \text{Pre}_D^T\left(\bigcup_{i \geq 1} B_i\right) &= \bigcup_{\delta \in D} \text{Pre}_\delta^T\left(\bigcup_{i \geq 1} B_i\right), \text{ by definition of } \text{Pre}_D^T \\
 &= \bigcup_{\delta \in D} \left(\bigcup_{i \geq 1} \text{Pre}_\delta^T(B_i)\right), \text{ because } \text{Pre}_\delta^T \text{ is continuous.} \\
 &= \bigcup_{i \geq 1} \left(\bigcup_{\delta \in D} \text{Pre}_\delta^T(B_i)\right) \\
 &= \bigcup_{i \geq 1} \text{Pre}_D^T(B_i)
 \end{aligned}$$

Computation of the Controller \mathcal{C} . To compute this controller, we first define the function $\mathcal{F} : \Sigma \times 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_{Obs}}$ which gives, for each action $\sigma \in \Sigma$ and set $B \subseteq \mathcal{D}_V$ of states to be forbidden, the set $\mathcal{F}(\sigma, B)$ of observations, in which the action σ must be forbidden. More precisely, this set $\mathcal{F}(\sigma, B)$ corresponds to the greatest set \mathcal{O} of observations such that, for each observation $obs \in \mathcal{O}$, there

exists a state $\vec{v} \in \mathcal{D}_V$ which is compatible with obs and which leads to B by a transition labeled by σ :

$$\mathcal{F}(\sigma, B) \stackrel{\text{def}}{=} \begin{cases} M(\text{Pre}_\sigma^T(B) \setminus B) & \text{if } \sigma \in \Sigma_c \\ \emptyset & \text{otherwise} \end{cases} \quad (3.2)$$

Example 3.4

Figure 3.6 illustrates this computation. Let us suppose that \vec{v}_1 leads to B by the controllable action σ_1 and that \vec{v}_2 leads to B by the controllable action σ_2 . The observations $obs_1, obs_2 \in \mathcal{F}(\sigma_1, B)$, because \vec{v}_1 is compatible with obs_1 and obs_2 , and the observation $obs_2 \in \mathcal{F}(\sigma_2, B)$, because \vec{v}_2 is compatible with obs_2 .

Next, the controller \mathcal{C} is defined in the following way:

$$\mathcal{C} \stackrel{\text{def}}{=} \langle \mathcal{S}, E \rangle \quad (3.3)$$

with the elements \mathcal{S} and E defined as follows:

- the *supervisory function* \mathcal{S} is given, for each observation $obs \in \mathcal{D}_{Obs}$, by:

$$\mathcal{S}(obs) \stackrel{\text{def}}{=} \{\sigma \in \Sigma_c \mid obs \in \mathcal{F}(\sigma, I(Bad))\} \quad (3.4)$$

It means that σ is forbidden in obs if $I(Bad)$ is reachable from a state compatible with obs by a transition labeled by σ .

- the set $E \stackrel{\text{def}}{=} I(Bad)$. It prevents the controlled system from beginning its execution in a state which can lead to Bad by a sequence of uncontrollable transitions.

The set $I(Bad)$ and the function \mathcal{F} are computed *offline*. Next, during the execution of the system, when the controller receives an observation $obs \in \mathcal{D}_{Obs}$, it computes *online* the set $\mathcal{S}(obs)$ of actions to be forbidden (the computation of this set is based on the function \mathcal{F}). Since Σ is finite, $\mathcal{S}(obs)$ is computable when $I(Bad)$ is computed.

Example 3.5

For the system depicted in Figure 3.1, the controller must ensure, during the phase of consumption of the parts X , the presence of a minimal number of parts of this kind: $Bad = \{\langle CX, x, x', y, y' \rangle \mid (x \leq 1000) \wedge (y \in [0, 2])\}$.

The controller has a partial observation of the system, which is modeled by the mask M defined in Example 3.3 i.e., for each state $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, the set of states which are undistinguishable from \vec{v} , is given by $\{\langle \ell, x_1, x', y, y' \rangle \mid x_1 \in [x - 1, x + 1]\}$. The controllable (resp. uncontrollable) transitions are drawn in plain (resp. dashed) lines. The computation of the set $I(Bad)$ gives:

$$I(Bad) = Bad \cup \{ \langle CX, x, x', y, y' \rangle \mid [(x \leq 1001) \wedge (y \in [1, 2])] \vee [(x \leq 1002) \wedge (y = 2)] \}$$

and the computation of the function \mathcal{F} gives:

$$\mathcal{F}(\sigma, I(Bad)) = \begin{cases} M(\{ \langle PX, x, x', y, y' \rangle \mid x \leq 1002 \}) & \text{if } \sigma = Stop_prod \\ \emptyset & \text{otherwise} \end{cases}$$

Therefore, the supervisory function \mathcal{S} forbids the action *Stop_prod* when the system is in a state $\vec{v} \in \{ \langle PX, x, x', y, y' \rangle \mid x \leq 1003 \}$ and it forbids no action in the other states of the system. One can note that the controller computed for this system is \preceq_p -maximal.

The following property proves that our algorithm synthesizes correct controllers for the basic problem:

Proposition 3.10

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, defined by (3.3), solves the basic problem (when $\Theta_0 \not\subseteq E$).

Proof

We prove by induction on the length n of the sequences of transitions (these sequences begin in an initial state) that $(\text{Reachable}_{\Delta/\mathcal{C}}^{\mathcal{T}/\mathcal{C}}((\Theta_0)_{/\mathcal{C}})]_{[1]} \cap I(Bad) = \emptyset$. Since $Bad \subseteq I(Bad)$, this property implies that $(\text{Reachable}_{\Delta/\mathcal{C}}^{\mathcal{T}/\mathcal{C}}((\Theta_0)_{/\mathcal{C}})]_{[1]} \cap Bad = \emptyset$.

- *Base case* ($n = 0$): the execution of the controlled system \mathcal{T}/\mathcal{C} starts in a state which does not belong to $I(Bad)$, since $(\Theta_0)_{/\mathcal{C}}|_{[1]} = \Theta_0 \setminus E = \Theta_0 \setminus I(Bad)$.

- *Induction step:* we suppose that the proposition holds for the sequences of transitions of length less than or equal to n and we prove that this property remains true for the sequences of transitions of length $n+1$. By induction hypothesis, each state \vec{v}_1 reachable by a sequence of transitions of length n does not belong to $I(Bad)$ and we show that each transition $\delta = \langle \sigma, G, A \rangle \in \Delta$, which can lead to a state $\vec{v}_2 \in I(Bad)$ from this state \vec{v}_1 in the system \mathcal{T} , cannot be fired from \vec{v}_1 in the controlled system \mathcal{T}/\mathcal{C} . For that, we consider two cases :
 - if $\delta \in \Delta_c$, then this transition cannot be fired from \vec{v}_1 in the system \mathcal{T}/\mathcal{C} , since $\sigma \in \mathcal{S}(obs)$ ($\forall obs \in M(\vec{v}_1)$) by (3.2) and (3.4).
 - if $\delta \in \Delta_{uc}$, then $\vec{v}_2 \in I(Bad)$ (by (3.1)), which is impossible by hypothesis.

Hence, in the controlled system, the forbidden state \vec{v}_2 cannot be reached from the state \vec{v}_1 by the transition δ .

The feedback interaction between a system $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ to be controlled and a controller \mathcal{C} (see Figure 3.4) gives the controlled system $\mathcal{T}/\mathcal{C} = \langle V/\mathcal{C}, (\Theta_0)/\mathcal{C}, (\Theta_m)/\mathcal{C}, \Sigma, \Delta/\mathcal{C} \rangle$ given in Definition 3.10. In some cases, it can be interesting to compute this controlled system. Unfortunately, when the mask M is a *covering* of the state space this controlled system cannot always be computed. Indeed, in our tool, we perform the computations by manipulating symbolically sets of states to avoid enumerating these sets. But, this approach cannot be used to compute the transitions $\langle \sigma, G/\mathcal{C}, A/\mathcal{C} \rangle$ (defined from $\langle \sigma, G, A \rangle$) of the controlled system, because the computation of G/\mathcal{C} requires to associate the observations of $M(\vec{v})$, which are not forbidden by the controller, with each state $\vec{v} \in G$. Obviously, this enumeration is not always possible, because the state space can be infinite.

However, when the mask M corresponds to a partition of the state space, the definition of the controlled system is simplified (the observations can be omitted in the definition, because each state has exactly an observation), which makes possible its computation. Indeed, the simplified definition of the controlled system is the following:

Definition 3.13 (Controlled System when the Mask is a Partition)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ (where M corresponds to a *partition* of \mathcal{D}_V), and a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, the system \mathcal{T} controlled by \mathcal{C} , denoted by $\mathcal{T}_{/\mathcal{C}}$, is an STS defined by $\mathcal{T}_{/\mathcal{C}} \stackrel{\text{def}}{=} \langle V, (\Theta_0)_{/\mathcal{C}}, \Theta_m, \Sigma, \Delta_{/\mathcal{C}} \rangle$, where:

- $(\Theta_0)_{/\mathcal{C}} \stackrel{\text{def}}{=} \{\vec{v} \mid \vec{v} \in (\Theta_0 \setminus E)\}$ is the set of initial states.
- $\Delta_{/\mathcal{C}}$ is defined from Δ as follows: for each transition $\langle \sigma, G, A \rangle \in \Delta$, we define a new transition $\langle \sigma, G_{/\mathcal{C}}, A \rangle \stackrel{\text{def}}{\in} \Delta_{/\mathcal{C}}$, where $G_{/\mathcal{C}} \stackrel{\text{def}}{=} \{\vec{v} \mid (\vec{v} \in G) \wedge (\sigma \notin \mathcal{S}(M(\vec{v})))\}$.

This controlled system $\mathcal{T}_{/\mathcal{C}} = \langle V, (\Theta_0)_{/\mathcal{C}}, \Theta_m, \Sigma, \Delta_{/\mathcal{C}} \rangle$ can be computed for our controller $\mathcal{C} = \langle \mathcal{S}, I(Bad) \rangle$ (defined by (3.3)) since (i) $(\Theta_0)_{/\mathcal{C}}$ is obtained by computing the set operation $\Theta_0 \setminus I(Bad)$ and (ii) the restricted guard $G_{/\mathcal{C}}$ of any transition $\langle \sigma, G_{/\mathcal{C}}, A \rangle \in \Delta_{/\mathcal{C}}$ (defined from the transition $\langle \sigma, G, A \rangle \in \Delta$) is obtained by computing the set operation $G \setminus \{M^{-1}(\mathcal{F}(\sigma, I(Bad)))\}$.

3.4.2 Effective Algorithm for the Basic Problem

The algorithm described in the previous section requires to compute the fixpoint $I(Bad)$, but this equation cannot always be computed for undecidability (or complexity) reasons. We overcome this obstacle by using *abstract interpretation* techniques (see e.g. [CC77, HPR97, Jea03]) in order to compute an overapproximation of the fixpoint $I(Bad)$ in a finite time. Since our algorithm then uses an overapproximation of $I(Bad)$, the synthesized controllers remain correct, but they can be less permissive.

We use abstract interpretation by means of the *representation framework* (see section 1.4.2) instead of the *Galois connection framework* (see section 1.4.1), because the abstract lattice that we use (i.e., the lattice of convex polyhedra) is not complete. The lattice of convex polyhedra constitutes one of the most used abstract lattice and it has been used successfully in many applications (e.g., for the analysis and verification of linear hybrid automata [HHWt97], for detecting buffer overflows in C [DRS01],...). We define the elements which constitute the representation framework as follows.

Definition of the Elements of the Representation Framework. The *con-*

crete lattice is defined by the 6-tuples $\langle 2^{D_V}, \sqsubseteq, \cup, \cap, \mathcal{D}_V, \emptyset \rangle$ (as a reminder, the tuple V gives the variables of the system to be controlled) and the used representation (see Definition 1.18) is defined by the 3-tuple $\langle \Lambda, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle, \gamma, \nabla$.
 (i) The abstract lattice $\langle \Lambda, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$ corresponds to the lattice of *convex polyhedra*; the concepts of *convex polyhedron* and of *lattice of convex polyhedra* are defined as follows:

Definition 3.14 (Convex Polyhedron)

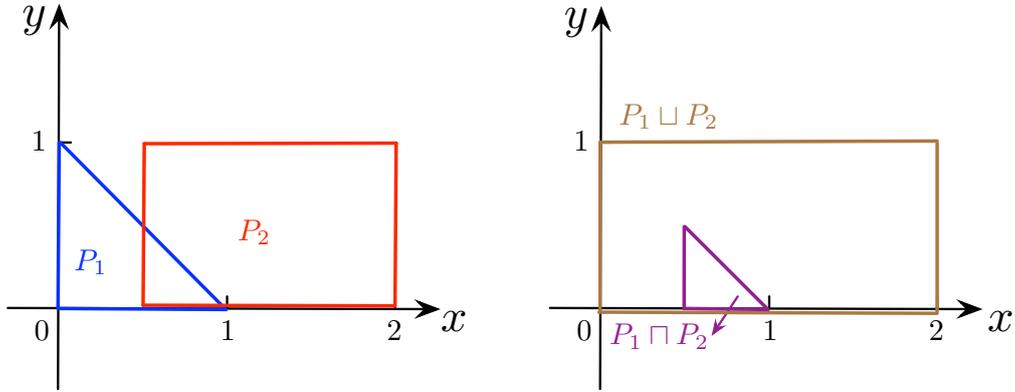
A *convex polyhedron* on the tuple of variables $V = \langle v_1, \dots, v_n \rangle$ is defined by a conjunction of k linear constraints $\bigwedge_{1 \leq i \leq k} a_{i,1}.v_1 + \dots + a_{i,n}.v_n \leq c_i$, where (i) $a_{i,1}, \dots, a_{i,n} \in \mathbb{Q}$ ($\forall i \in [1, k]$) are coefficients and (ii) $c_i \in \mathbb{Q}$ ($\forall i \in [1, n]$) is a constant.

Definition 3.15 (Lattice of Convex Polyhedra)

The *lattice of convex polyhedra* is defined by the 6-tuples $\langle \Lambda, \sqsubseteq, \sqcup, \sqcap, \top, \perp \rangle$ where:

- Λ is the set of *convex polyhedra* on the tuple of variables V .
- the \sqsubseteq operator is defined by the classical set inclusion operator.
- the \sqcup operator of two convex polyhedra P_1 and P_2 is defined by the *convex hull* of P_1 and P_2 , which gives the least convex polyhedron containing both P_1 and P_2 . The convex hull is used as an overapproximation of the set union, because this operation is not closed for convex polyhedra.
- the \sqcap operator of two convex polyhedra is defined by the classical set intersection operator. This operation is closed for convex polyhedra.
- \top is defined by the convex polyhedron which represents the entire state space.
- \perp is the empty set.

(ii) The *concretization function* $\gamma : \Lambda \rightarrow 2^{D_V}$ is defined by the identity function.
 (iii) The widening operator ∇ is the *standard widening* defined in [CH78]. The widening of two convex polyhedra P_1 and P_2 (denoted by $P_1 \nabla P_2$) such that $P_1 \sqsubseteq P_2$ consists in removing from P_1 all constraints which are not satisfied by P_2 . In other words, if between P_1 and P_2 the value of a variable or a linear expression grows, then the widening operator considers that it grows indefinitely

(a) Convex polyhedra P_1 and P_2 .(b) Convex polyhedra $P_1 \cap P_2$ and $P_1 \sqcup P_2$.**Figure 3.7** - An example of \cap and \sqcup operations.

and thus it removes it.

To correctly perform the computations in the abstract lattice, each element $B \subseteq \mathcal{D}_V$ of the concrete lattice must *safely* be transposed into an element $\alpha(B)$ of the abstract lattice i.e., $B \subseteq \gamma(\alpha(B))$ (see Definition 1.18). For that, we use the *abstraction function* $\alpha : 2^{\mathcal{D}_V} \rightarrow \Lambda$ defined as follows: for each set $B \in 2^{\mathcal{D}_V}$, if B corresponds to a polyhedron, then $\alpha(B)$ is defined by the least convex polyhedron which contains B , otherwise $\alpha(B)$ is defined by \top . In this definition, we consider two cases to transpose the elements of the concrete lattice, because when B is not a polyhedron, the least convex polyhedron containing B does not always exist (e.g., when B represents a circle). The abstraction of the elements, which do not correspond to polyhedra, is rough, but this definition of α is satisfactory for our tool, because we only need to abstract polyhedra.

Example 3.6

Let us consider a state space with two integer variables x and y . The convex polyhedron $P_1 = (x \geq 0) \wedge (y \geq 0) \wedge (x + y \leq 1)$ defines a right-angle triangle and the convex polyhedron $P_2 = (x \geq 0.5) \wedge (x \leq 2) \wedge (y \geq 0) \wedge (y \leq 1)$ defines a rectangle (see Figure 3.7(a)).

The intersection of P_1 and P_2 gives the convex polyhedron $P_3 = (x \geq 0.5) \wedge (y \geq 0) \wedge (x + y \leq 1)$ (see Figure 3.7(b)) and the convex hull of P_1 and P_2 gives the convex polyhedron $P_4 = (x \geq 0) \wedge (x \leq 2) \wedge (y \geq 0) \wedge (y \leq 1)$ (see Figure 3.7(b)).

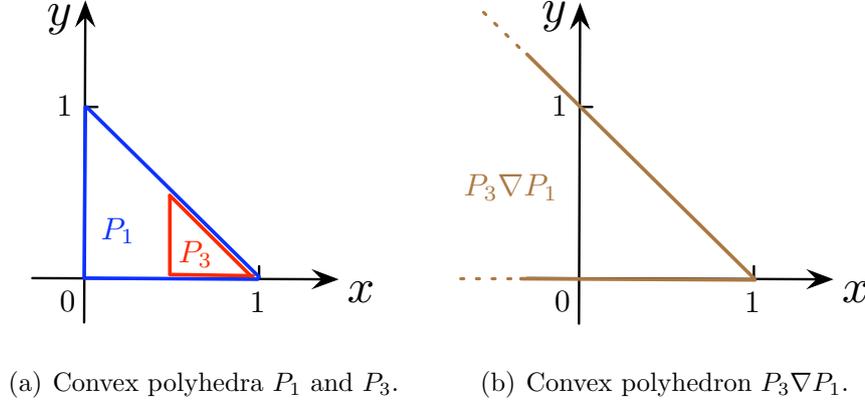


Figure 3.8 - An example of widening operations.

The widening operation $P_3 \nabla P_1$ gives the convex polyhedron $P_5 = (y \geq 0) \wedge (x + y \leq 1)$, because P_1 does not satisfy the linear constraint $x \geq 0.5$ (see Figure 3.8(b)).

Effective Algorithm for the Basic Problem. The effective algorithm is quite similar to the semi-algorithm defined in section 3.4.1: the main differences are that we perform the computations in the abstract lattice and that we compute an overapproximation of $I(Bad)$ to ensure the termination of our algorithm. More precisely, to compute this overapproximation, we use Theorem 1.4 with the instantiation of the representation framework defined above. This theorem requires to transpose the functions used in the computation of $I(Bad)$ into the abstract lattice. First, we perform this transposition for the function $\text{Pre}_D^T : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ (where $D \subseteq \Delta$) and we obtain the function $\text{Pre}_D^{T,\#} : \Lambda \rightarrow \Lambda$ which is defined, for each $\ell \in \Lambda$, by:

$$\text{Pre}_D^{T,\#}(\ell) \stackrel{\text{def}}{=} \bigsqcup_{\delta \in D} \text{Pre}_\delta^{T,\#}(\ell), \text{ where} \tag{3.5}$$

$$\text{Pre}_\delta^{T,\#}(\ell) \stackrel{\text{def}}{=} \alpha(G_\delta \cap A_\delta^{-1}(\gamma(\ell))) \tag{3.6}$$

Next, we transpose the fixpoint equation $I(Bad)$ into the abstract lattice and we obtain:

$$I^\#(Bad) \stackrel{\text{def}}{=} \text{lfp}^\square(\lambda \ell. \alpha(Bad) \sqcup \text{Pre}_{\Delta_{uc}}^{T,\#}(\ell)) \tag{3.7}$$

According to Theorem 1.4, we compute the element a_∞ defined as the limit of

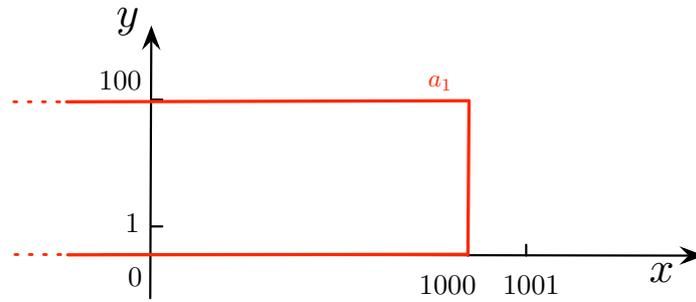
the following sequence:

$$a_i = \begin{cases} \perp & \text{if } i = 0 \\ a_{i-1} \nabla (\alpha(\text{Bad}) \sqcup \text{Pre}_{\Delta_{uc}}^{\mathcal{T}, \#}(a_{i-1})) & \text{if } i > 0 \end{cases} \quad (3.8)$$

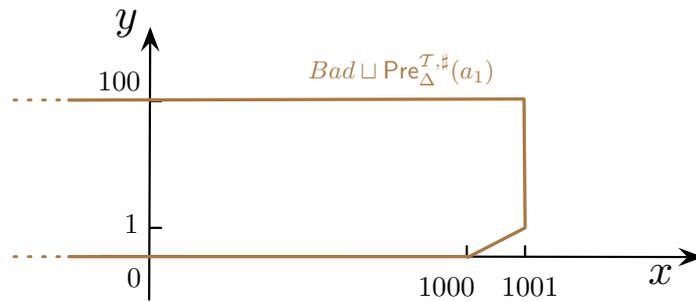
Theorem 1.4 ensures that this element a_∞ is obtained after a finite number of steps and that a_∞ is a post-fixpoint of $I^\#(\text{Bad})$. Moreover, it ensures that $\gamma(a_\infty)$ is a post-fixpoint of $I(\text{Bad})$. Therefore, $I'(\text{Bad}) = \gamma(a_\infty)$ is an overapproximation of $I(\text{Bad})$ (one can note that $\gamma(a_\infty) = a_\infty$, because γ is the identity function). Note that we can use this theorem, because all constraints that it requires are fulfilled.

After computing the overapproximation of $I(\text{Bad})$, we transpose the control function \mathcal{F} into the abstract lattice and we obtain the abstract function $\mathcal{F}^\#$. The computation of the control function may thus require overapproximations. Finally, we define our controller $\mathcal{C} = \langle \mathcal{S}, I'(\text{Bad}) \rangle$ as in (3.3) by using $I'(\text{Bad})$ and $\mathcal{F}^\#$ instead of $I(\text{Bad})$ and \mathcal{F} . The controller, that we obtain, is correct, because $I'(\text{Bad}) \supseteq I(\text{Bad})$. It is worthy notable that the obtained controller can make unreachable more states than necessary, then it can be less permissive.

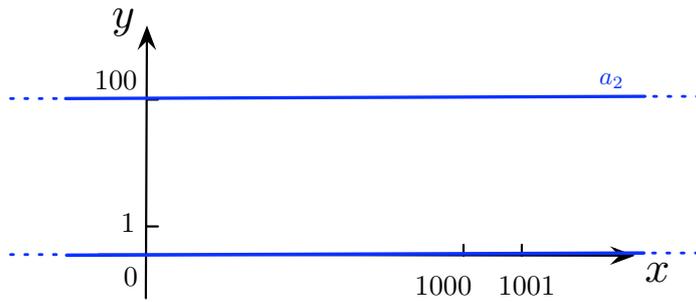
In general, a variant of the fixpoint computation strategy described in (3.8) is used to compute an overapproximation of the least fixpoint of a function. Indeed, in this strategy the widening operator is applied at each iteration, which can give rough results. Abstract interpretation theory suggests to wait several iterations before using for the first time the widening operator and to separate two successive applications of the widening operator from several iterations. In our tool, we use the fixpoint computation strategy defined in [Bou92] and based on these recommendations. This strategy is parametrized by three elements k_i , k_s and k_r . Parameter k_i gives the iteration at which the widening operator must be used for the first time and k_s gives the number of iterations between two successive applications of the widening operator. Therefore, the widening operator is used at the iterations $k_i + j \cdot k_s$ ($\forall j \geq 0$); otherwise, we use the \sqcup . Moreover, a post-fixpoint of a function can be refined by applying this function to this element and the parameter k_r specifies the number of times that this operation must be performed. Increasing the value of the parameters k_i , k_s and k_r generally improves the quality of the overapproximations, but the time complexity to compute them also increases. A trade-off between these two aspects must thus be found. This compromise depends on the considered function, the used representation and the user requirements. In our tool, we use small values for the parameters k_i , k_s and k_r (in general, $k_i, k_s, k_r \in [1, 10]$) and we obtain good results with these values.



(a) Computation of a_1 .



(b) Computation of $Bad \sqcap Pre_{\Delta}^{\mathcal{T};\#}(a_1)$.



(c) Computation of a_2 .

Figure 3.9 - An example of fixpoint computations with the strategy described in (3.8).

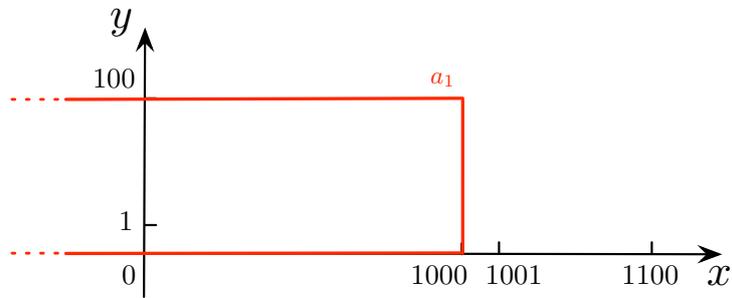
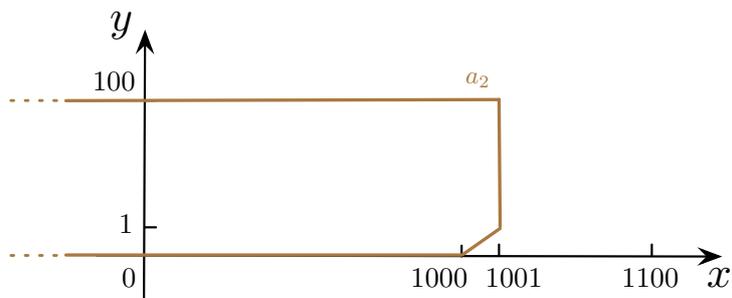
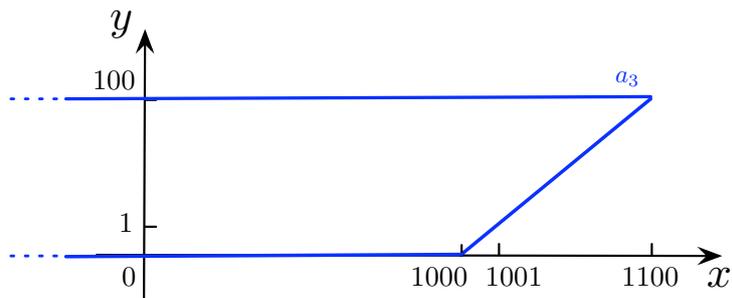
(a) Computation of a_1 .(b) Computation of a_2 .(c) Computation of a_3 .

Figure 3.10 - An example of fixpoint computations with the strategy used in our tool.

Example 3.7

Let us illustrate the effective algorithm by considering the system depicted in Figure 3.1, where the transitions δ_1 and δ_4 are respectively replaced by $\langle \text{Cons}; 0 \leq y \leq 100; x := x - 1, y := y - 1 \rangle$ and $\langle \text{Stop_prod}; \top \top_{\mathcal{D}_V}; y := 100 \rangle$. The set Bad of forbidden states is defined by $\{\langle \text{CX}, x, x', y, y' \rangle \mid (x \leq 1000) \wedge (y \in [0, 100])\}$ and the controller has a full observation of the system.

If we use the fixpoint computation strategy described in (3.8) to compute an overapproximation of $I(Bad)$, we obtain the following sequence of values for the variables x, x', y, y' in the location CX (see Figure 3.9): (i) $a_0 = \perp$, (ii) $a_1 = \perp \nabla ((x \leq 1000) \wedge (y \geq 0) \wedge (y \leq 100)) = (x \leq 1000) \wedge (y \geq 0) \wedge (y \leq 100)$, (iii) $a_2 = a_1 \nabla ((x \leq 1001) \wedge (y \geq 0) \wedge (y \leq 100) \wedge (x - y \leq 1000)) = (y \geq 0) \wedge (y \leq 100)$. Thus, $I'(Bad) = \{\langle \text{CX}, x, x', y, y' \rangle \mid (y \leq 100) \wedge (y \geq 0)\}$ and the controller always forbids the action Stop_prod in the location PX .

If we use the fixpoint computation strategy of our tool with $k_i = k_s = k_r = 3$, we obtain the following sequence of values for the x, x', y, y' in the location CX (see Figure 3.10): (i) $a_0 = \perp$, (ii) $a_1 = \perp \sqcup ((x \leq 1000) \wedge (y \geq 0) \wedge (y \leq 100)) = (x \leq 1000) \wedge (y \geq 0) \wedge (y \leq 100)$, (iii) $a_2 = a_1 \sqcup ((x \leq 1001) \wedge (y \geq 0) \wedge (y \leq 100) \wedge (x - y \leq 1000)) = (x \leq 1001) \wedge (y \geq 0) \wedge (y \leq 100) \wedge (x - y \leq 1000)$, (iv) $a_3 = a_2 \nabla ((x \leq 1002) \wedge (y \geq 0) \wedge (y \leq 100) \wedge (x - y \leq 1000)) = (y \geq 0) \wedge (y \leq 100) \wedge (x - y \leq 1000)$. In this example, the refinement (parameter k_r) of a_3 does not modify the value of this element. Thus, $I'(Bad) = \{\langle \text{CX}, x, x', y, y' \rangle \mid (y \leq 100) \wedge (y \geq 0) \wedge (x - y \leq 1000)\}$ and the controller forbids the action Stop_prod in the location PX when $x \leq 1100$. One can note that this controller is \preceq_p -maximal.

Quality of the Approximations. The method presented in this section always computes a correct controller, but it does not guarantee that this controller is \preceq_p -maximal. The quality of this controller depends on the overapproximation of $I(Bad)$ that we compute. We must thus compute *good* approximations and there are classical techniques to improve the quality of these approximations:

- the choice of the abstract lattice is the main issue. If it is not adapted to the kind of guards or assignments of the STS, the overapproximations are too rough. The practice shows that if the guards are linear constraints, and if the assignment functions are also linear, the lattice, that we use (i.e., the lattice of convex polyhedra [CH78]), gives good results.
- the computation of the fixpoint with the widening operator is also an important issue and it can be improved by several ways: (i) we can use a more

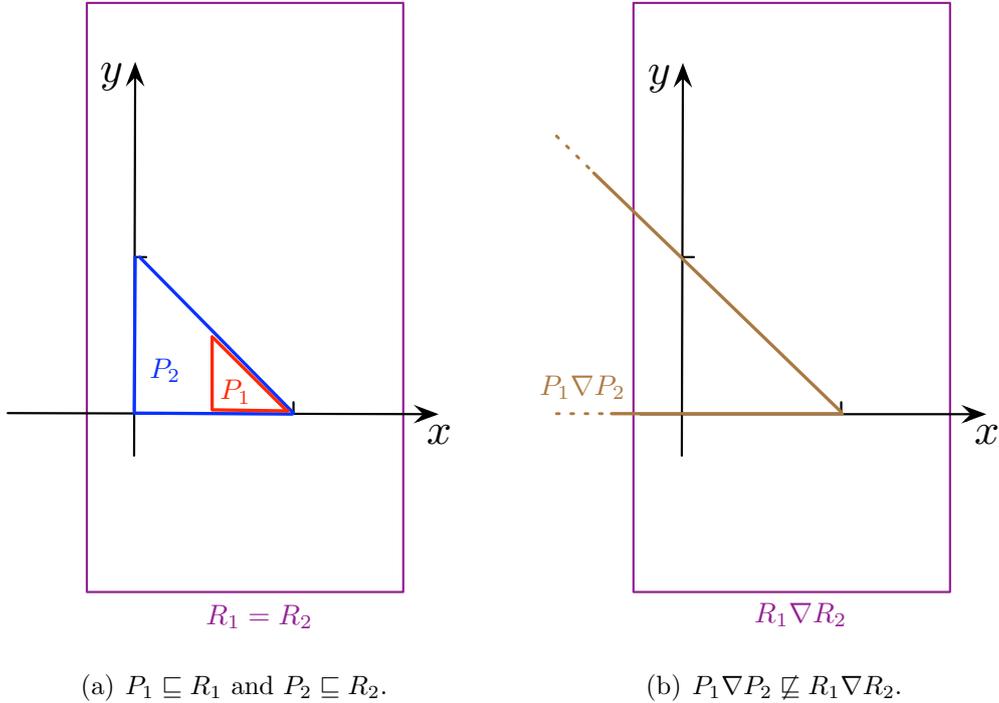


Figure 3.11 - An example which illustrates that using a more precise abstract lattice does not always give better results for the widening.

elaborate widening, like the widening *up to*, instead of the *standard widening* [HPR97], (ii) we can use one of the fixpoint computation strategies defined in [Bou92], (iii) we can refine our abstract lattice (see [Jea03] for more details), and (iv) we can refine the post-fixpoint obtained with the widening operator by using a narrowing operator [CC77]. Roughly, this operation consists in computing a decreasing sequence of post-fixpoints (where the first element is the post-fixpoint computed with the widening operator) in order to improve the overapproximation of the least fixpoint.

In the literature, there are few theoretical results on the quality of abstractions. This lack of results is mainly due to the fact that the widening operator is not monotonic. Indeed, because of this property, an improvement in the elements used in the abstractions does not necessarily imply that the iterative fixpoint computation gives a more precise overapproximation. Therefore, it is difficult to theoretically evaluate the quality of abstractions.

Example 3.8

This example illustrates that using a more precise abstract lattice to represent sets of states does not always give better results in an iterative fixpoint computation. Let us suppose that two sets of states are represented on one hand by the convex polyhedra P_1 and P_2 and on the other hand by the rectangles R_1 and R_2 with $P_1 \sqsubseteq R_1$ and $P_2 \sqsubseteq R_2$ (see Figure 3.11). P_1 and P_2 are thus more precise than R_1 and R_2 , but $P_1 \nabla P_2$ is not more precise than $R_1 \nabla R_2$.

Since it is difficult to theoretically analyze the quality of abstractions, we have implemented our algorithm to evaluate it experimentally. The results are available in section 3.7.

3.4.3 Evaluation and Improvement of the Permissiveness of the Controller

In this section, we compare the permissiveness of our control algorithm with the one of an algorithm defined in the literature [TK98] and we explain how we can modify our algorithm to improve its permissiveness.

Evaluation of the Permissiveness in the Finite Case. To the best of our knowledge, there is no control algorithm which synthesizes controllers with partial observation for the basic problem in the case of infinite state systems. Therefore, we restrict our attention to the case of finite state systems and the most permissive controller, that we found, is the one defined by Takai *et al.* in [TK98] (they assume that the masks are partitions of the state space). The following proposition shows that our algorithm and the one defined in [TK98] have the same permissiveness:

Proposition 3.11

When the basic problem is restricted to the case where the system to be controlled is finite and the mask is a partition of the state space, the controller defined in (3.3) and the one defined in [TK98] have the same permissiveness.

Proof

First, we outline the work defined in [TK98]. The authors propose an algorithm solving the basic problem where (i) the system to be controlled is modeled by a finite deterministic automaton $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$, (ii) the control specification is given by a set $Q \subseteq X$ of good states (i.e., $Q = X \setminus Bad$), and

(iii) the partial observation is modeled by a mask $M : X \rightarrow \mathcal{D}_{Obs}$ corresponding to a partition of X . This algorithm is composed of two steps:

- the computation of the set of *safe* states $Q^\dagger \subseteq Q$ where $Q^\dagger \stackrel{\text{def}}{=} \bigcap_{j=0}^{\infty} Q_j$, with Q_j defined recursively as follows:

$$Q_j \stackrel{\text{def}}{=} \begin{cases} Q & \text{if } j = 0 \\ g(Q_{j-1}) & \text{otherwise} \end{cases}$$

where, for each $Q' \subseteq Q$, the function $g(Q') \stackrel{\text{def}}{=} Q \cap (\bigcap_{\sigma \in \Sigma_{uc}} \{x_1 \in X \mid \forall x_2 \in X : (\langle x_1, \sigma, x_2 \rangle \in \Delta) \Rightarrow (x_2 \in Q')\})$.

- the computation of the control function f , which gives the actions that are allowed by the controller and which is defined, for each $obs \in \mathcal{D}_{Obs}$, by $f(obs) \stackrel{\text{def}}{=} \Sigma \setminus \{\sigma \in \Sigma_c \mid \exists x_1 \in Q^\dagger, \exists x_2 \notin Q^\dagger : (obs = M(x_1)) \wedge (\langle x_1, \sigma, x_2 \rangle \in \Delta)\}$.

Since the system is finite and the function g is monotonic, all computations terminate in a finite amount of time; in particular, there exists an integer n such that $Q^\dagger = \bigcap_{j=0}^n Q_j$.

We remark that $Q^\dagger = (I(Bad))^c$, because we can prove by induction that $\forall j \geq 0 : \bigcap_{i \leq j} Q_i = (\bigcup_{i \leq j} (\text{Pre}_{\Sigma_{uc}}^T)^i(Bad))^c$. Moreover, to prevent from reaching $I(Bad)$, our supervisory function \mathcal{S} forbids, for each observation $obs \in \mathcal{D}_{Obs}$, the set of actions $\mathcal{S}(obs) = \{\sigma \in \Sigma_c \mid \exists x_1 \notin I(Bad), \exists x_2 \in I(Bad) : (M(x_1) = obs) \wedge (\langle x_1, \sigma, x_2 \rangle \in \Delta)\}$. Therefore, we have that $f(obs) = \Sigma \setminus \mathcal{S}(obs)$, for each $obs \in \mathcal{D}_{Obs}$.

Improvement of the Permissiveness. Now, we explain how to improve the computation of our supervisory function \mathcal{S} , defined in (3.4), in order to obtain a more permissive controller \mathcal{C} acting on infinite state systems to be controlled. This improvement is based on the observations done in the following example.

Example 3.9

In the system depicted in Figure 3.12, (i) the set of states $X = \{x_i \mid i \in [1, 7]\}$, (ii) the set of initial states $X_0 = \{x_1, x_2\}$, (iii) the set of final states $X_m = X$, and (iv) all transitions are controllable.

Moreover, the set $Bad = \{x_5, x_6\}$ and the observer $\langle Obs, M \rangle$ is defined as follows: $\mathcal{D}_{Obs} = \{obs_1, obs_2, obs_3\}$ and the mask M is defined in the following way:

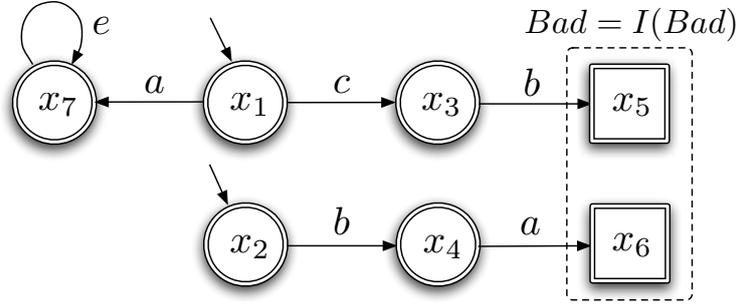


Figure 3.12 - Improvement of the control algorithm.

$$M(x) = \begin{cases} obs_1 & \text{if } x \in \{x_1, x_4, x_7\} \\ obs_2 & \text{if } x \in \{x_2, x_3\} \\ obs_3 & \text{if } x \in \{x_5, x_6\} \end{cases} \quad (3.9)$$

Our controller \mathcal{C} forbids the action b in the observation obs_2 and the action a in the observation obs_1 . However, it is sufficient to forbid b in obs_2 , because the state x_4 is then no longer reachable and this controller is strictly more permissive than \mathcal{C} , because it allows the system to reach x_7 .

We remark that when we forbid an action σ , some actions σ' can no more be fired in the system since they can only be fired from states that are no longer reachable. Our improved algorithm, which computes a controller for the basic problem, is based on this remark.

The idea of this algorithm is to choose an action $\sigma \in \Sigma$, that leads to $I(Bad)$ from a state which is not in $I(Bad)$, and to forbid this action. This operation is repeated while the set $I(Bad)$ is reachable in the current controlled system. The performance of this algorithm depends on the choice of the action σ to be forbidden. Roughly, our heuristic consists in choosing an action σ such that the controlled system, which is obtained, maximizes the states in $(I(Bad))^c$ and minimizes the states in $I(Bad)$. The main difference with the algorithm defined in section 3.4.1 is that we verify that an action can still be fired in the current controlled system, before deciding to forbid it or not. Our algorithm is composed of three subalgorithms: (i) `getController` (main algorithm), (ii) `actionToForbid` (called by `getController`) and (iii) `getMax` (called by `actionToForbid`).

Algorithm 1 (main algorithm). This algorithm computes a controller solving the

basic problem. For that, it calls Algorithm 2, which chooses an action σ to be forbidden. Next, it computes the set $\mathcal{F}(\sigma, I(Bad))$ of observations in which the action σ must be forbidden (this computation is based on (3.2)). Then, the supervisory function \mathcal{S} and the controller \mathcal{C} are modified to take into account the change of \mathcal{F} . These operations are repeated while $I(Bad)$ is reachable in the controlled system. This algorithm requires to compute the set $\text{Reachable}_{\Delta/\mathcal{C}}^{\mathcal{T}/\mathcal{C}}((\Theta_0)_{/\mathcal{C}})$ (see line 7) of states that are reachable in the controlled system $\mathcal{T}_{/\mathcal{C}}$. Therefore, to be able to compute this controlled system $\mathcal{T}_{/\mathcal{C}}$, we consider masks corresponding to partitions of the state space.

Algorithm 1: $\text{getController}(\mathcal{T}, \langle Obs, M \rangle, Bad)$

input : An STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ (with M corresponding to a partition of the state space) and a set of forbidden states Bad .

output: Either a controller \mathcal{C} solving the basic problem, or NIL when the algorithm does not find a solution to this problem.

```

1 begin
2    $\forall obs \in \mathcal{D}_{Obs}, \mathcal{S}(obs) \leftarrow \emptyset \ \backslash\backslash \ \mathcal{S}$  forbids no action.
3    $\mathcal{C} \leftarrow \langle \mathcal{S}, I(Bad) \rangle \ \backslash\backslash \ \mathcal{C}$  is defined by  $\mathcal{S}$  and  $I(Bad)$ .
4   if  $(\Theta_0)_{/\mathcal{C}} \subseteq I(Bad)$  then
5      $\backslash\backslash$  The controller  $\mathcal{C}$  is trivial
6     return (NIL)
7   while  $\text{Reachable}_{\Delta/\mathcal{C}}^{\mathcal{T}/\mathcal{C}}((\Theta_0)_{/\mathcal{C}}) \cap I(Bad) \neq \emptyset$  do
8      $\backslash\backslash$  The algorithm continues while  $I(Bad)$  is reachable.
9      $\sigma \leftarrow \text{actionToForbid}(\mathcal{T}, \langle Obs, M \rangle, Bad, \mathcal{F}, \mathcal{C})$ 
10     $\backslash\backslash$  Algorithm 2 chooses the action to be forbidden.
11     $\mathcal{F}(\sigma, I(Bad)) \leftarrow M(\text{Pre}_{\sigma}^{\mathcal{T}/\mathcal{C}}(I(Bad)) \setminus I(Bad))$ 
12     $\backslash\backslash$  Computation of the observations in which  $\sigma$  must be forbidden.
13     $\forall obs \in \mathcal{D}_{Obs}, \mathcal{S}(obs) \leftarrow \{\sigma \in \Sigma_c \mid obs \in \mathcal{F}(\sigma, I(Bad))\}$ 
14     $\backslash\backslash \ \mathcal{S}$  is modified to take into account the change of  $\mathcal{F}$ .
15     $\mathcal{C} \leftarrow \langle \mathcal{S}, I(Bad) \rangle \ \backslash\backslash \ \mathcal{C}$  is defined by  $\mathcal{S}$  and  $I(Bad)$ .
16  return ( $\mathcal{C}$ )
17 end

```

Algorithm 2. This algorithm selects an action σ that the controller \mathcal{C} in construction must forbid. For that, Algorithm 2 considers all actions σ_1 that can potentially be forbidden by this controller i.e., all actions that can be fired in the controlled system, that can lead to a forbidden state, and that have not been chosen by this algorithm in a previous call (see line 3). Next, it computes, for each of these actions σ_1 , the set of states that would be reachable if the controller \mathcal{C} (in construction) forbids σ_1 (see line 10) and it uses Algorithm 3, which is based on this reachability information, to obtain the *best* (w.r.t. our

heuristic) action to be forbidden among these actions σ_1 . Finally, it returns this action to Algorithm 1.

Algorithm 2: $\text{actionToForbid}(\mathcal{T}, \langle \text{Obs}, M \rangle, \text{Bad}, \mathcal{F}, \mathcal{C})$

input : An STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle \text{Obs}, M \rangle$ (with M corresponding to a partition of the state space), a set of forbidden states Bad , a control function \mathcal{F} in construction, and the controller \mathcal{C} defined from \mathcal{F} .

output: The action σ to be forbidden.

```

1 begin
2    $setMax \leftarrow \emptyset$ 
3   forall  $\sigma_1 \in \Sigma_c$  such that  $(\text{Pre}_{\sigma_1}^{\mathcal{T}/c}(I(\text{Bad})) \setminus I(\text{Bad})) \cap (\text{Reachable}_{\Delta/c}^{\mathcal{T}/c}((\Theta_0)/c)) \neq \emptyset$ 
   and  $\mathcal{F}(\sigma_1, I(\text{Bad})) = \emptyset$  do
4      $\setminus \setminus \sigma_1$  is a potential action to be forbidden.
5      $\forall \sigma_2 \in (\Sigma \setminus \sigma_1), \mathcal{F}_1(\sigma_2, I(\text{Bad})) \leftarrow \mathcal{F}(\sigma_2, I(\text{Bad}))$ 
6      $\setminus \setminus \mathcal{F}_1$  forbids the same actions as  $\mathcal{F}$ .
7      $\mathcal{F}_1(\sigma_1, I(\text{Bad})) \leftarrow M(\text{Pre}_{\sigma_1}^{\mathcal{T}/c}(I(\text{Bad})) \setminus I(\text{Bad})) \setminus \setminus$  And it also forbids  $\sigma_1$ .
8      $\forall obs \in \mathcal{D}_{Obs}, \mathcal{S}_1(obs) \leftarrow \{\sigma \in \Sigma_c \mid obs \in \mathcal{F}_1(\sigma, I(\text{Bad}))\}$ 
9      $\mathcal{C}_1 \leftarrow \langle \mathcal{S}_1, I(\text{Bad}) \rangle \setminus \setminus \mathcal{C}_1$  is defined by  $\mathcal{S}_1$  and  $I(\text{Bad})$ .
10     $setMax \leftarrow setMax \cup \langle \sigma_1, \text{Reachable}_{\Delta/c_1}^{\mathcal{T}/c_1}((\Theta_0)/c_1) \rangle$ 
11     $\setminus \setminus setMax$  contains the set of states that are reachable in the system  $\mathcal{T}$ 
12     $\setminus \setminus$  under the control of  $\mathcal{C}_1$ .
13   $\langle \sigma, reach \rangle \leftarrow \text{getMax}(setMax)$ 
14   $\setminus \setminus$  Algorithm 3 chooses the best (w.r.t. to our heuristic) action  $\sigma$  to be forbidden.
15  return ( $\sigma$ )
16 end

```

Algorithm 3. We want to compute an element $\langle \sigma, reach \rangle \in setMax$ such that:

1. $reach \cap (I(\text{Bad}))^c$ is \preceq_p -maximal in $setMax$ (i.e., $\nexists \langle \sigma', reach' \rangle \in setMax : (reach \cap (I(\text{Bad}))^c) \subset (reach' \cap (I(\text{Bad}))^c)$). It allows us to increase the good states in the current controlled system.
2. $reach \cap I(\text{Bad})$ is \preceq_p -minimal in $setMax$ (i.e., $\nexists \langle \sigma', reach' \rangle \in setMax : (reach \cap I(\text{Bad})) \supset (reach' \cap I(\text{Bad}))$). It allows us to decrease the forbidden states in the current controlled system.

It is actually a *Pareto criterion* [AP68], for which there does not always exist a solution. Thus, we use a classical approach to overcome this problem: we compute the set of solutions which satisfy the first criterion and then we keep all elements of this set which satisfy the second criterion. Therefore, Algorithm 3 works as follows:

- it computes the set $setMax_1$, which contains the elements $\langle \sigma_1, reach_1 \rangle \in setMax$, that satisfy the following condition (denoted by $Cond.(i)$ in the sequel): $reach_1 \cap (I(Bad))^c$ is \preceq_p -maximal in $setMax$ (i.e., $\nexists \langle \sigma_2, reach_2 \rangle \in setMax : (reach_1 \cap (I(Bad))^c) \subset (reach_2 \cap (I(Bad))^c)$).
- it computes the set $setMax_2$, which contains the elements $\langle \sigma_1, reach_1 \rangle \in setMax_1$, that satisfy the following condition (denoted by $Cond.(ii)$ in the sequel): $reach_1 \cap I(Bad)$ is \preceq_p -minimal in $setMax_1$ (i.e. $\nexists \langle \sigma_2, reach_2 \rangle \in setMax_1 : (reach_1 \cap I(Bad)) \supset (reach_2 \cap I(Bad))$).
- it non-deterministically chooses an element of $setMax_2$ and returns this value.

Algorithm 3: $getMax(setMax)$

input : A set $setMax$ of pairs $\langle \sigma, reach \rangle$.
output: An element $\langle \sigma, reach \rangle \in setMax$ which satisfies $Cond.(i)$ and $Cond.(ii)$ (see the explanations of this algorithm).

```

1 begin
2    $setMax_1 \leftarrow \{ \langle \sigma_1, reach_1 \rangle \in setMax \mid \nexists \langle \sigma_2, reach_2 \rangle \in setMax :$ 
    $((reach_1 \cap (I(Bad))^c) \subset (reach_2 \cap (I(Bad))^c)) \}$ 
3    $\backslash\backslash$  Contains the elements  $\langle \sigma_1, reach_1 \rangle \in setMax$  such that  $reach_1 \cap (I(Bad))^c$  is
4    $\backslash\backslash$   $\preceq_p$ -maximal ( $Cond.(i)$ ).
5    $setMax_2 \leftarrow \{ \langle \sigma_1, reach_1 \rangle \in setMax_1 \mid \nexists \langle \sigma_2, reach_2 \rangle \in setMax_1 :$ 
    $(reach_1 \cap I(Bad)) \supset (reach_2 \cap I(Bad)) \}$ 
6    $\backslash\backslash$  Contains the elements  $\langle \sigma_1, reach_1 \rangle \in setMax_1$  such that  $reach_1 \cap I(Bad)$  is
7    $\backslash\backslash$   $\preceq_p$ -minimal ( $Cond.(ii)$ ).
8   Chooses an element  $\langle \sigma, reach \rangle$  of  $setMax_2$  non-deterministically
9   return  $(\langle \sigma, reach \rangle)$ 
10 end
```

Algorithm 1 computes non-trivial and valid controllers for the basic problem (due to lines 4 and 7 of Algorithm 1). Moreover, this algorithm outperforms or gives, at least, the same result as the controller defined in (3.3), because the following property always holds: $\forall obs \in \mathcal{D}_{Obs}, \mathcal{S}'(obs) \subseteq \mathcal{S}(obs)$ (where \mathcal{S}' is the supervisory function computed by Algorithm 1 and \mathcal{S} is the supervisory function of the controller defined in (3.3)).

To ensure the termination of Algorithm 1, we use abstract interpretation techniques as in section 3.4.2. More precisely, we transpose the computations into the abstract lattice and perform them in this *simpler* lattice. Moreover, to ensure the convergence in the computation of the function $Reachable_{\Delta/c}^{\mathcal{T}/c}((\Theta_0)/c)$ (see line 7 of Algorithm 1 and lines 3 and 10 of Algorithm 2), we proceed similarly to the method defined for the function **Coreach** described in section 3.4.2. As explained above, these computations may require overapproximations.

3.5 Computation by Means of Abstract Interpretation of a Memoryless Controller for the Deadlock Free Problem

So far we have been interested in computing controllers solving the basic problem. But a solution to this problem does not ensure that the controlled system has the possibility to fire a transition in each reachable state. Since, most of the time, a deadlocking system is not tolerated, we extend, in this section, the method defined in section 3.4 in order to synthesize controllers for the deadlock free problem.

First, in section 3.5.1, we present a *semi-algorithm* which computes a *memoryless controller* for the deadlock free problem, and then, in section 3.5.2, we explain how to extend it to obtain, at the price of some overapproximations, an *effective algorithm*.

3.5.1 Semi-algorithm for the Deadlock Free Problem

Our algorithm, which synthesizes controllers for the deadlock free problem, is composed of two parts: (i) first, we compute, using a fixpoint equation, the set $I_{df}(Bad)$ of states that can lead to *Bad* or to *deadlocking states* by triggering only uncontrollable transitions and (ii) then, we forbid all controllable actions that can lead to a state of the set $I_{df}(Bad)$ (this part is similar to the one defined for the basic problem). Before formalizing these two steps, we explain how to compute the deadlocking states:

Computation of the Deadlocking States. Computing $I_{df}(Bad)$ requires to compute the states that would be in deadlock after control. Note that if a state $\langle \vec{v}, obs_1 \rangle$ (with $obs_1 \in M(\vec{v})$) of the controlled system is in deadlock, then the states $\langle \vec{v}, obs_2 \rangle$ ($\forall obs_2 \in M(\vec{v})$) must also be forbidden, because when the system arrives in the state \vec{v} , the controller non-deterministically receives an observation of the set $M(\vec{v})$. So, when we give the deadlocking states of the controlled system, we can omit the observations. Hence, we define a function $\text{Pre}_{df}^T : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ which gives, for each set $B \subseteq \mathcal{D}_V$ of states to be forbidden, the set $\text{Pre}_{df}^T(B)$ of states that would be in deadlock in the controlled system which avoids B . Since we cannot compute the controlled system (when the mask is a covering of the state space), we must define these deadlocking states of the controlled system

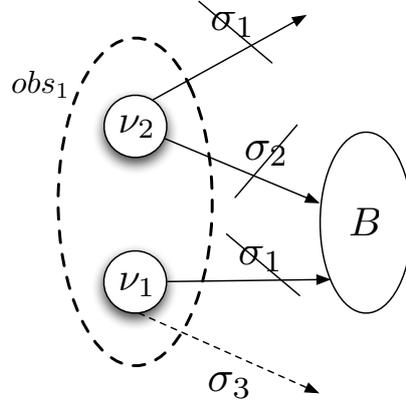


Figure 3.13 - An example of deadlocking states.

from the system to be controlled. Let $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$ be a system under the control of $\mathcal{C} = \langle \mathcal{S}, E \rangle$, which avoids the set $B \subseteq \mathcal{D}_V$, then a state $\vec{v} \in \mathcal{D}_V$ of the system \mathcal{T} will be in deadlock in the controlled system \mathcal{T}/\mathcal{C} if and only if the two following conditions are satisfied in the system \mathcal{T} :

- the state \vec{v} has no outgoing uncontrollable transition.
- there exists an observation $obs \in M(\vec{v})$ such that, for each controllable transition $\delta = \langle \sigma, G, A \rangle \in \Delta_c$, (i) this transition δ cannot be fired from \vec{v} (i.e., $\vec{v} \notin G$) or (ii) the action σ is forbidden by control in this observation obs (i.e., $\sigma \in \mathcal{S}(obs)$ which is equivalent to $obs \in \mathcal{F}(\sigma, B)$).

Formally, the deadlocking states of the controlled system can be characterized in the following way:

Definition 3.16 (Deadlocking States in the Controlled System)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$, a set $B \subseteq \mathcal{D}_V$ of states to be forbidden, and a controller \mathcal{C} , then a state $\vec{v} \in \mathcal{D}_V$ of the system \mathcal{T} will be in deadlock in the controlled system \mathcal{T}/\mathcal{C} if and only if the two following conditions hold in the system \mathcal{T} :

- $\forall \delta = \langle \sigma, G, A \rangle \in \Delta_{uc} : \vec{v} \notin G$
- $\exists obs \in M(\vec{v}), \forall \delta = \langle \sigma, G, A \rangle \in \Delta_c : (\vec{v} \notin G) \vee (obs \in \mathcal{F}(\sigma, B))$

Example 3.10

The system depicted in Figure 3.13 illustrates the definition of deadlocking states. In this system, the states \vec{v}_1 and \vec{v}_2 are compatible with the observation obs_1 . The state \vec{v}_1 is deadlock free, because the uncontrollable action σ_3 can always be fired and the state \vec{v}_2 is in deadlock, because no action can be fired from this state. Indeed, the action σ_1 is forbidden in obs_1 , because this action leads to B from \vec{v}_1 and the action σ_2 is forbidden in obs_1 , because its leads to B from \vec{v}_2 .

However, we have two remarks regarding the computation of the deadlocking states (i.e., the function Pre_{df}^T) defined in Definition 3.16:

1. The function Pre_{df}^T will be used to define the fixpoint equation $I_{df}(Bad)$ (see (3.12)). By the Knaster-Tarski theorem [Tar55], if the function Pre_{df}^T is monotonic, then it is ensured that the fixpoint equation $I_{df}(Bad)$ admits a solution. However, the computation of the deadlocking states depends on the function \mathcal{F} (see Definition 3.16) which is not monotonic. Therefore, in the sequel, we will use the monotonic function $\widehat{\mathcal{F}}$ instead of \mathcal{F} in the computation of these states:

$$\widehat{\mathcal{F}}(\sigma, B) = \begin{cases} M(\text{Pre}_{\sigma}^T(B)) & \text{if } \sigma \in \Sigma_c \\ \emptyset & \text{otherwise} \end{cases} \quad (3.10)$$

One can note that the function $\widehat{\mathcal{F}}$ is rougher than \mathcal{F} .

2. Computing all states, that will be in deadlock in the controlled system, is not possible with our symbolic operations. Indeed, for the second condition of Definition 3.16, we must combine, for each controllable transition $\langle \sigma, G, A \rangle$, the set of states G^c and the set of observations $\mathcal{F}(\sigma, B)$, but the domains of these sets are different. If we transform $\mathcal{F}(\sigma, B)$ into $M^{-1}(\mathcal{F}(\sigma, B))$, we no longer know in which observations the action σ has been forbidden. However, we need this information, because we have an existential quantification on the observations in the second condition of Definition 3.16. If we transform G^c into $M(G^c)$, then there can be states in $M^{-1}(M(G^c))$ from which δ can be fired.

To overcome the second remark, we will consider the subcase⁹ where the mask M is a partition of the state space. Indeed, in this case, each state $\vec{v} \in \mathcal{D}_V$

⁹In [KLMM10], we have shown that the deadlocking states can also be computed in the subcase where M is a *covering* and the observation space \mathcal{D}_{Obs} is *finite*.

is compatible with *exactly* one observation, which simplifies the definition of deadlocking states and makes possible the computation of these elements:

Definition 3.17 (Deadlocking States when M is a partition)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ (where M corresponds to a *partition* of the state space), a set $B \subseteq \mathcal{D}_V$ of states to be forbidden, and a controller \mathcal{C} , then a state $\vec{v} \in \mathcal{D}_V$ of the system \mathcal{T} will be in deadlock in the controlled system \mathcal{T}/\mathcal{C} if and only if the two following conditions¹⁰ hold in the system \mathcal{T} :

- $\forall \delta = \langle \sigma, G, A \rangle \in \Delta_{uc} : \vec{v} \notin G$
- $\forall \delta = \langle \sigma, G, A \rangle \in \Delta_c : (\vec{v} \notin G) \vee (\vec{v} \in M^{-1}(\widehat{\mathcal{F}}(\sigma, B)))$

A state \vec{v} is in deadlock if it has no outgoing uncontrollable transition and if its outgoing controllable transitions $\delta = \langle \sigma, G, A \rangle$ are forbidden by control in the observation $M(\vec{v})$ (i.e., $\sigma \in \mathcal{S}(M(\vec{v}))$ which is equivalent to $\vec{v} \in M^{-1}(\widehat{\mathcal{F}}(\sigma, B))$).

Since, for each $\sigma \in \Sigma_{uc}$, the set $\widehat{\mathcal{F}}(\sigma, B) = \emptyset$, the function $\text{Pre}_{df}^{\mathcal{T}}(B)$ (with $B \subseteq \mathcal{D}_V$), which gives the set of states, that would be in deadlock in the controlled system avoiding B , can be expressed as follows:

$$\text{Pre}_{df}^{\mathcal{T}}(B) = B \cup \left[\bigcap_{\langle \sigma, G, A \rangle \in \Delta} \left(G^c \cup (M^{-1}(\widehat{\mathcal{F}}(\sigma, B))) \right) \right] \quad (3.11)$$

Now that we have a function (see (3.11)) computing the deadlocking states, we formalize the two steps of our algorithm defined for the deadlock free problem:

Computation of $I_{df}(Bad)$. The set $I_{df}(Bad)$ corresponds to the set of states leading uncontrollably to a forbidden state or to a deadlocking state. To compute $I_{df}(Bad)$, we first compute the set $I(Bad)$ (defined in (3.1)). Then, if we make unreachable these forbidden states by cutting all controllable transitions that lead to a bad state, the corresponding controlled system could have new deadlocking states. These states are given by the function $\text{Pre}_{df}^{\mathcal{T}}(I(Bad))$ and we add them to the set of forbidden states. Adding these deadlocking states to the set of forbidden states can provide new states leading uncontrollably to a forbidden state. Consequently, the set $I_{df}(Bad)$ is defined by $\bigcup_{n \geq 0} (\text{Pre}_{df}^{\mathcal{T}} \circ I)^n(Bad)$. This

¹⁰We assume that $M^{-1}(\emptyset) = \emptyset$.

set of states cannot always be computed, because the coreachability is *undecidable* in the model of STS, and, in section 3.5.2, we will use abstract interpretation techniques to overapproximate it. However, to compute this overapproximation, we must characterize the set $I_{df}(Bad)$ by a fixpoint equation. This equation is defined as follows over the complete lattice $\langle 2^{\mathcal{D}_V}, \subseteq, \cup, \cap, \mathcal{D}_V, \emptyset \rangle$:

$$I_{df}(Bad) \stackrel{\text{def}}{=} \text{lfp}^{\subseteq}(\lambda B. Bad \cup \text{Pre}_{df}^{\mathcal{T}}(I(B))) \quad (3.12)$$

Since the function $\lambda B. Bad \cup \text{Pre}_{df}^{\mathcal{T}}(I(B))$ is monotonic, the Knaster-Tarski's theorem [Tar55] ensures that the least fixpoint of this function exists and the following proposition proves that this fixpoint equation gives the set of states leading uncontrollably to a forbidden state or to a deadlocking state:

Proposition 3.12

The function $\lambda B. Bad \cup \text{Pre}_{df}^{\mathcal{T}}(I(B))$ defined over the complete lattice $\langle 2^{\mathcal{D}_V}, \subseteq, \cup, \cap, \mathcal{D}_V, \emptyset \rangle$ is such that $\text{lfp}^{\subseteq}(\lambda B. Bad \cup \text{Pre}_{df}^{\mathcal{T}}(I(B))) = \bigcup_{n \geq 0} (\text{Pre}_{df}^{\mathcal{T}} \circ I)^n(Bad)$.

Proof

By the Knaster-Tarski and Kleene's theorems, it suffices to show that the functions I and $\text{Pre}_{df}^{\mathcal{T}}$ are continuous to prove this proposition. In Proposition 3.9, we have already proven that the function I is continuous. For the second function, we must prove that, for each increasing sequence of sets $B_1 \subseteq B_2 \subseteq \dots \subseteq B_n \subseteq \dots$ (where $\forall i \geq 1 : B_i \in 2^{\mathcal{D}_V}$), the equality $\text{Pre}_{df}^{\mathcal{T}}(\bigcup_{i \geq 1} B_i) = \bigcup_{i \geq 1} \text{Pre}_{df}^{\mathcal{T}}(B_i)$ holds. This equality is proved by the following reasoning, where a transition $\delta \in \Delta$ is defined by the triple $\langle \sigma, G, A \rangle$:

$$\begin{aligned} & \text{Pre}_{df}^{\mathcal{T}}\left(\bigcup_{i \geq 1} B_i\right) \\ &= \left(\bigcup_{i \geq 1} B_i\right) \cup \left[\bigcap_{\delta \in \Delta} \left(G^c \cup \left(M^{-1}(\widehat{\mathcal{F}}(\sigma, \bigcup_{i \geq 1} B_i)) \right) \right) \right], \text{ by} \\ & \quad \text{definition of } \text{Pre}_{df}^{\mathcal{T}}. \\ &= \left(\bigcup_{i \geq 1} B_i\right) \cup \left[\bigcap_{\delta \in \Delta} \left(G^c \cup \left[M^{-1}(M(\text{Pre}_{\sigma}^{\mathcal{T}}\left(\bigcup_{i \geq 1} B_i\right))) \right] \right) \right], \\ & \quad \text{by definition of } \widehat{\mathcal{F}}. \end{aligned}$$

$$\begin{aligned}
&= \left(\bigcup_{i \geq 1} B_i \right) \cup \left[\bigcap_{\delta \in \Delta} \left[G^c \cup \left[\bigcup_{i \geq 1} (M^{-1}(M(\text{Pre}_\sigma^T(B_i)))) \right] \right] \right], \\
&\quad \text{because } \text{Pre}_\sigma^T = \text{Pre}_{\text{Trans}(\sigma)}^T \text{ is continuous (see Proof of} \\
&\quad \text{Proposition 3.9) and because } M \text{ and } M^{-1} \text{ are} \\
&\quad \text{continuous.} \\
&= \left(\bigcup_{i \geq 1} B_i \right) \cup \left[\bigcap_{\delta \in \Delta} \left[\bigcup_{i \geq 1} \left(G^c \cup [M^{-1}(M(\text{Pre}_\sigma^T(B_i)))] \right) \right] \right], \\
&\quad \text{because } a \cup \left(\bigcup_i b_i \right) = \bigcup_i (a \cup b_i). \\
&= \left(\bigcup_{i \geq 1} B_i \right) \cup \left[\bigcup_{i \geq 1} \left[\bigcap_{\delta \in \Delta} \left(G^c \cup [M^{-1}(M(\text{Pre}_\sigma^T(B_i)))] \right) \right] \right], \\
&\quad \text{because } \Delta \text{ is finite and } a \cap \left(\bigcup_i b_i \right) = \bigcup_i (a \cap b_i). \\
&= \bigcup_{i \geq 1} \left(B_i \cup \left[\bigcap_{\delta \in \Delta} \left(G^c \cup [M^{-1}(M(\text{Pre}_\sigma^T(B_i)))] \right) \right] \right) \\
&= \bigcup_{i \geq 1} \text{Pre}_{df}^T(B_i)
\end{aligned}$$

Computation of the Controller \mathcal{C} . The controller \mathcal{C} is defined as in section 3.4.1 (see (3.3)) by using $I_{df}(Bad)$ instead of $I(Bad)$. More precisely, the controller \mathcal{C} is defined by the pair $\langle \mathcal{S}, I_{df}(Bad) \rangle$ where the supervisory function \mathcal{S} is defined as in (3.4) by using the function $\mathcal{F}(\sigma, I_{df}(Bad))$ instead of $\mathcal{F}(\sigma, I(Bad))$ ($\forall \sigma \in \Sigma$).

Example 3.11

For the system depicted in Figure 3.1, the controller must prevent the system from reaching the set $Bad = \{ \langle \text{CX}, x, x', y, y' \rangle \mid (x \leq 0) \wedge (y \in [0, 2]) \} \cup \{ \langle \text{PX}, x, x', y, y' \rangle \mid x \geq 6 \}$. The controller has a partial observation of the system, which is modeled by the mask M where, for each state $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$:

- if $x \geq 2$, then the controller cannot distinguish \vec{v} from the states in the set $\{\langle \ell, x_1, x', y, y' \rangle \mid x_1 \geq 2\}$,
- if $x < 2$, then the controller perfectly observes this state.

The computation of the set $I(Bad)$ gives:

$$I(Bad) = Bad \cup \{\langle CX, x, x', y, y' \rangle \mid [(x \leq 1) \wedge (y \in [1, 2])] \vee [(x \leq 2) \wedge (y = 2)]\}$$

and the computation of the function $\widehat{\mathcal{F}}$ gives:

$$\widehat{\mathcal{F}}(\sigma, I(Bad)) = \begin{cases} M(\{\langle PX, x, x', y, y' \rangle \mid x \leq 2\}) & \text{if } \sigma = Stop_prod \\ M(\{\langle PX, x, x', y, y' \rangle \mid x \geq 5\}) & \text{if } \sigma = Prod \\ M(\{\langle Choice, x, x', y, y' \rangle \mid x \geq 6\}) & \text{if } \sigma = Choice_X \\ \emptyset & \text{otherwise} \end{cases}$$

Therefore, to prevent the system from reaching $I(Bad)$, the controller forbids (i) the action *Stop_prod* when the system is in the location **PX**, (ii) the action *Prod* when the system is in a state $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid x \geq 2\}$, and (iii) the action *Choice_X* when the system is in a state $\vec{v} \in \{\langle Choice, x, x', y, y' \rangle \mid x \geq 2\}$. Since the states $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid x \geq 2\}$ are in deadlock, they are added to the forbidden states i.e., $\text{Pre}_{df}^T(I(Bad)) = I(Bad) \cup \{\langle PX, x, x', y, y' \rangle \mid x \geq 2\}$. The set of states $I(\text{Pre}_{df}^T(I(Bad))) = \text{Pre}_{df}^T(I(Bad))$. Consequently, to prevent the system from reaching these states, the controller forbids (i) the action *Stop_prod* when the system is in the location **PX**, (ii) the action *Prod* when the system is in a state $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid x \geq 1\}$, and (iii) the action *Choice_X* when the system is in a state $\vec{v} \in \{\langle Choice, x, x', y, y' \rangle \mid x \geq 2\}$. Since the states $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid x \geq 1\}$ are in deadlock, they are added to the forbidden states. By continuing the computations in this way, we obtain (i) a set $I_{df}(Bad) = I(Bad) \cup \{\langle PX, x, x', y, y' \rangle \mid x, x', y, y' \in \mathbb{Z}\}$ and (ii) a control function \mathcal{F} defined by:

$$\mathcal{F}(\sigma, I_{df}(Bad)) = \begin{cases} M(\{\langle Choice, x, x', y, y' \rangle \mid x, x', y, y' \in \mathbb{Z}\}) & \text{if } \sigma = Choice_X \\ \emptyset & \text{otherwise} \end{cases}$$

Thus, the controller always forbids the action $Choice_X$ in the location $Choice$ to prevent the system from reaching $I_{df}(Bad)$. One can note that the controller computed by our algorithm is \preceq_p -maximal.

The following property proves that our algorithm synthesizes correct controllers for the deadlock free problem:

Proposition 3.13

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Theta_m, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, defined in section 3.5.1, solves the deadlock free problem (when $\Theta_0 \not\subseteq E$).

Proof

Since $I(Bad) \subseteq I_{df}(Bad)$, it can be proved, as in the proof of Proposition 3.10, that Bad is not reachable in this more restrictive controlled system.

Now, let us suppose that the controlled system \mathcal{T}/\mathcal{C} does not satisfy the deadlock free property. Then, there exists at least one deadlocking state $\vec{v} \in \mathcal{D}_V$, which is reachable in this controlled system. By definition of the fixpoint equation $I_{df}(Bad)$ (see (3.12)), this state $\vec{v} \in I_{df}(Bad)$, and so is any state $\vec{v}' \in \mathcal{D}_V$ which can lead to \vec{v} by a sequence of uncontrollable transitions. But according to the definition of \mathcal{C} , the states \vec{v} and \vec{v}' are both unreachable in the controlled system \mathcal{T}/\mathcal{C} , which implies a contradiction.

3.5.2 Effective Algorithm for the Deadlock Free Problem

The computation of the algorithm described in the previous section does not always terminate and we proceed as in section 3.4.2 to overcome this obstacle.

More precisely, we define an effective algorithm which works like the semi-algorithm defined in section 3.5.1, except that we perform the computations in the abstract lattice and that we compute an overapproximation of $I_{df}(Bad)$ to ensure the termination of our algorithm. To compute this overapproximation, we use the instantiation of the representation framework defined in section 3.4.2 and we transpose the fixpoint equation $I_{df}(Bad)$ into the abstract lattice in the following way:

$$I_{df}^\sharp(Bad) \stackrel{\text{def}}{=} \text{lfp}^\sqsubseteq(\lambda \ell. \alpha(Bad) \sqcup \text{Pre}_{df}^{\mathcal{T}, \sharp}(I^\sharp(\ell))) \quad (3.13)$$

where (i) the fixpoint equation $I^\sharp(\ell)$ is defined in (3.7) and (ii) the function $\text{Pre}_{df}^{\mathcal{T}, \sharp} : \Lambda \rightarrow \Lambda$ (as a reminder, Λ is the domain of the abstract lattice) is defined,

for each $\ell \in \Lambda$, by:

$$\text{Pre}_{df}^{\mathcal{T},\#}(\ell) \stackrel{\text{def}}{=} \ell \sqcup \left[\prod_{\langle \sigma, G, A \rangle \in \Delta} \left[\alpha(G^c) \sqcup \left(\alpha(M^{-1}(M(\gamma(\text{Pre}_{\text{Trans}(\sigma)}^{\mathcal{T},\#}(\ell)))))) \right) \right] \right] \quad (3.14)$$

where $\text{Pre}_{\text{Trans}(\sigma)}^{\mathcal{T},\#}(\ell)$ is defined in (3.5).

Next, we use the fixpoint computation strategy defined in section 3.4.2 to compute a post-fixpoint a_∞ of $I_{df}^\#(Bad)$ and the element $\gamma(a_\infty)$ (denoted by $I'_{df}(Bad)$) gives an overapproximation of $I_{df}(Bad)$.

Then, we transpose the control function \mathcal{F} into the abstract lattice and we obtain the abstract function $\mathcal{F}^\#$. Finally, we define our controller $\mathcal{C} = \langle \mathcal{S}, I'_{df}(Bad) \rangle$ as in (3.3) by using $I'_{df}(Bad)$ and $\mathcal{F}^\#$.

3.6 The Non-blocking Problem

The computation of controllers for the non-blocking problem is interesting, but our approach, based on the computation of safe approximations, cannot efficaciously be applied to this problem. Indeed, let us suppose that we adapt our approach, defined for the deadlock free problem, to the non-blocking problem by computing the set $I_{bl}(Bad)$ of states that can lead to Bad or to blocking states by triggering only uncontrollable transitions. The set of blocking states of a system \mathcal{T} can then be obtained by computing the complement of the set of non-blocking states, which is given by $\text{Coreach}_\Delta^{\mathcal{T}}(\Theta_m)$. Since the computation of $\text{Coreach}_\Delta^{\mathcal{T}}(\Theta_m)$ does not always terminate, we can use abstract interpretation techniques to compute an overapproximation of this set. However, we then obtain an underapproximation of the blocking states and hence the controllers that we compute are not correct. To obtain correct controllers, we must compute an underapproximation of $\text{Coreach}_\Delta^{\mathcal{T}}(\Theta_m)$, but the computation of such approximations with abstract interpretation do not generally give good results.

3.7 Experimental Results

The algorithms presented in the previous sections have been implemented in our tool SMACS (Symbolic MAsked Controller Synthesis) in order to evaluate our approach experimentally. In section 3.7.1, we briefly describe our tool (a more detailed description of SMACS and an explanation of our implementation choices

are given in appendix A) and, in section 3.7.2, we present several examples and describe the solutions computed by SMACS.

3.7.1 Description of SMACS

The input of SMACS is a description of the STS to be controlled (we consider STS with explicit locations) and the variables of this system can be of two types: integer and real. The guards of the transitions are boolean combinations of linear constraints and the assignments are given by linear expressions. The set of forbidden states is specified by a boolean combination of linear constraints and the masks can be of four kinds:

- **Undistinguishable locations:** the controller cannot distinguish some given locations of the system.
- **Hidden variables:** the controller has no information regarding the value of some given variables.
- **Partially hidden variables:** the value of a numerical variable is unknown when it belongs to a given interval (the user can specify an interval for each variable).
- **Undistinguishable intervals:** when the system is in a state $\vec{\nu} = \langle \nu_1, \dots, \nu_n \rangle \in \mathcal{D}_V$, the controller cannot distinguish this state from the states in the set $\{\langle \nu'_1, \dots, \nu'_n \rangle \in \mathcal{D}_V \mid \forall i \in [1, n] : \nu'_i \in [\nu_i - c_i, \nu_i + c'_i]\}$, where $c_i, c'_i \in \mathbb{R}$ ($\forall i \in [1, n]$) are constants. One can note that c_i and c'_i can be different from c_j and c'_j ($\forall i \neq j \in [1, n]$).

The first three masks are partitions of the state space and the last one is a covering. If no mask is specified, the controller has a full observation of the system.

The output of SMACS is a description of the function \mathcal{F} : it displays, for each action σ , the set of observations in which σ is forbidden. Moreover, SMACS computes and displays (graphically) the controlled system for the first three masks.

SMACS allows the use of three abstract lattices: the lattice of intervals, the lattice of octagons and the lattice of convex polyhedra. The lattice of intervals and the lattice of convex polyhedra have respectively been defined in section 1.4.1 and in Definition 3.15. The lattice of octagons can be seen as a particular case of the lattice of convex polyhedra where the coefficients $a_{i,j}$ (for each $i \in [1, k]$ and for each $j \in [1, n]$), used in Definition 3.14, are equal to 1.

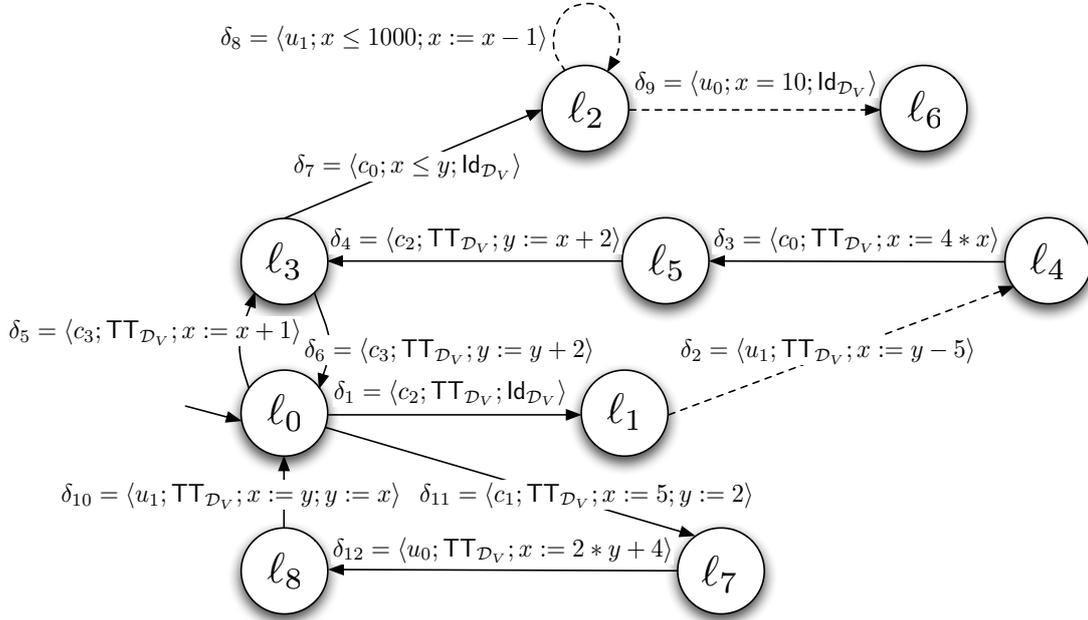


Figure 3.14 - Toy example.

3.7.2 Empirical Evaluation

We consider several examples and present the solutions computed by SMACS in order to evaluate our approach experimentally. Our experiments are performed with several abstract lattices.

Toy Example. The first system to be controlled is depicted in Figure 3.14. This STS has a tuple $V = \langle \ell, x, y \rangle$ of three variables, where (i) $\ell \in \{\ell_i \mid i \in [0, 8]\}$ gives the current location of the system, and (ii) $x, y \in \mathbb{Z}$. The initial state is $\langle \ell_0, 0, 0 \rangle$ and the set *Bad* of forbidden states is defined by $\{\langle \ell_6, x, y \rangle \mid x, y \in \mathbb{Z}\}$. We consider several cases that are summarized in Table 3.1:

- (i) The controller does not ensure the deadlock free property and it has a full observation of the system. With the lattices of convex polyhedra and octagons, SMACS computes a set $I(\text{Bad}) = \text{Bad} \cup \{\langle \ell_2, x, y \rangle \mid 10 \leq x \leq 1000\}$. Next, SMACS forbids the action c_0 in the location ℓ_3 when $(10 \leq x \leq 1000) \wedge (y \geq x)$ to prevent the system from reaching $I(\text{Bad})$. The graphical output of the controlled system generated by SMACS is given in Figure 3.15. With the lattice of intervals, the control is more restrictive:

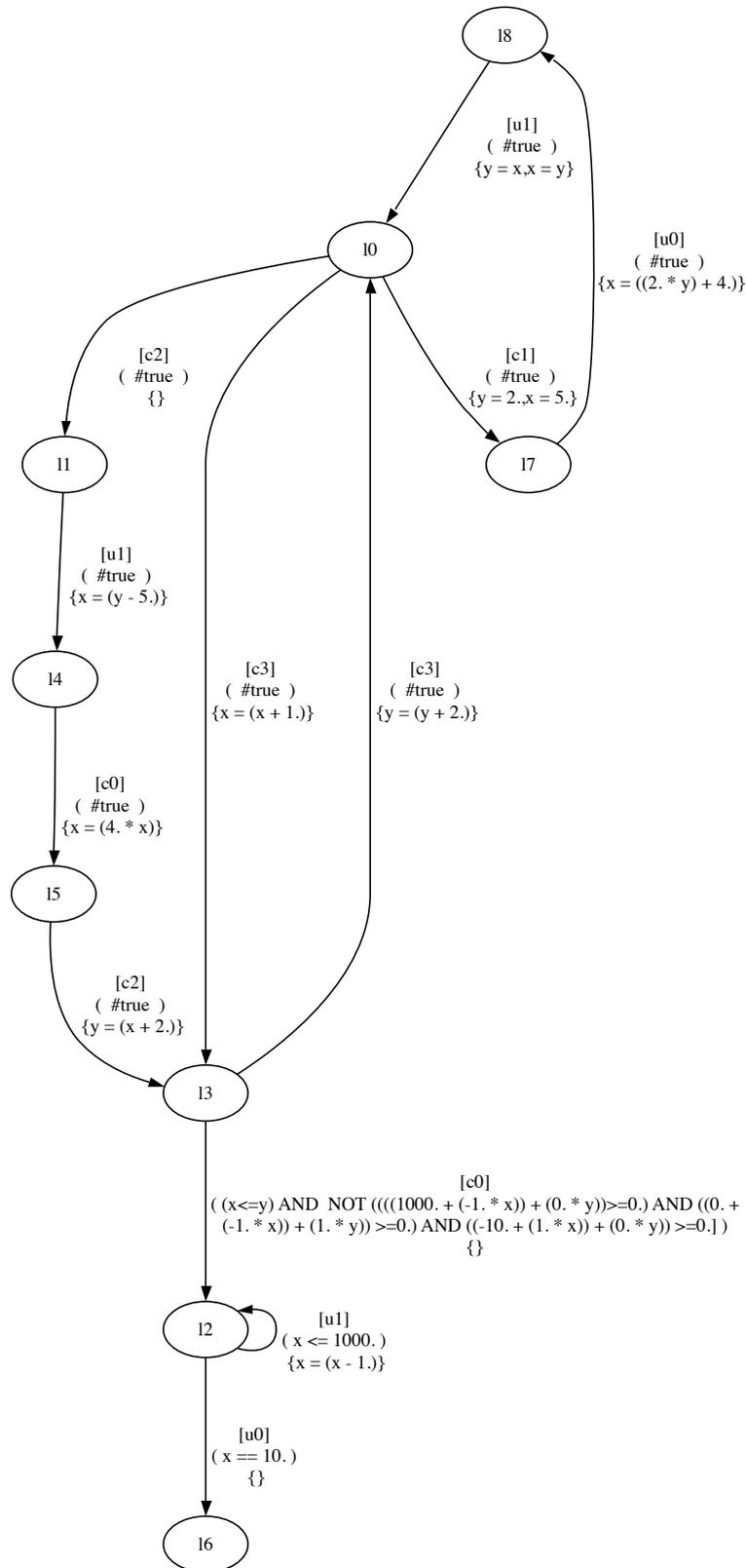


Figure 3.15 - Controlled system generated by SMACS for the toy example when the controller has a full observation of the system and when the deadlock free property is not ensured.

SMACS forbids the action c_0 in the location ℓ_3 when $(x \leq 1000) \wedge (y \geq 0)$. SMACS obtains these results in 0.01 second¹¹

- (ii) The controller ensures the deadlock free property for the case (i). With the lattices of convex polyhedra and octagons, SMACS computes a set $I_{df}(Bad) = Bad \cup \{\langle \ell_2, x, y \rangle \mid 10 \leq x\}$, because the states $\vec{v} \in \{\langle \ell_2, x, y \rangle \mid x > 1000\}$ are in deadlock. Next, SMACS prevents the system from firing the action c_0 in the location ℓ_3 when $(10 \leq x) \wedge (y \geq x)$. With the lattice of intervals, SMACS always forbids the action c_0 in the location ℓ_3 . SMACS obtains these results in 0.01 second.

- (iii) The controller does not ensure the deadlock free property and it does not distinguish the locations ℓ_3 and ℓ_4 (i.e., $\forall x, y \in \mathbb{Z} : M(\langle \ell_3, x, y \rangle) = M(\langle \ell_4, x, y \rangle)$). With the lattices of convex polyhedra and octagons, the set $I(Bad) = Bad \cup \{\langle \ell_2, x, y \rangle \mid 10 \leq x \leq 1000\}$. SMACS forbids the action c_0 in the location ℓ_3 when $(10 \leq x \leq 1000) \wedge (y \geq x)$, and it also forbids this action in the location ℓ_4 when $(10 \leq x \leq 1000) \wedge (y \geq x)$, because of the partial observation. With the lattice of intervals, SMACS forbids the action c_0 in the locations ℓ_3 and ℓ_4 when $(x \leq 1000) \wedge (y \geq 0)$. SMACS obtains these results in 0.01 second.

- (iv) The controller ensures the deadlock free property for the case (iii). With the lattices of convex polyhedra and octagons, SMACS computes a set $I_{df}(Bad) = Bad \cup \{\langle \ell_2, x, y \rangle \mid 10 \leq x\} \cup \{\langle \ell_4, x, y \rangle \mid (10 \leq x) \wedge (y \geq x)\} \cup \{\langle \ell_1, x, y \rangle \mid y \geq 15\}$. The states $\vec{v} \in \{\langle \ell_4, x, y \rangle \mid (10 \leq x) \wedge (y \geq x)\}$ are forbidden, because they are in deadlock. The states $\vec{v} \in \{\langle \ell_1, x, y \rangle \mid y \geq 15\}$ are forbidden because they uncontrollably lead to the previous set of deadlocking states. SMACS forbids the action c_0 in the location ℓ_3 when $(10 \leq x) \wedge (y \geq x)$, the action c_0 in the location ℓ_4 when $(10 \leq x) \wedge (y \geq x)$ (because of the partial information), and the action c_2 in the location ℓ_0 when $y \geq 15$. With the lattice of intervals, SMACS forbids the action c_0 in the locations ℓ_3 and ℓ_4 , and the action c_2 in the location ℓ_0 . SMACS obtains these results in 0.01 second.

- (v) The controller does not ensure the deadlock free property. It perfectly observes the system and the transition δ_7 is replaced by $\langle c_0; x \leq y; x := 2 * x + 1 \rangle$. With the lattice of convex polyhedra, SMACS computes a set

¹¹All experiments are performed on an Intel Core i7 with 12 GB of RAM.

Toy example	Synthesis of a \preceq_p -maximal controller ?		
	Convex polyhedra	Octagons	Intervals
Case (i)	Yes	Yes	No
Case (ii)	Yes	Yes	No
Case (iii)	Yes	Yes	No
Case (iv)	Yes	Yes	No
Case (v)	Yes	No	No
Case (vi)	Yes	No	No

Table 3.1 - Controllers computed by SMACS for the toy example by using the lattices of convex polyhedra, octagons, and intervals. All computations are done in 0.01 second.

$I(Bad) = Bad \cup \{\langle \ell_2, x, y \rangle \mid 10 \leq x \leq 1000\}$ and it forbids the action c_0 in the location ℓ_3 when $(2.x \leq 999) \wedge (2.x \geq 9) \wedge (y \geq x)$. With the lattices of octagons and intervals, the results are rougher: with the first lattice, SMACS forbids the action c_0 in the location ℓ_3 when $y \geq x$, and with the second one, it always forbids the action c_0 in the location ℓ_3 . SMACS obtains these results in 0.01 second.

- (vi) The controller ensures the deadlock free property for the case (v). With the lattice of convex polyhedra, SMACS computes a set $I_{df}(Bad) = Bad \cup \{\langle \ell_2, x, y \rangle \mid 10 \leq x\}$ and it forbids the action c_0 in the location ℓ_3 when $(2.x \geq 9) \wedge (y \geq x)$. With the lattices of octagons and intervals, the control is more restrictive: with the first lattice, SMACS forbids the action c_0 in the location ℓ_3 when $y \geq x$, and with the second one, it always forbids the action c_0 in the location ℓ_3 . SMACS obtains these results in 0.01 second.

Cat and Mouse. The STS depicted in Figure 3.16 illustrates a modified version of the cat and mouse example given in [RW87]. This STS has a tuple $V = \langle \ell, x, y \rangle$ of three variables, where (i) $\ell \in \{\text{Awake, Cheese, Dead, Sleep, Tired}\}$ gives the current location of the system and (ii) x (resp. y) $\in \mathbb{Z}$ identifies the number of the room occupied by the mouse (resp. the cat). The initial state is $\langle \text{Sleep}, 1, 0 \rangle$. When the cat wakes up (transition δ_3), it can either eat the mouse if both are in the same room (transition δ_5), or move to another room (transition δ_4) and sleep again (transition δ_6). In the location **Cheese**, if the mouse is between the room 1 and the room 1000, it can smell the cheese and it then moves to the room 0, where it is killed by a trap. The set Bad of forbidden states is defined

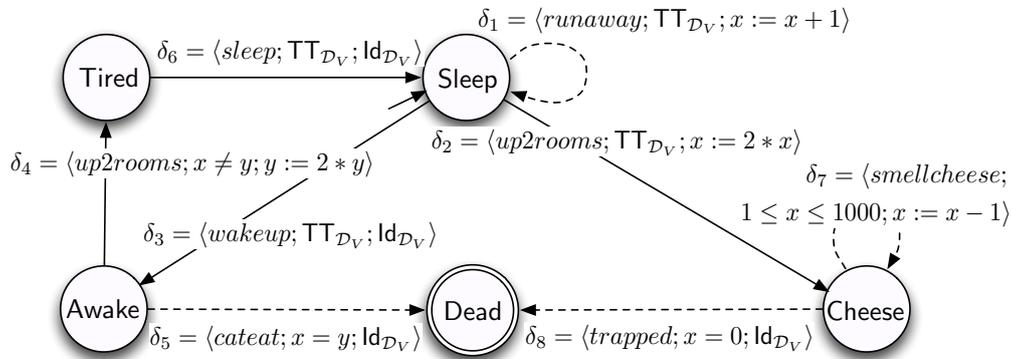


Figure 3.16 - The cat and mouse example

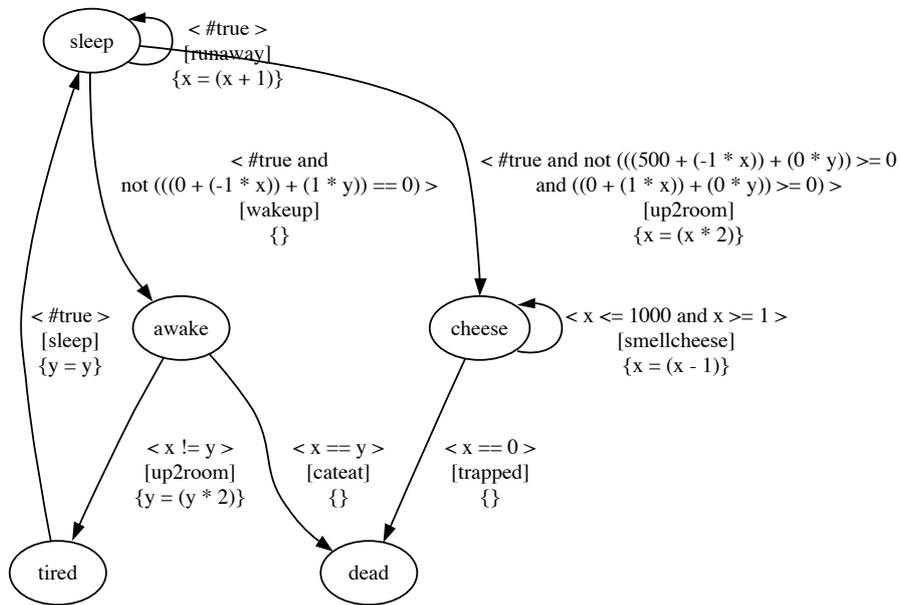


Figure 3.17 - Controlled system generated by SMACS for the cat and mouse example when the controller has a full observation of the system and when the deadlock free property is not ensured.

by $\{\langle \text{Dead}, x, y \rangle \mid x, y \in \mathbb{Z}\}$ and we consider several cases that are summarized in Table 3.2:

- (i) The controller does not ensure the deadlock free property and it has a full observation of the system. With the lattice of convex polyhedra, SMACS computes a set $I(\text{Bad}) = \text{Bad} \cup \{\langle \text{Cheese}, x, y \rangle \mid 0 \leq x \leq 1000\} \cup \{\langle \text{Awake}, x, y \rangle \mid x = y\}$. It forbids the action *up2rooms* in the location **Sleep** when $(0 \leq x) \wedge (x \leq 500)$ and the action *wakeup* in the location **Sleep** when $x = y$. The graphical output of the controlled system generated by SMACS is given in Figure 3.17. With the lattices of octagons and intervals, the results are rougher. With the first lattice, SMACS forbids the action *up2rooms* in the location **Sleep** and the action *wakeup* in the location **Sleep** when $x = y$. With the second lattice, it always forbids the actions *up2rooms* and *wakeup* in the location **Sleep**. SMACS obtains these results in 0.01 second.
- (ii) The controller ensures the deadlock free property for the case (i). With the lattices of convex polyhedra and octagons, the computation of the set $I_{af}(\text{Bad})$ gives $\text{Bad} \cup \{\langle \text{Cheese}, x, y \rangle \mid x, y \in \mathbb{Z}\} \cup \{\langle \text{Awake}, x, y \rangle \mid x = y\}$. Next, SMACS forbids the action *up2rooms* in the location **Sleep** and the action *wakeup* in the location **Sleep** when $x = y$. With the lattice of intervals, SMACS always forbids the actions *up2rooms* and *wakeup* in the location **Sleep**. SMACS obtains these results in 0.01 second.
- (iii) The controller does not ensure the deadlock free property and it has a partial observation of the system. This partial observation is modeled by a mask $M : \text{Loc} \times \mathbb{Z} \times \mathbb{Z} \rightarrow 2^{\text{Loc} \times \mathbb{Z} \times \mathbb{Z}}$, where, for each state $\vec{v} = \langle \ell, x, y \rangle \in \mathcal{D}_V$, the set of states which are undistinguishable from \vec{v} is given by $\{\langle \ell, x', y' \rangle \mid (x' \in [x - 4, x + 4]) \wedge (y' \in [y - 4, y + 4])\}$. Thus, there is an imprecision regarding the positions of the cat and of the mouse. With the lattice of convex polyhedra, SMACS computes a set $I(\text{Bad}) = \text{Bad} \cup \{\langle \text{Cheese}, x, y \rangle \mid 0 \leq x \leq 1000\} \cup \{\langle \text{Awake}, x, y \rangle \mid x = y\}$. It forbids the action *up2rooms* in the location **Sleep** when $(x \leq 504) \wedge (x \geq -4)$ and the action *wakeup* in the location **Sleep** when $(x - y \leq 8) \wedge (y - x \leq 8)$. With the lattices of octagons and intervals, the control is more restrictive. With the first lattice, SMACS forbids the action *up2rooms* in the location **Sleep** and the action *wakeup* in the location **Sleep** when $(x - y \leq 8) \wedge (y - x \leq 8)$. With the second lattice, SMACS always forbids the actions *up2rooms* and *wakeup* in the location **Sleep**. SMACS obtains these results in 0.01 second.

- (iv) The controller ensures the deadlock free property for the case (iii). With the lattices of convex polyhedra and octagons, the computation of the set $I_{df}(Bad)$ gives $Bad \cup \{\langle \text{Cheese}, x, y \rangle \mid x, y \in \mathbb{Z}\} \cup \{\langle \text{Awake}, x, y \rangle \mid x = y\}$. SMACS forbids the action *up2rooms* in the location **Sleep** and the action *wakeup* in the location **Sleep** when $(x - y \leq 8) \wedge (y - x \leq 8)$. With the lattice of intervals, SMACS always forbids the actions *up2rooms* and *wakeup* in the location **Sleep**. SMACS obtains these results in 0.01 second.

- (v) The controller does not ensure the deadlock free property and it has a partial observation of the system. This partial observation is modeled by a mask $M : Loc \times \mathbb{Z} \times \mathbb{Z} \rightarrow 2^{Loc \times \mathbb{Z} \times \mathbb{Z}}$, where, for each state $\vec{v} = \langle \ell, x, y \rangle \in \mathcal{D}_V$, the set of states which are undistinguishable from \vec{v} is given by $\{\langle \ell, x, y' \rangle \mid y' \in \mathbb{Z}\}$. Thus, the controller does not observe the variable y . With the lattice of convex polyhedra, SMACS computes a set $I(Bad) = Bad \cup \{\langle \text{Cheese}, x, y \rangle \mid 0 \leq x \leq 1000\} \cup \{\langle \text{Awake}, x, y \rangle \mid x = y\}$. It forbids the action *up2rooms* in the location **Sleep** when $(x \leq 500) \wedge (x \geq 0)$ and the action *wakeup* in the location **Sleep** (because it does not observe the variable y). With the lattices of octagons and intervals, the control is more restrictive: SMACS always forbids the actions *up2rooms* and *wakeup* in the location **Sleep**. SMACS obtains these results in 0.01 second.

- (vi) The controller ensures the deadlock free property for the case (v). With the lattices of convex polyhedra and octagons, SMACS computes a set $I_{df}(Bad) = Bad \cup \{\langle \text{Cheese}, x, y \rangle \mid x, y \in \mathbb{Z}\} \cup \{\langle \text{Awake}, x, y \rangle \mid x = y\}$. It forbids the actions *up2rooms* and *wakeup* in the location **Sleep**. With the lattice of intervals, SMACS computes the same controller. It obtains these results in 0.01 second.

Producer and Consumer. We consider a modified version of the producer and consumer example given in Figure 3.1, where the transitions δ_1 , δ_4 , δ_5 and δ_8 are respectively replaced by $\langle \text{Cons}; 0 \leq y \leq 500; x := x - 1, y := y - 1 \rangle$, $\langle \text{Stop_prod}; \top\top_{\mathcal{D}_V}; y := 500 \rangle$, $\langle \text{Cons}' ; 0 \leq y' \leq 500; x' := x' - 1, y' := y' - 1 \rangle$, and $\langle \text{Stop_prod}' ; \top\top_{\mathcal{D}_V}; y' := 500 \rangle$, which makes the system *more difficult* to be controlled. The set Bad of forbidden states is defined by $\{\langle \text{CX}, x, x', y, y' \rangle \mid (x \leq 10) \wedge (y \in [0, 500])\} \cup \{\langle \text{CX}', x, x', y, y' \rangle \mid (x' \leq 10) \wedge (y' \in [0, 500])\}$ and we consider several cases that are summarized in Table 3.3:

- (i) The controller does not ensure the deadlock free property and it has a full observation of the system. With the lattices of convex polyhedra and

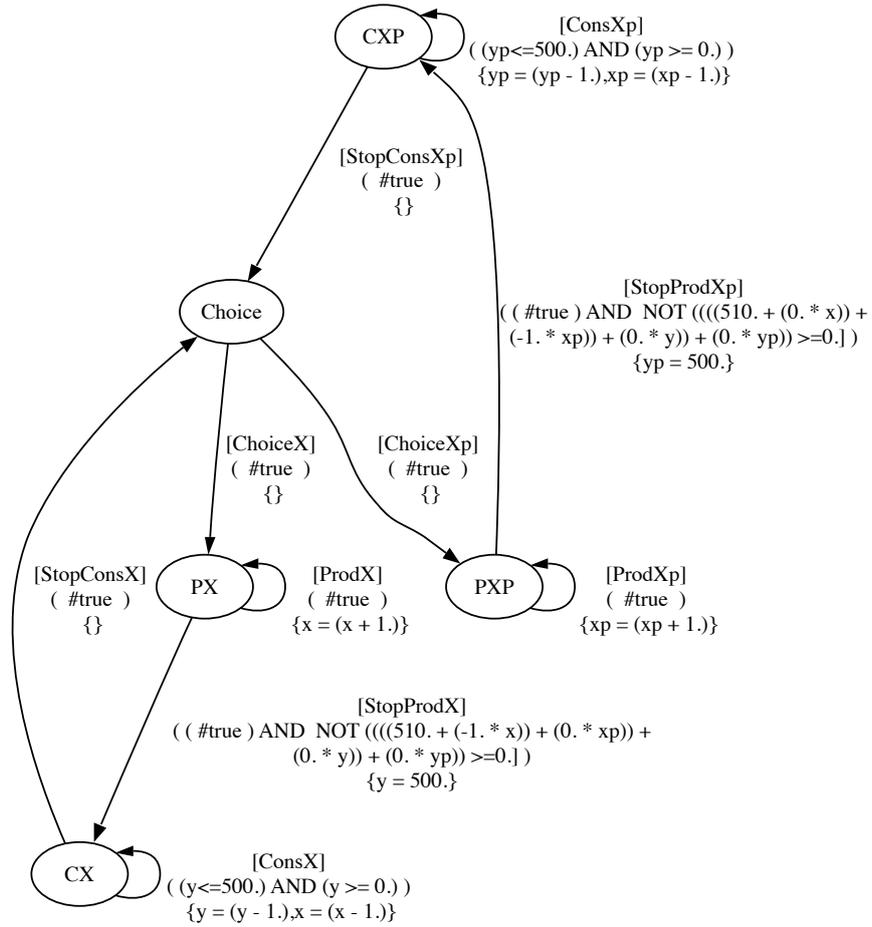


Figure 3.18 - Controlled system generated by SMACS for the producer and consumer example (where $\delta_1 = \langle Cons; 0 \leq y \leq 500; x := x - 1, y := y - 1 \rangle$, $\delta_4 = \langle Stop_prod; \top \top_{\mathcal{D}_V}; y := 500 \rangle$, $\delta_5 = \langle Cons'; 0 \leq y' \leq 500; x' := x' - 1, y' := y' - 1 \rangle$, and $\delta_8 = \langle Stop_prod'; \top \top_{\mathcal{D}_V}; y' := 500 \rangle$) when the controller has a full observation of the system and when the deadlock free property is not ensured.

Cat and mouse example	Synthesis of a \preceq_p -maximal controller ?		
	Convex polyhedra	Octagons	Intervals
Case (i)	Yes	No	No
Case (ii)	Yes	Yes	No
Case (iii)	Yes	No	No
Case (iv)	Yes	Yes	No
Case (v)	Yes	No	No
Case (vi)	Yes	Yes	Yes

Table 3.2 - Controllers computed by SMACS for the cat and mouse example by using the lattices of convex polyhedra, octagons, and intervals. All computations are done in 0.01 second.

octagons, SMACS forbids the action *Stop_prod* in the location PX when $x \leq 510$ and the action *Stop_prod'* in the location PX' when $x' \leq 510$. The graphical output of the controlled system generated by SMACS is given in Figure 3.18. With the lattices of intervals, the control is more restrictive: SMACS always forbids the action *Stop_prod* in the location PX and the action *Stop_prod'* in the location PX'. SMACS obtains these results in 0.01 second.

- (ii) The controller does not ensure the deadlock free property and it has a partial observation of the system. This partial observation is modeled by the mask M defined in Example 3.3 i.e., for each state $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, the set of states which are undistinguishable from \vec{v} , is given by $\{\langle \ell, x_1, x', y, y' \rangle \mid x_1 \in [x - 1, x + 1]\}$. With the lattices of convex polyhedra and octagons, SMACS forbids the action *Stop_prod* in the location PX when $x \leq 511$ and the action *Stop_prod'* in the location PX' when $x' \leq 510$. With the lattice of intervals, SMACS always forbids the action *Stop_prod* in the location PX and the action *Stop_prod'* in the location PX'. SMACS obtains these results in 0.01 second.
- (iii) The controller does not ensure the deadlock free property and it has a partial observation of the system. This partial observation is modeled by a mask M where, for each state $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, the set of states which are undistinguishable from \vec{v} , is given by $\{\langle \ell, x, x'_1, y, y'_1 \rangle \mid x'_1, y'_1 \in \mathbb{Z}\}$. Thus, the controller does not observe the variables x' and y' . With the lattices of convex polyhedra and octagons, SMACS forbids the action

Producer and consumer example	Synthesis of a \preceq_p -maximal controller ?		
	Convex polyhedra	Octagons	Intervals
Case (i)	Yes	Yes	No
Case (ii)	Yes	Yes	No
Case (iii)	Yes	Yes	No

Table 3.3 - Controllers computed by SMACS for the producer and consumer example by using the lattices of convex polyhedra, octagons, and intervals. All computations are done in 0.01 second.

Stop_prod in the location PX when $x \leq 510$ and the action *Stop_prod'* in the location PX' (because it does not observe the variable x'). With the lattice of intervals, SMACS always forbids the action *Stop_prod* in the location PX and the action *Stop_prod'* in the location PX'. SMACS obtains these results in 0.01 second.

Now, we consider more complex systems to be controlled by increasing the number of variables in the previous examples (see Tables 3.4 to 3.12):

- Toy (Tables 3.4 to 3.6): the number of variables of the system is increased and these variables are used in the uncontrollable transitions δ_8 and δ_9 that can lead to forbidden states. The controller does not observe some of these new variables. With the lattice of intervals, SMACS uses few memory and computes the controllers fast. However, these controllers are very restrictive. With the lattices of convex polyhedra and octagons, SMACS computes maximal controllers and we can remark that the performance (time and memory) is better with the first lattice. For example, when the system has 220 variables, SMACS computes the controller with the lattice of convex polyhedra in 38.6 seconds and uses 108 MB of RAM, whereas it needs 722 seconds and 492 MB of RAM with the lattice of octagons. It is due to the fact that SMACS uses more linear constraints to represent a set of states with the lattice of octagons.
- Cat and Mouse (Tables 3.7 to 3.9): the number of variables of the system is increased by adding mice. The controller has a perfect observation of the system and must prevent the dead of all mice. The better performance (time and memory) is obtained with the lattice of intervals, but the controllers that SMACS computes with this lattice are very restrictive. With the lattice of octagons, SMACS computes better controllers, but they are not maximal. Moreover, the performance (time and memory) is not good. SMACS only

Number of variables in the toy example	Convex polyhedra		
	Time	Maximal Memory (MB)	\preceq_p -Maximal Controller ?
30	0.28	< 12	Yes
80	2.47	60	Yes
130	8.9	72	Yes
180	21.8	96	Yes
220	38.6	108	Yes

Table 3.4 - The time (in seconds) and the memory (in MB) that SMACS uses with the lattice of convex polyhedra to synthesize a controller for a modified version of the toy example.

computes maximal controllers with the lattice of convex polyhedra and the performance obtained with this lattice is better than the one obtained with the lattice of octagons.

- Producer and Consumer (Tables 3.10 to 3.12): the system is made more complex by producing n (where $n > 0$) kinds of parts. The production of each kind of parts requires the definition of two variables. The control requirements consist in ensuring that there are at least 11 parts of each kind. Moreover, the controller does not observe the value of some variables when it belongs to a given interval. Again, the better performance (time and memory) is obtained with the lattice of intervals, but the controllers that SMACS computes with this lattice are very rough. With the lattices of convex polyhedra and octagons, SMACS computes maximal controllers and we can remark that the performance (time and memory) is better with the first lattice. For example, when the system has 160 variables, SMACS computes the controller with the lattice of convex polyhedra in 92.3 seconds and uses 624 MB of RAM, whereas it needs 1181 seconds and 2124 MB of RAM with the lattice of octagons.

Number of variables in the toy example	Octagons		
	Time	Maximal Memory (MB)	\preceq_p -Maximal Controller ?
30	2.4	24	Yes
80	36.9	240	Yes
130	157.7	312	Yes
180	391.1	396	Yes
220	722	492	Yes

Table 3.5 - The time (in seconds) and the memory (in MB) that SMACS uses with the lattice of octagons to synthesize a controller for a modified version of the toy example.

Number of variables in the toy example	Intervals		
	Time	Maximal Memory (MB)	\preceq_p -Maximal Controller ?
30	0.06	< 12	No
80	0.08	< 12	No
130	0.15	< 12	No
180	0.25	< 12	No
220	0.37	< 12	No

Table 3.6 - The time (in seconds) and the memory (in MB) that SMACS uses with the lattice of intervals to synthesize a controller for a modified version of the toy example.

Number of variables in the cat and mouse example	Convex polyhedra		
	Time	Maximal Memory (MB)	\preceq_p -Maximal Controller ?
10	0.08	< 12	Yes
40	0.8	36	Yes
90	4.3	60	Yes
140	14.3	84	Yes
200	36.1	108	Yes

Table 3.7 - The time (in seconds) and the memory (in MB) that SMACS uses with the lattice of convex polyhedra to synthesize a controller for a modified version of the cat and mouse example.

Number of variables in the cat and mouse example	Octagons		
	Time	Maximal Memory (MB)	\preceq_p -Maximal Controller ?
10	0.3	< 12	No
40	5.8	216	No
90	58.6	228	No
140	210.3	372	No
200	601	468	No

Table 3.8 - The time (in seconds) and the memory (in MB) that SMACS uses with the lattice of octagons to synthesize a controller for a modified version of the cat and mouse example.

Number of variables in the cat and mouse example	Intervals		
	Time	Maximal Memory (MB)	\preceq_p -Maximal Controller ?
10	0.03	< 12	No
40	0.06	< 12	No
90	0.15	< 12	No
140	0.22	< 12	No
200	0.41	< 12	No

Table 3.9 - The time (in seconds) and the memory (in MB) that SMACS uses with the lattice of intervals to synthesize a controller for a modified version of the cat and mouse example.

Number of variables in the producer and consumer example	Convex polyhedra		
	Time	Maximal Memory (MB)	\preceq_p -Maximal Controller ?
40	1.26	96	Yes
80	8.8	168	Yes
120	33.4	288	Yes
160	92.3	624	Yes
180	141.2	852	Yes

Table 3.10 - The time (in seconds) and the memory (in MB) that SMACS uses with the lattice of convex polyhedra to synthesize a controller for a modified version of the producer and consumer example.

Number of variables in the producer and consumer example	Octagons		
	Time	Maximal Memory (MB)	\preceq_p -Maximal Controller ?
40	7.8	408	Yes
80	82.4	876	Yes
120	381.6	1284	Yes
160	1181	2124	Yes
180	times out (> 1200)	—	—

Table 3.11 - The time (in seconds) and the memory (in MB) that SMACS uses with the lattice of octagons to synthesize a controller for a modified version of the producer and consumer example.

Number of variables in the producer and consumer example	Intervals		
	Time	Maximal Memory (MB)	\preceq_p -Maximal Controller ?
40	0.15	< 12	No
80	0.61	< 12	No
120	1.7	12	No
160	3.6	24	No
180	5	< 36	No

Table 3.12 - The time (in seconds) and the memory (in MB) that SMACS uses with the lattice of intervals to synthesize a controller for a modified version of the producer and consumer example.

Chapter 4

Synthesis of Centralized k -memory and Online Controllers with Partial Observation for Infinite State Systems

 IN the previous chapter, we provided methods to synthesize *memoryless* controllers for infinite state systems under partial observation. In these algorithms, the control decisions taken by the controller are only based on the current observation that it receives from the system. A way to improve the permissiveness of the controllers is to provide them with memory in order to retain the history of the execution of the system partially or totally; this actually allows the controller to have a finer knowledge of the set of states in which the system can be. There are mainly two ways to use this memory:

- The first possibility is to record the last k (where k is a finite number) observations received from the system. So, the controller knows a part of the history of the execution of the system. It can then use this information to refine its knowledge of the possible paths taken by the system and hence to have more accurate control decisions. These controllers are called *k-memory controllers*.
- The second possibility is to retain the set of states in which the system can

be. More precisely, during the execution of the system, when the controller receives an observation from the system, it updates its estimate of the current state of the system from this observation and its control decisions. Next, it uses this estimate to determine the actions that it forbids. That allows, in some sense, to retain the entire history of the execution of the system, because the estimate computed by the controller is based on all observations received from the beginning of the execution of the system and on all its control decisions. These controllers are called *online controllers*.

In this chapter, we study these two possibilities. First, in section 4.1, we define an algorithm which synthesizes a k -memory controller for the state avoidance control problem. Next, in section 4.2, we define an algorithm that synthesizes an online controller. In section 4.3, we compare the permissiveness of the memoryless, k -memory and online controllers synthesized by our algorithms.

4.1 Synthesis of k -Memory Controllers

In this section, we extend the method presented in chapter 3 to synthesize k -memory controllers for the state avoidance control problem. First, in section 4.1.1, we define the framework that we use. Next, in section 4.1.2, we define a *semi-algorithm* which computes a k -memory controller for the state avoidance control problem. Finally, in section 4.1.3, we explain how to extend this semi-algorithm by using abstract interpretation to obtain an *effective* algorithm.

4.1.1 Framework and State Avoidance Control Problem

The framework is quite similar to the one defined in section 3.3; the only changes concern the definition of the controller and hence the definition of the controlled system. In this framework, a k -memory controller can retain the last k observations that it received from the system and use this information to choose the actions to be forbidden. It is formally defined as follows:

Definition 4.1 (k -Memory Controller)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$, and an integer¹ $k \geq 1$, a *k -memory controller* with partial observation for the system \mathcal{T} is a pair $\mathcal{C}_k = \langle \mathcal{S}_k, E_k \rangle$, where:

- $\mathcal{S}_k : (\mathcal{D}_{Obs})^k \rightarrow 2^{\Sigma_c}$ is a supervisory function which defines, for each k -tuple of observations $\langle obs_1, \dots, obs_k \rangle \in (\mathcal{D}_{Obs})^k$, a set $\mathcal{S}_k(\langle obs_1, \dots, obs_k \rangle)$ of controllable actions that have to be forbidden when obs_1, \dots, obs_k are the last k observations received by the controller from the system (note that obs_k is the last observation received from the system).
- $E_k \subseteq \mathcal{D}_V$ is a set of states, which restricts the set of initial states of \mathcal{T} .

Let $\langle obs_1, \dots, obs_k \rangle$ be the content of the controller's memory (where obs_k corresponds to the last observation that the controller received from the system), when the controller receives a new observation obs_{k+1} , its memory evolves from $\langle obs_1, \dots, obs_k \rangle$ to $\langle obs_2, \dots, obs_{k+1} \rangle$. Note that at the beginning of the execution of the system (before the first observation), the memory of the controller is initialized to $\langle \Gamma, \dots, \Gamma \rangle$ (Γ is thus a special symbol which means that a component of the memory contains no observation).

The controlled system $\mathcal{T}/\mathcal{C}_k$ resulting from the interaction between the system \mathcal{T} and the k -memory controller \mathcal{C}_k is defined by an STS with k additional variables, which provide the last k observations that the controller received from the system. It is formally defined as follows:

Definition 4.2 (Controlled System)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$, and a k -memory controller $\mathcal{C}_k = \langle \mathcal{S}_k, E_k \rangle$, the system \mathcal{T} controlled by \mathcal{C}_k , denoted by $\mathcal{T}/\mathcal{C}_k$, is an STS including observations which is defined by $\mathcal{T}/\mathcal{C}_k \stackrel{\text{def}}{=} \langle V/\mathcal{C}_k, (\Theta_0)_{/\mathcal{C}_k}, \Sigma, \Delta_{/\mathcal{C}_k} \rangle$, where:

- $V/\mathcal{C}_k \stackrel{\text{def}}{=} V \cup \langle Obs_1, \dots, Obs_k \rangle$ is a pair composed of the tuple of variables V and of the k -tuple of variables $\langle Obs_1, \dots, Obs_k \rangle$. The domain \mathcal{D}_{Obs_i} of each variable Obs_i ($\forall i \in [1, k]$) is defined by $\mathcal{D}_{Obs_i} \stackrel{\text{def}}{=} \mathcal{D}_{Obs} \cup \{\Gamma\}$.
- $(\Theta_0)_{/\mathcal{C}_k} \stackrel{\text{def}}{=} \{ \langle \vec{v}, \langle \Gamma, \dots, \Gamma, obs \rangle \rangle \mid (\vec{v} \in (\Theta_0 \setminus E_k)) \wedge (obs \in M(\vec{v})) \}$ is the set of initial states.

¹Note that $k \neq \infty$.

- $\Delta_{/C_k}$ is defined from Δ as follows: for each transition $\langle \sigma, G, A \rangle \in \Delta$, we define a new transition $\langle \sigma, G_{/C_k}, A_{/C_k} \rangle \stackrel{\text{def}}{\in} \Delta_{/C_k}$, where (i) $G_{/C_k} \stackrel{\text{def}}{=} \{\langle \vec{v}, \langle obs_1, \dots, obs_k \rangle \rangle \mid (\vec{v} \in G) \wedge (obs_k \in M(\vec{v})) \wedge (\sigma \notin \mathcal{S}_k(\langle obs_1, \dots, obs_k \rangle))\}$, and (ii) $A_{/C_k}$ is defined, for each $\vec{v}_1 \in \mathcal{D}_V$ and for each $\langle obs_1, \dots, obs_k \rangle \in (\mathcal{D}_{Obs})^{k-1} \times M(\vec{v}_1)$, by $A_{/C_k}(\langle \vec{v}_1, \langle obs_1, \dots, obs_k \rangle \rangle) \stackrel{\text{def}}{=} \{\langle \vec{v}_2, \langle obs_2, \dots, obs_k, obs_{k+1} \rangle \rangle \mid (\vec{v}_2 = A(\vec{v}_1)) \wedge (obs_{k+1} \in M(\vec{v}_2))\}$.

This definition generalizes Definition 3.10. A state $\langle \vec{v}, \langle obs_1, \dots, obs_k \rangle \rangle$ (with $obs_k \in M(\vec{v})$) of the controlled system models the fact that when the system was in the state \vec{v} , the observations obs_1, \dots, obs_k were the last k information that the controller received from the system. The guards and the update functions of the controlled system $\mathcal{T}_{/C_k}$ are computed from the ones of the system \mathcal{T} by taking into account the control decisions of the controller C_k and the observations that it can receive from the system.

- **Guards:** A transition $\langle \sigma, G_{/C_k}, A_{/C_k} \rangle \in \Delta_{/C_k}$, defined from the transition $\delta = \langle \sigma, G, A \rangle \in \Delta$, can be fired from a state $\langle \vec{v}, \langle obs_1, \dots, obs_k \rangle \rangle \in \mathcal{D}_{V_{/C_k}}$ in the controlled system if and only if (i) δ can be fired from \vec{v} in \mathcal{T} , (ii) \vec{v} is compatible with obs_k , and (iii) σ is not forbidden by the controller C_k in $\langle obs_1, \dots, obs_k \rangle$.
- **Update functions:** The update function $A_{/C_k}$ of the transition $\langle \sigma, G_{/C_k}, A_{/C_k} \rangle \in \Delta_{/C_k}$, defined from the transition $\delta = \langle \sigma, G, A \rangle \in \Delta$, allows the controlled system to reach the state $\langle \vec{v}_2, \langle obs_2, \dots, obs_k, obs_{k+1} \rangle \rangle \in \mathcal{D}_{V_{/C_k}}$ from a state $\langle \vec{v}_1, \langle obs_1, \dots, obs_k \rangle \rangle \in \mathcal{D}_{V_{/C_k}}$ if and only if (i) the update function A allows the system \mathcal{T} to reach \vec{v}_2 from \vec{v}_1 , and (ii) \vec{v}_2 is compatible with obs_{k+1} . The definition of $A_{/C_k}$ is based on the evolution of the controller's memory when it receives a new observation obs_{k+1} : the memory evolves from $\langle obs_1, \dots, obs_k \rangle$ to $\langle obs_2, \dots, obs_k, obs_{k+1} \rangle$. Moreover, the update functions $A_{/C_k}$ satisfy the following property: $(\langle \vec{v}_2, \langle obs_2, \dots, obs_k, obs_{k+1} \rangle \rangle \in A_{/C_k}(\langle \vec{v}_1, \langle obs_1, \dots, obs_k \rangle \rangle)) \Rightarrow (\forall obs'_{k+1} \in M(\vec{v}_2) : \langle \vec{v}_2, \langle obs_2, \dots, obs_k, obs'_{k+1} \rangle \rangle \in A_{/C_k}(\langle \vec{v}_1, \langle obs_1, \dots, obs_k \rangle \rangle))$. This property means that when the system arrives in a state \vec{v}_2 , the controller can receive any observation of \vec{v}_2 .

The problem, that we want to solve, is the basic problem adapted to this framework². It is formally defined as follows:

Problem 4.1 (Memory Basic State Avoidance Control Problem)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the *memory basic state avoidance control problem* (*memory basic problem* for short) consists in computing a k -memory controller $\mathcal{C}_k = \langle \mathcal{S}_k, E_k \rangle$ such that (i) $\text{Reachable}_{\Delta/\mathcal{C}_k}^{\mathcal{T}/\mathcal{C}_k}((\Theta_0)_{/\mathcal{C}_k}) \neq \emptyset$ and (ii) $\left(\text{Reachable}_{\Delta/\mathcal{C}_k}^{\mathcal{T}/\mathcal{C}_k}((\Theta_0)_{/\mathcal{C}_k}) \right]_{[1]} \cap Bad = \emptyset$.

As a reminder, $\left(\text{Reachable}_{\Delta/\mathcal{C}_k}^{\mathcal{T}/\mathcal{C}_k}((\Theta_0)_{/\mathcal{C}_k}) \right]_{[1]} = \{ \vec{v} \mid \exists \langle \vec{v}, \langle obs_1, \dots, obs_k \rangle \rangle \in \text{Reachable}_{\Delta/\mathcal{C}_k}^{\mathcal{T}/\mathcal{C}_k}((\Theta_0)_{/\mathcal{C}_k}) \}$ (see section 1.1). This problem consists in synthesizing a *non-trivial* and *valid* k -memory controller.

Note that the memoryless controller is a particular case of the k -memory controller, since it corresponds to the case where $k = 1$. Therefore, all propositions proved in section 3.3.4 remain valid. In particular, the memory basic problem is *undecidable*, which explains the use of abstract interpretation in section 4.1.3 to obtain an effective algorithm.

4.1.2 Semi-algorithm for the Memory Basic Problem

Our algorithm, which synthesizes controllers for the memory basic problem, is an extension of the one defined for the memoryless case and is also composed of two parts.

Computation of $I(Bad)$. The set $I(Bad)$ of states leading uncontrollably to Bad is given by the fixpoint equation defined in (3.1) which, as a reminder, is equal to $\text{lfp}^{\subseteq}(\lambda B. Bad \cup \text{Pre}_{\Delta_{uc}}^{\mathcal{T}}(B))$.

Computation of the Controller \mathcal{C}_k . The second step consists in computing the controller \mathcal{C}_k whose aim is to forbid all controllable actions that can lead to a state in $I(Bad)$. For that, we first define the function $\mathcal{F}_k : \Sigma \times 2^{\mathcal{D}_V} \rightarrow 2^{(\mathcal{D}_{Obs})^k}$, which gives, for each action $\sigma \in \Sigma$ and set $B \subseteq \mathcal{D}_V$ of states to be forbidden, the set $\mathcal{F}_k(\sigma, B)$ of sequences of observations of length less than or equal to k for

²We only consider this case to remain quite concise, but controllers ensuring the deadlock free property can also be computed for this framework.

which the action σ must be forbidden. Each k -tuple $\langle obs_1, \dots, obs_k \rangle \in \mathcal{F}_k(\sigma, B)$ should correspond to a possible sequence of observations of the system. Moreover, the last observation obs_k of this sequence must be such that there exists a state \vec{v} compatible with this observation which leads to B by a transition labeled by σ .

To obtain $\mathcal{F}_k(\sigma, B)$, we compute all possible sequences of states of length k that end by a state \vec{v} leading to B by a transition labeled by σ and all possible sequences of states of length strictly less than k that start by an initial state and that ends by a state \vec{v} leading to B by a transition labeled by σ . Moreover, we apply the mask M to these sequences to take into account the paths that are undistinguishable from them. We compute this set with the following algorithm:

Algorithm 4. In order to compute $\mathcal{F}_k(\sigma, B)$, Algorithm 4 considers all sequences $\delta_1, \dots, \delta_k$ of transitions of length k (with δ_k labeled by σ) and, for each of these sequences, it iteratively computes (line 10) the sequences of observations that can occur by executing this sequence. It starts this iteration by computing the set $Pred$ of states that do not belong to B and that lead to B by δ_k (line 11), and by applying the mask M to this set (line 25). Next, for each δ_i ($\forall i \in [1, k - 1]$), it performs a similar operation by computing the set of states that are not in B and that lead to the set $Pred$ (computed at the previous iteration) by δ_i (line 11) and by applying the mask M to this set (line 25). However, two particular cases must be considered:

- some sequences $\delta_1, \dots, \delta_k$ of transitions of length k cannot lead to B . This situation is detected when the set $Pred$ computed for a transition δ_i is empty (line 13). In this case, the computations for this sequence are stopped.
- some sequences of transitions of length strictly less than k can lead to B . These sequences start from an initial state and this situation is thus detected when the set $Pred$ computed for a transition δ_i contains initial states (line 17). These sequences of transitions generate sequences of observations of length strictly less than k and we add some symbols Γ to the beginning of them to obtain sequences of observations of length k that we add to the set $\mathcal{F}_k(\sigma, B)$. Adding these symbols Γ does not make erroneous the computation of the controller, because, at the beginning of the execution of the system, the memory of the controller is initialized to $\langle \Gamma, \dots, \Gamma \rangle$.

The output $\mathcal{F}_k(\sigma, B)$ of Algorithm 4 corresponds to a finite union of k -tuples K of sets of observations. This function forbids the action σ after a sequence

Algorithm 4: $\text{computeFm}(\mathcal{T}, \langle \text{Obs}, M \rangle, \sigma, B, k)$

input : An STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle \text{Obs}, M \rangle$, an action $\sigma \in \Sigma$, a set of forbidden states $B \subseteq \mathcal{D}_V$, and a fixed integer $k \geq 1$.

output: $\mathcal{F}_k(\sigma, B)$.

```

1 begin
2   if  $\sigma \in \Sigma_{uc}$  then
3      $\mathcal{F}_k(\sigma, B) \leftarrow \emptyset$ 
4   else
5     foreach  $\delta_1, \dots, \delta_k \in \Delta$  s.t.  $\delta_k \in \text{Trans}(\sigma)$  do
6        $\backslash\backslash$  we enumerate all sequences of transitions of length  $k$ 
7        $addPath \leftarrow \text{tt}$   $\backslash\backslash$  indicates if the sequence  $\delta_1, \dots, \delta_k$  leads to  $B$ 
8       Let  $Tmp_1$  be an empty  $k$ -tuple
9        $Pred \leftarrow B$ 
10      for  $i \leftarrow k$  downto 1 do
11         $Pred \leftarrow \text{Pre}_{\delta_i}^{\mathcal{T}}(Pred) \setminus B$ 
12         $\backslash\backslash$  it gives the states that are not in  $B$  and that lead to  $Pred$ 
13        if  $Pred = \emptyset$  then
14           $\backslash\backslash$  the sequence  $\delta_1, \dots, \delta_k$  does not lead to  $B$ 
15           $addPath \leftarrow \text{ff}$ 
16          break
17        if  $Pred \cap \Theta_0 \neq \emptyset$  then
18           $\backslash\backslash$  there are sequences of observations of length less than  $k$ 
19           $\backslash\backslash$  that lead to  $B$ 
20           $Tmp_2 \leftarrow Tmp_1$ 
21           $Tmp_2]_{[i]} \leftarrow M(Pred \cap \Theta_0)$   $\backslash\backslash$  we only keep the initial states
22          for  $j \leftarrow i$  downto 1 do
23             $Tmp_2]_{[j]} \leftarrow \Gamma$ 
24             $\mathcal{F}_k(\sigma, B) \leftarrow \mathcal{F}_k(\sigma, B) \cup Tmp_2$ 
25           $Tmp_1]_{[i]} \leftarrow M(Pred)$ 
26          if  $addPath = \text{tt}$  then
27             $\mathcal{F}_k(\sigma, B) \leftarrow \mathcal{F}_k(\sigma, B) \cup Tmp_1$ 
28      return( $\mathcal{F}_k(\sigma, B)$ )
29 end

```

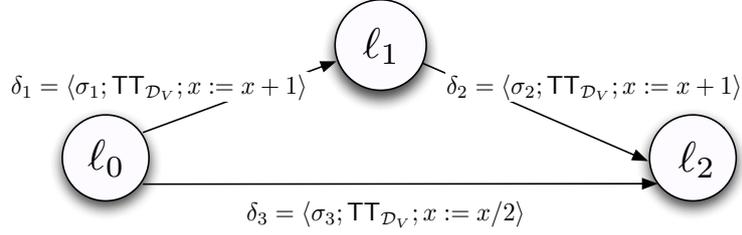


Figure 4.1 - Computation of the function \mathcal{F}_k .

$\langle obs_1, \dots, obs_k \rangle$ of observations if and only if $\langle obs_1, \dots, obs_k \rangle$ belongs to one of these k -tuples K .

Example 4.1

The STS depicted in Figure 4.1 illustrates Algorithm 4. This system has a tuple $V = \langle \ell, x \rangle$ of two variables, where (i) $\ell \in \{\ell_0, \ell_1, \ell_2\}$, and (ii) $x \in \mathbb{N}_0$. The set Θ_0 of initial states is defined by $\{\langle \ell_0, 5 \rangle, \langle \ell_0, 6 \rangle, \langle \ell_0, 8 \rangle, \langle \ell_0, 10 \rangle\}$ and the set $B = \{\langle \ell_2, 10 \rangle, \langle \ell_2, 12 \rangle\}$. The controller perfectly observes the system. The computation of $\mathcal{F}_k(\sigma_3, B)$ for a 2-memory controller gives $\langle \Gamma, \{\langle \ell_0, 5 \rangle, \langle \ell_0, 6 \rangle\} \rangle$ and the computation of $\mathcal{F}_k(\sigma_2, B)$ gives $\langle \{\langle \ell_0, 8 \rangle, \langle \ell_0, 10 \rangle\}, \{\langle \ell_1, 9 \rangle, \langle \ell_1, 11 \rangle\} \rangle$. For example, \mathcal{F}_k forbids σ_2 after the sequence of observations $\langle \langle \ell_0, 10 \rangle, \langle \ell_1, 11 \rangle \rangle$.

One can note that our computation of the function \mathcal{F}_k is only an overapproximation of the sequences of observations leading to B . Indeed, in Example 4.1, the sequence of observations $\langle \langle \ell_0, 8 \rangle, \langle \ell_1, 11 \rangle \rangle$ belongs to $\mathcal{F}_k(\sigma_2, B)$, whereas it cannot occur in the system. An exact computation should record all possible and precise paths leading to B , but with infinite state spaces, we cannot always represent all these paths in a finite way (they are potentially infinite).

Next, by using this function \mathcal{F}_k , we can define the controller \mathcal{C}_k in the following way:

$$\mathcal{C}_k \stackrel{\text{def}}{=} \langle \mathcal{S}_k, E_k \rangle \quad (4.1)$$

with the elements \mathcal{S}_k and E_k defined as follows:

- the supervisory function \mathcal{S}_k is given, for each k -tuple $\langle obs_1, \dots, obs_k \rangle$ corresponding to a possible content of the memory of the controller, by:

$$\mathcal{S}_k(\langle obs_1, \dots, obs_k \rangle) \stackrel{\text{def}}{=} \{ \sigma \in \Sigma_c \mid \langle obs_1, \dots, obs_k \rangle \in \mathcal{F}_k(\sigma, I(\text{Bad})) \} \quad (4.2)$$

It means that σ is forbidden after the sequence of observations $\langle obs_1, \dots, obs_k \rangle$ if $I(Bad)$ is reachable by a transition labeled by σ after this sequence.

- the set $E_k \stackrel{\text{def}}{=} I(Bad)$. It prevents the controlled system from beginning its execution in a state which can lead to Bad by a sequence of uncontrollable transitions.

The set $I(Bad)$ and the function \mathcal{F}_k are computed offline. Next, during the execution of the system, when the controller receives a new observation, it updates the content of its memory and computes online the function \mathcal{S}_k from this content to obtain the actions to be forbidden.

Example 4.2

In this example, we illustrate the computation of the k -memory controller and we show that it provides a better control than the memoryless controller.

We consider the system defined in Example 3.5 where, as a reminder, (i) $Bad = \{\langle CX, x, x', y, y' \rangle \mid (x \leq 1000) \wedge (y \in [0, 2])\}$, (ii) $I(Bad) = Bad \cup \{\langle CX, x, x', y, y' \rangle \mid [(x \leq 1001) \wedge (y \in [1, 2])] \vee [(x \leq 1002) \wedge (y = 2)]\}$, and (iii) the controller cannot distinguish a state $\langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$ from the states in the set $\{\langle \ell, x_1, x', y, y' \rangle \mid x_1 \in [x - 1, x + 1]\}$.

The computation of the function \mathcal{F}_k for a 2-memory controller gives the following values: (i) $\mathcal{F}_k(Stop_prod, I(Bad)) = \langle M(\{\langle \ell, x, x', y, y' \rangle \mid [(\ell = \text{PX}) \wedge (x \leq 1001)] \vee [(\ell = \text{Choice}) \wedge (x \leq 1002)]\}), M(\{\langle \text{PX}, x, x', y, y' \rangle \mid x \leq 1002\}) \rangle$, and (ii) $\mathcal{F}_k(\sigma, I(Bad)) = \emptyset$ for each $\sigma \in \Sigma \setminus \{Stop_prod\}$.

Therefore, the supervisory function \mathcal{S}_k forbids the action $Stop_prod$ if the current state of the system belongs to the set $\{\langle \text{PX}, x, x', y, y' \rangle \mid x \leq 1003\}$ and if the previous state of the system belongs to the set $\{\langle \ell, x, x', y, y' \rangle \mid [(\ell = \text{PX}) \wedge (x \leq 1002)] \vee [(\ell = \text{Choice}) \wedge (x \leq 1003)]\}$.

This 2-memory controller is *strictly* more permissive than the memoryless controller \mathcal{C} computed in Example 3.5, which always forbids the action $Stop_prod$ when the current state of the system belongs to the set $\{\langle \text{PX}, x, x', y, y' \rangle \mid x \leq 1003\}$. In Proposition 4.4, we prove that a k -memory controller is always more permissive than a memoryless controller.

The following property proves that our algorithm synthesizes correct controllers for the memory basic problem:

Proposition 4.1

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the controller $\mathcal{C}_k = \langle \mathcal{S}_k, E_k \rangle$, defined in (4.1), solves the memory basic problem (when $\Theta_0 \not\subseteq E_k$).

Proof

We prove by induction on the length n of the sequences of transitions (these sequences begin in an initial state) that $\left(\text{Reachable}_{\Delta/\mathcal{C}_k}^{\mathcal{T}/\mathcal{C}_k} ((\Theta_0)_{/\mathcal{C}_k}) \Big|_{[1]} \right) \cap I(Bad) = \emptyset$. Since $Bad \subseteq I(Bad)$, this property implies that $\left(\text{Reachable}_{\Delta/\mathcal{C}_k}^{\mathcal{T}/\mathcal{C}_k} ((\Theta_0)_{/\mathcal{C}_k}) \Big|_{[1]} \right) \cap Bad = \emptyset$.

- *Base case* ($n = 0$): the execution of the controlled system $\mathcal{T}/\mathcal{C}_k$ starts in a state which does not belong to $I(Bad)$, since $(\Theta_0)_{/\mathcal{C}_k} \Big|_{[1]} = \Theta_0 \setminus E_k = \Theta_0 \setminus I(Bad)$.
- *Induction step*: we suppose that the proposition holds for the sequences of transitions of length less than or equal to n and we prove that this property remains true for the sequences of transitions of length $n+1$. By induction hypothesis, each state \vec{v}_1 reachable by a sequence of transitions of length n does not belong to $I(Bad)$ and we show that each transition $\delta = \langle \sigma, G, A \rangle \in \Delta$, which can lead to a state $\vec{v}_2 \in I(Bad)$ from this state \vec{v}_1 in the system \mathcal{T} , cannot be fired from \vec{v}_1 in the controlled system $\mathcal{T}/\mathcal{C}_k$. For that, we consider two cases :
 - if $\delta \in \Delta_c$, then $M(\vec{v}_1) \subseteq M(\text{Pre}_\delta^{\mathcal{T}}(I(Bad)) \setminus I(Bad))$, because $\vec{v}_1 \in (\text{Pre}_\delta^{\mathcal{T}}(I(Bad)) \setminus I(Bad))$. This transition cannot be fired from \vec{v}_1 in the controlled system $\mathcal{T}/\mathcal{C}_k$, because by definition of \mathcal{F}_k (see Algorithm 4) and by definition of \mathcal{S}_k (see (4.2)), σ is forbidden after each k -tuple $\langle obs_1, \dots, obs_k \rangle$ which ends by an observation $obs_k \in M(\vec{v}_1)$ and which corresponds to a possible execution of $\mathcal{T}/\mathcal{C}_k$.
 - if $\delta \in \Delta_{uc}$, then $\vec{v}_2 \in I(Bad)$ (by (3.1)), which is impossible by hypothesis.

Hence, in the controlled system, the forbidden state \vec{v}_2 cannot be reached from the state \vec{v}_1 by the transition δ .

4.1.3 Effective Algorithm for the Memory Basic Problem

As in chapter 3, an effective algorithm can be obtained by using abstract interpretation techniques. More precisely, we first compute an overapproximation $I'(Bad)$ of $I(Bad)$ (see section 3.4.2 for the details). Next, we transpose the function \mathcal{F}_k (more precisely, we transpose Algorithm 4) into the abstract lattice and we obtain the abstract function \mathcal{F}_k^\sharp (the computation of this function may thus require overapproximations). Finally, we define our controller $\mathcal{C}_k = \langle \mathcal{S}_k, I'(Bad) \rangle$ as in (4.1) by using $I'(Bad)$ and \mathcal{F}_k^\sharp instead of $I(Bad)$ and \mathcal{F}_k .

4.2 Synthesis of Online Controllers

In this section, we present a method which synthesizes an *online* controller for the state avoidance control problem. First, in section 4.2.1, we define the framework that we use. Next, in section 4.2.2, we define a *semi-algorithm* which computes an online controller for the state avoidance control problem. Finally, in section 4.2.3, we explain how to extend this semi-algorithm by using abstract interpretation to obtain an *effective* algorithm.

4.2.1 Framework and State Avoidance Control Problem

The framework is quite similar to the one defined in section 3.3; the only changes concern the definition of the controller and hence the definition of the controlled system. In this framework, an online controller can compute (and retain) an estimate of the current state of the system and use this information to choose the actions to be forbidden. It is formally defined as follows:

Definition 4.3 (Online Controller)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, and an observer $\langle Obs, M \rangle$, an *online controller* with partial observation for the system \mathcal{T} is a pair $\mathcal{C}_o = \langle \mathcal{S}_o, E_o \rangle$, where:

- $\mathcal{S}_o : 2^{\mathcal{D}_V} \rightarrow 2^{\Sigma_c}$ is a supervisory function which defines, for each subset $P_i \subseteq \mathcal{D}_V$, a set $\mathcal{S}_o(P_i)$ of controllable actions that have to be forbidden when P_i is the estimate of the current state of the system computed by the controller.
- $E_o \subseteq \mathcal{D}_V$ is a set of states, which restricts the set of initial states of \mathcal{T} .

The estimation P_i of the possible current states of the system is reevaluated by the controller at each step of the execution of the system and its computation is detailed in section 4.2.2.

A system \mathcal{T} under the control of a controller \mathcal{C}_o cannot be characterized by a single STS as in the memoryless and k -memory cases. Indeed, the controller \mathcal{C}_o computes the forbidden actions from the state estimate P_i , whose value changes during the execution of \mathcal{T} ; thus this computation depends on the history of the execution of \mathcal{T} . We have then decided to formalize the behavior of the system \mathcal{T} under the control of \mathcal{C}_o by using the concept of *controlled executions* which characterizes the executions that can occur in this system:

Definition 4.4 (Controlled Execution)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$, an online controller $\mathcal{C}_o = \langle \mathcal{S}_o, E_o \rangle$, and an execution $s = \vec{v}_1 \xrightarrow{\langle \sigma_2, G_2, A_2 \rangle} \vec{v}_2 \xrightarrow{\langle \sigma_3, G_3, A_3 \rangle} \dots \xrightarrow{\langle \sigma_n, G_n, A_n \rangle} \vec{v}_n$ of \mathcal{T} , s is a *controlled execution* of the system \mathcal{T} under the control of \mathcal{C}_o if it satisfies the following conditions:

- $\vec{v}_1 \in \Theta_0 \setminus E_o$
- when the system arrives in the state \vec{v}_i ($\forall i \in [1, n - 1]$), the controller \mathcal{C}_o computes a state estimate P_i such that $\sigma_{i+1} \notin \mathcal{S}_o(P_i)$.

A controlled execution is thus an execution that can occur in the system \mathcal{T} under the control of \mathcal{C}_o .

The problem, that we want to solve, is the basic problem adapted to this framework³. It is formally defined as follows:

Problem 4.2 (Online Basic State Avoidance Control Problem)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the *online basic state avoidance control problem* (*online basic problem* for short) consists in computing an online controller $\mathcal{C}_o = \langle \mathcal{S}_o, E_o \rangle$ such that (i) $\Theta_0 \setminus E_o \neq \emptyset$ and (ii) each controlled execution $\vec{v}_1 \xrightarrow{\delta_2} \vec{v}_2 \dots \xrightarrow{\delta_n} \vec{v}_n$ of the system \mathcal{T} under the control of \mathcal{C}_o does not reach Bad i.e., the state $\vec{v}_n \notin Bad$.

Since the memoryless controller is a particular case of the online controller (it corresponds to the case where the state estimate is given by the set of states

³We only consider this case to remain quite concise.

compatible with the current observation), all propositions proved in section 3.3.4 remain valid. In particular, the online basic problem is *undecidable*, which explains the use of abstract interpretation in section 4.2.3 to obtain an effective algorithm.

4.2.2 Semi-algorithm for the Online Basic Problem

Our algorithm, which synthesizes controllers for the online basic problem, is an extension of the one defined for the memoryless case and is also composed of two parts. However, in the online case, most of the computations are performed online. In particular, an estimate of the current state of the system is computed during the execution of the system, which allows the controller to have a better knowledge of the system and hence to define a more precise control policy.

Computation of $I(Bad)$. The first step consists in computing the set $I(Bad)$ of states leading uncontrollably to Bad . This set is given in (3.1).

Computation of the Controller \mathcal{C}_o . The second step consists in computing the controller \mathcal{C}_o whose aim is to forbid all controllable actions that can lead to a state in $I(Bad)$. The principle of our online algorithm is the following. When the controller receives a new observation obs_i , it computes the set P_i of states, in which the system could be, from (i) this observation, (ii) the set P_{i-1} of states in which the system could be at the previous step, and (iii) the set $\mathcal{S}_o(P_{i-1})$ of actions forbidden at the previous step. Next, it computes the set $\mathcal{S}_o(P_i)$ of actions to be forbidden: an action σ is forbidden if and only if $I(Bad)$ can be reached from P_i by a transition labeled by σ .

Now, we detail the computation of the state estimate P_i and the computation of the online controller \mathcal{C}_o .

Let obs_i be the i^{th} observation (with $i \geq 1$) that the controller received from the system, P_{i-1} be the set of states in which the system could be at the previous step (the set P_{i-1} is only used when $i \geq 2$ and is defined in (4.3)), and $\mathcal{S}_o(P_{i-1})$ be the set of actions forbidden at the previous step (the set $\mathcal{S}_o(P_{i-1})$ is defined in (4.5)), then the set P_i of states, in which the system could be, is defined as follows:

$$P_i \stackrel{\text{def}}{=} \begin{cases} M^{-1}(obs_i) \cap \text{Post}_{\Sigma \setminus \mathcal{S}_o(P_{i-1})}^{\tau}(P_{i-1}) & \text{if } i \geq 2 \\ (\Theta_0 \setminus E_o) \cap M^{-1}(obs_1) & \text{if } i = 1 \end{cases} \quad (4.3)$$

As a reminder, E_o is the set of initial states that the online controller forbids and $\text{Post}_{\Sigma \setminus \mathcal{S}_o(P_{i-1})}^T(P_{i-1})$ gives the set of states that are reachable from P_{i-1} by firing exactly one transition allowed by the controller.

The online controller is formally defined in the following way:

$$\mathcal{C}_o \stackrel{\text{def}}{=} \langle \mathcal{S}_o, E_o \rangle \quad (4.4)$$

with the elements \mathcal{S}_o and E_o defined as follows:

1. the supervisory function \mathcal{S}_o is given, for each state estimate $P_i \subseteq \mathcal{D}_V$, by:

$$\mathcal{S}_o(P_i) \stackrel{\text{def}}{=} \{\sigma \in \Sigma_c \mid P_i \cap \text{Pre}_\sigma^T(I(\text{Bad})) \neq \emptyset\} \quad (4.5)$$

It means that σ is forbidden for the set P_i if and only if $I(\text{Bad})$ can be reached from P_i by a transition labeled by σ .

2. the set $E_o \stackrel{\text{def}}{=} I(\text{Bad})$. It prevents the system from beginning its execution in a state which can lead to Bad by a sequence of uncontrollable transitions.

The set $I(\text{Bad})$ is computed *offline*. Next, during the execution of the system, when the controller receives a new observation from the system, it first computes *online* the new state estimate P_i by using (4.3) and then, based on this information, it computes *online* the set $\mathcal{S}_o(P_i)$ of actions to be forbidden by using (4.5). Since Σ is finite, $\mathcal{S}_o(P_i)$ is computable when $I(\text{Bad})$ is computed.

Example 4.3

In this example, we illustrate the computation of the online controller and we show that it provides a better control than the k -memory controller (for any fixed value k) for the execution that we will consider. This example is parametrized by the value k .

We consider the system depicted in Figure 3.1 and the controller must ensure, during the phase of consumption of the parts X , the presence of a minimal number of parts of this kind: $\text{Bad} = \{\langle \text{CX}, x, x', y, y' \rangle \mid (x \leq k) \wedge (y \in [0, 2])\}$. The controller does not observe the variable x i.e., for each state $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, the set of states, which are undistinguishable from \vec{v} , is given by $\{\langle \ell, x_1, x', y, y' \rangle \mid x_1 \in \mathbb{Z}\}$. The computation of the set $I(\text{Bad})$ gives:

$$\begin{aligned} I(\text{Bad}) = & \text{Bad} \cup \{\langle \text{CX}, x, x', y, y' \rangle \mid [(x \leq k + 1) \wedge (y \in [1, 2])] \\ & \vee [(x \leq k + 2) \wedge (y = 2)]\} \end{aligned}$$

Let us consider the following execution: $\langle \text{Choice}, 0, 0, 0, 0 \rangle \xrightarrow{\delta_9} \langle \text{PX}, 0, 0, 0, 0 \rangle \xrightarrow{\delta_3} \langle \text{PX}, 1, 0, 0, 0 \rangle \xrightarrow{\delta_3} \langle \text{PX}, 2, 0, 0, 0 \rangle \xrightarrow{\delta_3} \langle \text{PX}, 3, 0, 0, 0 \rangle \xrightarrow{\delta_4} \langle \text{CX}, 3, 0, 2, 0 \rangle \xrightarrow{\delta_2} \langle \text{Choice}, 3, 0, 2, 0 \rangle \xrightarrow{\delta_9} \langle \text{PX}, 3, 0, 2, 0 \rangle \xrightarrow{\delta_3} \langle \text{PX}, 4, 0, 2, 0 \rangle \xrightarrow{\delta_3} \dots \xrightarrow{\delta_3} \langle \text{PX}, k + 3, 0, 2, 0 \rangle$. At each step of this execution, the online controller computes an estimate P_i which is equal to the current state. Thus, when the system arrives in the state $\langle \text{PX}, k + 3, 0, 2, 0 \rangle$, the controller computes an estimate $P_i = \{\langle \text{PX}, k + 3, 0, 2, 0 \rangle\}$ and it allows the action *Stop_prod*, because this action does not lead to a forbidden state from the state $\langle \text{PX}, k + 3, 0, 2, 0 \rangle$. For the k -memory controller, when the system arrives in the state $\langle \text{PX}, k + 3, 0, 2, 0 \rangle$, the content of its memory is equal to $\langle M(\langle \text{PX}, 4, 0, 2, 0 \rangle), \dots, M(\langle \text{PX}, k + 3, 0, 2, 0 \rangle) \rangle$. It then forbids the action *Stop_prod*, because the sequence of states $\langle \text{PX}, 3, 0, 2, 0 \rangle, \dots, \langle \text{PX}, k + 2, 0, 2, 0 \rangle$ leads to the forbidden state $\langle \text{CX}, k + 2, 0, 2, 0 \rangle$ by *Stop_prod* and also $\langle M(\langle \text{PX}, 3, 0, 2, 0 \rangle), \dots, M(\langle \text{PX}, k + 2, 0, 2, 0 \rangle) \rangle = \langle M(\langle \text{PX}, 4, 0, 2, 0 \rangle), \dots, M(\langle \text{PX}, k + 3, 0, 2, 0 \rangle) \rangle$. Thus, the online controller is strictly more permissive than the k -memory controller for the sequence considered in this example. In Proposition 4.5, we prove that it is always more permissive than the k -memory controller.

The following property proves that our algorithm synthesizes correct controllers for the online basic problem:

Proposition 4.2

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle \text{Obs}, M \rangle$ and a predicate *Bad*, which represents a set of forbidden states, the controller $\mathcal{C}_o = \langle \mathcal{S}_o, E_o \rangle$, defined by (4.4), solves the online basic problem (when $\Theta_0 \not\subseteq E_o$).

Proof

We prove by induction on the length n of the sequences of transitions (these sequences begin in an initial state) that $I(\text{Bad})$ (and thus *Bad*) is not reachable in the system \mathcal{T} under the control of \mathcal{C}_o :

- *Base case* ($n = 0$): the execution of the system \mathcal{T} under the control of \mathcal{C}_o starts in a state which belongs to $(\Theta_0 \setminus E_o) \cap M^{-1}(\text{obs}_1) = (\Theta_0 \setminus I(\text{Bad})) \cap M^{-1}(\text{obs}_1)$ (where obs_1 is the first observation received from the system). Thus, this system avoids $I(\text{Bad})$.

- *Induction step:* we suppose that the proposition holds for the sequences of transitions of length less than or equal to n and we prove that this property remains true for the sequences of transitions of length $n + 1$. The state estimate computed after a sequence of transitions of length i (with $i \geq 0$) is denoted by P_{i+1} . By induction hypothesis, each state \vec{v}_1 reachable by a sequence of transitions of length n (thus $\vec{v}_1 \in P_{n+1}$) does not belong to $I(Bad)$ and we have to show that each transition $\delta = \langle \sigma, G, A \rangle \in \Delta$, which can lead to a state $\vec{v}_2 \in I(Bad)$ from this state \vec{v}_1 in the system \mathcal{T} , cannot be fired from \vec{v}_1 in the system \mathcal{T} under the control of \mathcal{C}_o . For that, we consider two cases :
 - if $\delta \in \Delta_c$, then this transition cannot be fired from \vec{v}_1 in the system \mathcal{T} under the control of \mathcal{C}_o . Indeed, $\vec{v}_1 \in \text{Pre}_\sigma^{\mathcal{T}}(I(Bad))$ (because $\vec{v}_1 \in \text{Pre}_\sigma^{\mathcal{T}}(\vec{v}_2)$ and $\vec{v}_2 \in I(Bad)$), which implies that $\sigma \in \mathcal{S}_o(P_{n+1})$ (because $\vec{v}_1 \in (P_{n+1} \cap \text{Pre}_\sigma^{\mathcal{T}}(I(Bad)))$).
 - if $\delta \in \Delta_{uc}$, then $\vec{v}_2 \in I(Bad)$ (by (3.1)), which is impossible by hypothesis.

Hence, in the system \mathcal{T} under the control of \mathcal{C}_o , the forbidden state \vec{v}_2 cannot be reached from the state \vec{v}_1 by the transition δ .

4.2.3 Effective Algorithm for the Online Basic Problem

As in chapter 3, an effective algorithm can be obtained by using abstract interpretation techniques. For that, we first compute an overapproximation $I'(Bad)$ of $I(Bad)$ (see section 3.4.2 for the details). Next, we define our online controller $\mathcal{C}_o = \langle \mathcal{S}_o, I'(Bad) \rangle$ as in (4.4) by using $I'(Bad)$ and the state estimates P_i^\sharp instead of $I(Bad)$ and P_i . The state estimates P_i^\sharp are obtained by transposing the function (see (4.3)) defining P_i into the abstract lattice (thus the computation of P_i^\sharp may require overapproximations).

4.3 Comparisons between the Memoryless, k -Memory and Online Controllers

Up to now, we have proposed three different controllers for the basic problem: the memoryless, k -memory, and online controllers. In this section, we theoret-

ically compare the permissiveness of these three controllers and we discuss the results obtained.

Memoryless and k -memory Controllers. We first prove that the k -memory controllers are more permissive than the memoryless controllers. For that, we prove the more general property which follows:

Proposition 4.3

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$, a set Bad (which represents a set of forbidden states), and two integers k, k' such that $k' > k \geq 1$, the k' -memory controller $\mathcal{C}'_{k'} = \langle \mathcal{S}'_{k'}, E'_{k'} \rangle$ is more permissive than the k -memory controller $\mathcal{C}_k = \langle \mathcal{S}_k, E_k \rangle$, where $\mathcal{C}'_{k'}$ and \mathcal{C}_k are defined in (4.1).

Proof

In this proof, I denotes $I(Bad)$. We first suppose that the k' -memory controller receives more than k' observations. Let $\langle obs_1, \dots, obs_{k'} \rangle \in (\mathcal{D}_{Obs})^{k'}$ be the last k' observations received from the system. We prove that, for any action $\sigma \in \Sigma_c$, if the k' -memory controller forbids this action σ after this sequence of observations, then the k -memory controller also forbids this action after this sequence, which proves the property. If σ is forbidden by the k' -memory controller, we have that $\sigma \in \mathcal{S}'_{k'}(\langle obs_1, \dots, obs_{k'} \rangle)$ and hence $\langle obs_1, \dots, obs_{k'} \rangle \in \mathcal{F}'_{k'}(\sigma, I)$. By definition of Algorithm 4, there exist k' transitions $\delta_1, \dots, \delta_{k'}$ for which Algorithm 4 generates a k' -tuple Tmp'_1 (see line 25) of sets of observations which contains $\langle obs_1, \dots, obs_{k'} \rangle$. We only keep the last k observations of the sequence $\langle obs_1, \dots, obs_{k'} \rangle$ i.e., we only keep $\langle obs_{1+k'-k}, \dots, obs_{k'} \rangle$. Algorithm 4 generates for the k transitions $\delta_{1+k'-k}, \dots, \delta_{k'}$ a k -tuple Tmp_1 (see line 25) of sets of observations which contains $\langle obs_{1+k'-k}, \dots, obs_{k'} \rangle$. Therefore, $\langle obs_{1+k'-k}, \dots, obs_{k'} \rangle \in \mathcal{F}_k(\sigma, I)$ and thus $\sigma \in \mathcal{S}_k(\langle obs_{1+k'-k}, \dots, obs_{k'} \rangle)$. Now, we consider the case where the k' -memory controller received strictly less than k' observations i.e., the memory of $\mathcal{C}'_{k'}$ contains $\langle \Gamma, \dots, \Gamma, obs_1, \dots, obs_j \rangle$ (with $j < k'$). Again, we prove that, for any action $\sigma \in \Sigma_c$, if the k' -memory controller forbids this action σ after this sequence of observations, then the k -memory controller also forbids this action after this sequence, which proves the property. If σ is forbidden by the k' -memory controller, we have that $\sigma \in \mathcal{S}'_{k'}(\langle \Gamma, \dots, \Gamma, obs_1, \dots, obs_j \rangle)$ and hence $\langle \Gamma, \dots, \Gamma, obs_1, \dots, obs_j \rangle \in \mathcal{F}'_{k'}(\sigma, I)$. By definition of Algorithm 4, there exist k' transitions $\delta_1, \dots, \delta_{k'}$ for which Algorithm 4 generates a k' -tuple Tmp'_2 (see line 24) of sets of observations which contains $\langle \Gamma, \dots, \Gamma, obs_1, \dots, obs_j \rangle$ and we consider two cases:

- *either* $j < k$: Algorithm 4 generates for the k transitions $\delta_{1+k'-k}, \dots, \delta_{k'}$ a k -tuple Tmp_2 (see line 24) of sets of observations which contains $\langle \Gamma, \dots, \Gamma, obs_1, \dots, obs_j \rangle$. Therefore, $\langle \Gamma, \dots, \Gamma, obs_1, \dots, obs_j \rangle \in \mathcal{F}_k(\sigma, I)$ and thus $\sigma \in \mathcal{S}_k(\langle \Gamma, \dots, \Gamma, obs_1, \dots, obs_j \rangle)$.
- *or* $j \geq k$: Algorithm 4 generates for the k transitions $\delta_{1+k'-k}, \dots, \delta_{k'}$ a k -tuple Tmp_1 (see line 25) of sets of observations which contains $\langle obs_{1+j-k}, \dots, obs_j \rangle$. Therefore, $\langle obs_{1+j-k}, \dots, obs_j \rangle \in \mathcal{F}_k(\sigma, I)$ and thus $\sigma \in \mathcal{S}_k(\langle obs_{1+j-k}, \dots, obs_j \rangle)$.

Thus, increasing the size k of the memory of the k -memory controller gives a more permissive solution, but it also increases the time complexity to compute it. Indeed, Algorithm 4 enumerates all sequences of transitions of length k to compute the function \mathcal{F}_k and thus it enumerates $|\Delta|^k$ sequences. Therefore, when the size k of the memory increases, the time complexity of Algorithm 4 also increases. This proposition also holds when abstract interpretation techniques are used to obtain effective algorithms. Indeed, we can remark that the proof of this proposition remains correct when the overapproximations, induced by abstract interpretation, are taken into account.

From Proposition 4.3, we can deduce the following property:

Proposition 4.4

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$, a set Bad (which represents a set of forbidden states), and an integer $k \geq 1$, the k -memory controller $\mathcal{C}_k = \langle \mathcal{S}_k, E_k \rangle$ (defined in (4.1)) is more permissive than the memoryless controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ (defined in (3.3)).

Moreover, in Example 4.2, we shown that the k -memory controller (with $k \geq 2$) can be strictly more permissive than the memoryless controller.

Thus, the k -memory controller has a better control policy than the memoryless controller. However, it requires additional offline computations to compute the function \mathcal{F}_k . Indeed, as explained above, the time complexity of the computation of \mathcal{F}_k increases when k increases. Similarly, for the online part of these methods, the membership test of the k -memory controller has a greater time complexity than the one of the memoryless controller (it is k times greater). The choice between these two controllers thus depends on the requirements of the user. The k -memory controller must be used when the aim is to obtain a solution of better quality and the memoryless controller must be used when the

purpose is to quickly compute a solution.

k -Memory and Online Controllers. Now, we prove that the online controllers are more permissive than the k -memory controllers:

Proposition 4.5

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$, a set Bad (which represents a set of forbidden states), and an integer $k \geq 1$, the online controller $\mathcal{C}_o = \langle \mathcal{S}_o, E_o \rangle$ (defined in (4.4)) is more permissive than the k -memory controller $\mathcal{C}_k = \langle \mathcal{S}_k, E_k \rangle$ (defined in (4.1)).

Proof

In this proof, I denotes $I(Bad)$. Let obs_1, \dots, obs_j be the sequence of observations received by the controllers from the system since the beginning of the execution of the system. By Proposition 4.2, we have, for each $i \in [1, j]$, that:

$$P_i \cap I(Bad) = \emptyset \tag{4.6}$$

where P_i ($\forall i \geq 1$) denotes the state estimate computed by \mathcal{C}_o after the reception of the observation obs_i .

We prove that, for any action $\sigma \in \Sigma_c$, if the online controller forbids this action σ after the sequence obs_1, \dots, obs_j of observations, then the k -memory controller also forbids this action after this sequence, which proves the property.

We first consider the case where $j \geq k$. We suppose that σ is forbidden by the online controller after the sequence obs_1, \dots, obs_j and we have then that $\sigma \in \mathcal{S}_o(P_j)$. We will prove that there exist k transitions $\delta_1, \dots, \delta_k$ (with $\delta_k \in \text{Trans}(\sigma)$) for which Algorithm 4 generates a k -tuple Tmp_1 (line 25) of sets of observations that contains $\langle obs_{j-k+1}, \dots, obs_j \rangle$. First, we prove that $obs_j \in Tmp_1 \Big|_{[k]}$:

$$\begin{aligned} & \sigma \in \mathcal{S}_o(P_j) \\ \Rightarrow & P_j \cap \text{Pre}_\sigma^T(I) \neq \emptyset, \text{ by (4.5)} \\ \Rightarrow & \exists \vec{v}_1 \in P_j, \exists \delta_k \in \text{Trans}(\sigma) : \vec{v}_1 \in \text{Pre}_{\delta_k}^T(I) \\ \Rightarrow & \exists \vec{v}_1 \in P_j, \exists \delta_k \in \text{Trans}(\sigma) : \vec{v}_1 \in (\text{Pre}_{\delta_k}^T(I) \setminus I), \text{ because } \vec{v}_1 \notin I \\ & \text{by (4.6)} \tag{\alpha} \\ \Rightarrow & obs_j \in M(\text{Pre}_{\delta_k}^T(I) \setminus I), \text{ because } obs_j \in M(\vec{v}_1) \text{ by (4.3)} \\ \Rightarrow & obs_j \in Tmp_1 \Big|_{[k]}, \text{ by definition of } Tmp_1 \text{ in Algorithm 4} \end{aligned}$$

Next, we prove that $obs_{j-1} \in Tmp_1 \upharpoonright_{[k-1]}$. By (α) , we know that $\exists \vec{v}_1 \in P_j, \exists \delta_k \in \text{Trans}(\sigma) : \vec{v}_1 \in (\text{Pre}_{\delta_k}^T(I) \setminus I)$ and, by reasoning on this state \vec{v}_1 , we can deduce the following information:

$$\begin{aligned}
& \vec{v}_1 \in P_j \\
\Rightarrow & \vec{v}_1 \in \text{Post}_{\Sigma \setminus \mathcal{S}_o(P_{j-1})}^T(P_{j-1}), \text{ by (4.3)} \\
\Rightarrow & \exists \vec{v}_2 \in P_{j-1} : \vec{v}_1 \in \text{Post}_{\Sigma \setminus \mathcal{S}_o(P_{j-1})}^T(\vec{v}_2) \\
\Rightarrow & \exists \vec{v}_2 \in P_{j-1} : \vec{v}_1 \in \text{Post}_{\Sigma}^T(\vec{v}_2), \text{ because } \Sigma \setminus \mathcal{S}_o(P_{j-1}) \subseteq \Sigma \\
\Rightarrow & \exists \vec{v}_2 \in P_{j-1} : \vec{v}_2 \in \text{Pre}_{\Sigma}^T(\vec{v}_1) \\
\Rightarrow & \exists \vec{v}_2 \in P_{j-1}, \exists \delta_{k-1} \in \Delta : \vec{v}_2 \in \text{Pre}_{\delta_{k-1}}^T(\vec{v}_1), \text{ because } \delta_{k-1} \in \text{Trans}(\Sigma) \\
\Rightarrow & \exists \vec{v}_2 \in P_{j-1}, \exists \delta_{k-1} \in \Delta : \vec{v}_2 \in \text{Pre}_{\delta_{k-1}}^T(\text{Pre}_{\delta_k}^T(I) \setminus I), \text{ because} \\
& \vec{v}_1 \in (\text{Pre}_{\delta_k}^T(I) \setminus I) \text{ by } (\alpha). \\
\Rightarrow & \exists \vec{v}_2 \in P_{j-1}, \exists \delta_{k-1} \in \Delta : \vec{v}_2 \in (\text{Pre}_{\delta_{k-1}}^T(\text{Pre}_{\delta_k}^T(I) \setminus I) \setminus I), \text{ because} \\
& \vec{v}_2 \notin I \text{ by (4.6)} \tag{\beta} \\
\Rightarrow & obs_{j-1} \in M(\text{Pre}_{\delta_{k-1}}^T(\text{Pre}_{\delta_k}^T(I) \setminus I) \setminus I), \text{ because } obs_{j-1} \in M(\vec{v}_2) \\
& \text{by (4.3)} \\
\Rightarrow & obs_{j-1} \in Tmp_1 \upharpoonright_{[k-1]}, \text{ by definition of } Tmp_1 \text{ in Algorithm 4}
\end{aligned}$$

By (β) , we know that $\exists \vec{v}_2 \in P_{j-1}, \exists \delta_{k-1} \in \Delta : \vec{v}_2 \in (\text{Pre}_{\delta_{k-1}}^T(\text{Pre}_{\delta_k}^T(I) \setminus I) \setminus I)$ and we can prove similarly as above that (i) $obs_{j-2} \in Tmp_1 \upharpoonright_{[k-2]}$ and (ii) $\exists \vec{v}_3 \in P_{j-2}, \exists \delta_{k-2} \in \Delta : \vec{v}_3 \in [\text{Pre}_{\delta_{k-2}}^T(\text{Pre}_{\delta_{k-1}}^T(\text{Pre}_{\delta_k}^T(I) \setminus I) \setminus I) \setminus I]$. Then, with the same reasoning, we prove that $obs_{j-i} \in Tmp_1 \upharpoonright_{[k-i]}$ and $\exists \vec{v}_{i+1} \in P_{j-i}, \exists \delta_{k-i} \in \Delta : \vec{v}_{i+1} \in [\text{Pre}_{\delta_{k-i}}^T(\dots(\text{Pre}_{\delta_k}^T(I) \setminus I)\dots) \setminus I]$ (for $i = 3, 4, \dots, k-1$). Thus, we have proven that $obs_{j-i} \in Tmp_1 \upharpoonright_{[k-i]}$ ($\forall i \in [0, k-1]$). Therefore, the last k observations $obs_{j-k+1}, \dots, obs_j$ of the sequence obs_1, \dots, obs_j are such that $\langle obs_{j-k+1}, \dots, obs_j \rangle \in Tmp_1$ and thus $\langle obs_{j-k+1}, \dots, obs_j \rangle \in \mathcal{F}_k(\sigma, I(\text{Bad}))$ (by definition of \mathcal{F}_k). This implies that $\sigma \in \mathcal{S}_k(\langle obs_{j-k+1}, \dots, obs_j \rangle)$. The action σ is thus forbidden by the k -memory controller after the sequence obs_1, \dots, obs_j . For the case where $j < k$, we can prove as above that $\sigma \in \mathcal{S}_k(\langle \Gamma, \dots, \Gamma, obs_1, \dots, obs_j \rangle)$.

Moreover, in Example 4.3, we showed that the online controller can be strictly more permissive than the k -memory controller.

The online controller has a better control policy than the k -memory controller

and its offline part requires less computations than the one of the k -memory controller. However, it requires significant online computations to choose the actions to be forbidden. Indeed, the online part of the k -memory method only requires a membership test (see (4.2)), whereas the one of the online method requires the computation of several operations having a polynomial time complexity⁴ (see (4.3) and (4.5)). In particular, if the time to choose the actions to be forbidden is short, we will not always be able to use the online controller. We must then use the k -memory controller or the memoryless controller.

Unfortunately, our approach, based on abstract interpretation, does not ensure the validity of this proposition since the algorithms, which synthesize the online and k -memory controllers, are quite different. So, if, for example, our effective online algorithm computes rough state estimates (e.g., $P_i = \mathcal{D}_V$) and our effective k -memory algorithm does not use overapproximations in its computations, then our online controller will be less permissive than our k -memory controller.

Full Observation. Under full observation (that corresponds to the case where $\mathcal{D}_{Obs} = \mathcal{D}_V$ and $M = \text{Id}_{\mathcal{D}_V}$), we have that the memoryless, k -memory and online controllers have the same permissiveness.

Proposition 4.6

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, an observer $\langle Obs, M \rangle$ (where $\mathcal{D}_{Obs} = \mathcal{D}_V$ and $M = \text{Id}_{\mathcal{D}_V}$), a set Bad (which represents a set of forbidden states), and an integer $k \geq 1$, the memoryless controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ (defined in (3.3)), the k -memory controller $\mathcal{C}_k = \langle \mathcal{S}_k, E_k \rangle$ (defined in (4.1)) and the online controller $\mathcal{C}_o = \langle \mathcal{S}_o, E_o \rangle$ (defined in (4.4)) have the same permissiveness.

Proof

We first prove that the memoryless and online controllers have the same permissiveness. For these two controllers, the computation of the set $I(Bad)$ gives the same result. Next, when the system arrives in a state \vec{v} , the memoryless controller receives this state as information and it forbids the set $\mathcal{S}(\vec{v})$ of actions. By (3.4), this set $\mathcal{S}(\vec{v}) = \{\sigma \in \Sigma_c \mid \vec{v} \in (\text{Pre}_\sigma^{\mathcal{T}}(I(Bad)) \setminus I(Bad))\}$ i.e., the memoryless controller forbids an action σ in the state \vec{v} if there exists a

⁴More precisely, the online part of the online method requires to compute set intersections, set differences and the functions Post , Pre and M^{-1} . These operations can be computed in polynomial time.

transition labeled by σ which leads to $I(Bad)$ from \vec{v} . When the online controller receives the information \vec{v} from the system, it computes a state estimate $P_i = \{\vec{v}\}$ and it forbids the set $\mathcal{S}_o(\vec{v})$ of actions. This set $\mathcal{S}_o(\vec{v}) = \mathcal{S}(\vec{v})$, because:

$$\begin{aligned}
 \mathcal{S}_o(\vec{v}) &= \{\sigma \in \Sigma_c \mid \{\vec{v}\} \cap \text{Pre}_\sigma^T(I(Bad)) \neq \emptyset\}, \text{ by (4.5)} \\
 &= \{\sigma \in \Sigma_c \mid \vec{v} \in \text{Pre}_\sigma^T(I(Bad))\} \\
 &= \{\sigma \in \Sigma_c \mid \vec{v} \in (\text{Pre}_\sigma^T(I(Bad)) \setminus I(Bad))\}, \text{ because } \vec{v} \notin I(Bad) \\
 &\quad \text{by Proposition 4.2} \\
 &= \mathcal{S}(\vec{v})
 \end{aligned}$$

Thus, the memoryless and online controllers have the same control policy and hence they have the same permissiveness.

This property, Proposition 4.4 (i.e., $\mathcal{C} \preceq_p \mathcal{C}_k$), and Proposition 4.5 (i.e., $\mathcal{C}_k \preceq_p \mathcal{C}_o$) imply that the memoryless and k -memory controllers have the same permissiveness.

In conclusion, we have proven that the memoryless, k -memory, and online controllers have the same permissiveness.

Consequently, when the system is perfectly observed, it is preferable to synthesize a memoryless controller.

One can note that this proposition remains valid in our approach based on abstract interpretation.

Chapter 5

Synthesis of Decentralized and Modular Controllers for Infinite State Systems

 IN chapters 3 and 4, we were interested in the *centralized* control of infinite state systems (i.e., a *single* controller interacts with the system in order to restrict its behavior). However, in some cases, the nature of the system to be controlled makes unrealistic the use of such an approach (e.g., distributed systems). The *decentralized* framework is an alternative to the *centralized* framework and is more suitable for the control of distributed systems with synchronous communications. This framework has widely been studied in the past years and has shown its usefulness in several domains like manufacturing systems, communication network protocols, ... [RW92a]. In this approach, n local controllers interact with the system in a feedback manner to satisfy some control objectives. Each local controller has its own view of the system and its own set of controllable actions. Indeed, due to the distributed nature of the system to be controlled, the controllers observe the system through different sets of sensors (these sets can overlap) and control it by means of different sets of actuators (these sets can also overlap). Moreover, one supposes the existence of a *coordination mechanism* to define the *global* control to be applied to the system: each local controller defines its own set of actions to be forbidden (based on its

own view of the system) and the global control applied to the system is defined by a *fusion rule* which depends on the actions forbidden by each controller.

As explained above, the decentralized control is suitable for the control of distributed systems with synchronous communications, but it has also other advantages. It decreases the computational complexity of synthesizing controllers, because the overall task of synthesis is divided into synthesizing local controllers. It also provides more flexibility. Indeed, if a sensor must be replaced (because of a failure for example), only the controllers, which use it, must be recomputed.

In the decentralized framework, we have generally no information regarding the structure of the distributed system to be controlled while the computation of the controllers, which implies that these computations are performed on the model of the global system. However, if this structure is known and if the system is composed of several subsystems acting in parallel, it is of interest to exploit this knowledge to compute the set of controllers ensuring the expected properties without having to compute the model of the global system. Indeed, in this way, the classical problem of the state space explosion inherent to the parallel composition of subsystems is avoided. This might also have an impact when performing some fixpoint computations that are necessary to calculate the set of states that have to be avoided. In the sequel, this particular framework is called *modular framework* as opposed to the decentralized framework for which the structure of the system is not taken into account when performing the computation of the controllers.

The remainder of this chapter is structured as follows. First, in section 5.1, we define an algorithm which synthesizes decentralized controllers for the state avoidance control problem by using techniques similar to the ones used in chapters 3 and 4. Next, in section 5.2, we turn our attention to the modular approach and we define an algorithm synthesizing controllers for the state avoidance control problem. Finally, in section 5.3, we experimentally evaluate our methods.

5.1 Synthesis of Decentralized Controllers

In this section, we define an algorithm which synthesizes controllers for the state avoidance control problem in the decentralized framework.

The remainder of this chapter is structured as follows. First, in section 5.1.1, we formally define the decentralized framework and the problems that we want to solve. Next, in section 5.1.2, we propose an algorithm which synthesizes controllers for the state avoidance control problem. In section 5.1.3, we extend

this algorithm to the deadlock free case. Finally, in section 5.1.4, we theoretically compare the permissiveness of our centralized and decentralized controllers.

5.1.1 Framework and State Avoidance Control Problem

The system to be controlled is given by an STS. This system is controlled by n local controllers; each local controller has its own view of the system and its own set of controllable actions. The *global control* applied to \mathcal{T} is given by a fusion rule, which depends on the set of actions forbidden by each local controller.

Means of Observation. In the decentralized framework, each local controller \mathcal{C}_i ($\forall i \in [1, n]$) has its own *partial* view of the state space of the system, which is modeled by an observer (see Definition 3.8). Therefore, an observer $\langle Obs_i, M_i \rangle$ is associated with each controller \mathcal{C}_i and when the system arrives in a state \vec{v} , the controller \mathcal{C}_i receives an observation $obs_i \in M_i(\vec{v})$.

Remark 5.1

For more conciseness, we suppose throughout this chapter that the masks M_i correspond to *partitions* of the state space, but the results, that we present, also hold when the masks M_i correspond to coverings.

Means of Control. Each local controller \mathcal{C}_i ($\forall i \in [1, n]$) interacts with the system in a feedback manner: it receives an observation from the system and defines, from this information, a set of actions that it proposes to forbid. In this mechanism, the set of actions Σ is partitioned, for each controller \mathcal{C}_i , into the set of controllable actions $\Sigma_{i,c}$, that can be forbidden by \mathcal{C}_i , and the set of uncontrollable actions $\Sigma_{i,uc}$, that cannot be forbidden by \mathcal{C}_i . The subsets $\Sigma_{1,c}, \dots, \Sigma_{n,c}$ are not necessarily disjoint. The set of actions that can be controlled by at least one controller is denoted by Σ_c and is defined by $\Sigma_c \stackrel{\text{def}}{=} \bigcup_{i=1}^n \Sigma_{i,c}$ and the set of actions that cannot be controlled is denoted by Σ_{uc} and is defined by $\Sigma_{uc} \stackrel{\text{def}}{=} \Sigma \setminus \Sigma_c$. That also induces a partition of the set of transitions Δ into the set Δ_c (resp. $\Delta_{i,c}$) and the set Δ_{uc} (resp. $\Delta_{i,uc}$) in the following way: $\langle \sigma, G, A \rangle \stackrel{\text{def}}{\in} \Delta_c$ (resp. $\Delta_{i,c}$) if $\sigma \in \Sigma_c$ (resp. $\Sigma_{i,c}$), and $\langle \sigma, G, A \rangle \stackrel{\text{def}}{\in} \Delta_{uc}$ (resp. $\Delta_{i,uc}$) if $\sigma \in \Sigma_{uc}$ (resp. $\Sigma_{i,uc}$).

In the sequel, we use the function $\text{In} : \Sigma \rightarrow 2^{\{1, \dots, n\}}$ which gives, for each action $\sigma \in \Sigma$, the set of indices of the controllers that control σ i.e., $\text{In}(\sigma) \stackrel{\text{def}}{=} \{i \mid \sigma \in \Sigma_{i,c}\}$.

Decentralized Controller and Controlled System. A local controller has the possibility to forbid some actions and the aim is to restrict the behavior of the system by combining the control decisions of the n local controllers in order to ensure the desired property.

The formal definition of a local controller is given in Definition 3.9. The *global* control applied to the system is defined by a *fusion rule* which gives the actions and the initial states to be forbidden and which depends on the control decisions of each local controller. Within our framework, a fusion rule is formally defined as follows:

Definition 5.1 (Fusion Rule)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, n observers $\langle Obs_i, M_i \rangle$ ($\forall i \in [1, n]$) and n controllers $\mathcal{C}_i = \langle \mathcal{S}_i, E_i \rangle$ ($\forall i \in [1, n]$), a *fusion rule* for the actions and the initial states to be forbidden is a pair $\mathcal{R} \stackrel{\text{def}}{=} \langle \mathcal{R}^{\mathcal{S}}, \mathcal{R}^{\mathcal{E}} \rangle$, where:

- the function $\mathcal{R}^{\mathcal{S}} : 2^{\Sigma_{1,c}} \times \dots \times 2^{\Sigma_{n,c}} \rightarrow 2^{\Sigma_c}$ gives, for each set $B_i \subseteq \Sigma_{i,c}$ ($\forall i \in [1, n]$) of controllable actions that \mathcal{C}_i proposes to forbid, the set $\mathcal{R}^{\mathcal{S}}(B_1, \dots, B_n)$ of actions that the system cannot fire. Thus, this function defines the *global* control to be applied to the system by giving the actions that are effectively forbidden.
- the function $\mathcal{R}^{\mathcal{E}} : 2^{\mathcal{D}_V} \times \dots \times 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ gives, for each set $E_i \subseteq \mathcal{D}_V$ ($\forall i \in [1, n]$) of states that \mathcal{C}_i proposes to forbid at the beginning of the execution of the system, the set $\mathcal{R}^{\mathcal{E}}(E_1, \dots, E_n)$ of states that are effectively forbidden.

We explain further how we instantiate the fusion rule \mathcal{R} . Based on Definitions 3.9 and 5.1, a *decentralized controller* is defined as follows:

Definition 5.2 (Decentralized Controller)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, and n observers $\langle Obs_i, M_i \rangle$ ($\forall i \in [1, n]$), a *decentralized controller* is defined by n local controllers \mathcal{C}_i ($\forall i \in [1, n]$) coordinated by a fusion rule \mathcal{R} and is formally given by a pair $\langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$, where:

- $\mathcal{C}_i = \langle \mathcal{S}_i, E_i \rangle$ ($\forall i \in [1, n]$) is a local controller (see Definition 3.9) defined w.r.t. (i) the observer $\langle Obs_i, M_i \rangle$, (ii) the set of controllable actions $\Sigma_{i,c}$, and (iii) the set of uncontrollable actions $\Sigma_{i,uc}$.
- \mathcal{R} is a fusion rule (see Definition 5.1).

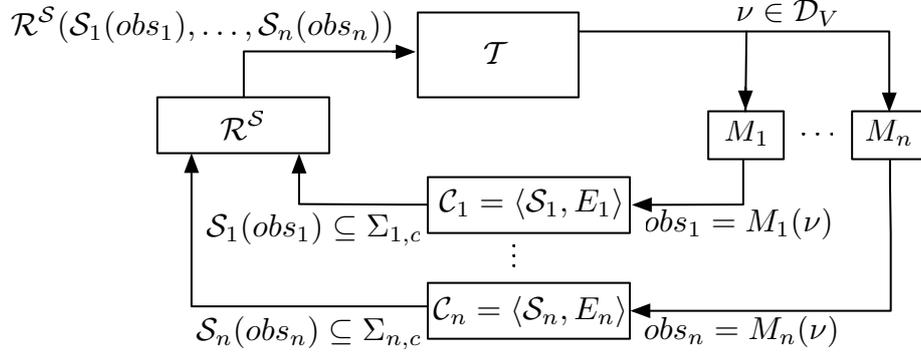


Figure 5.1 - Feedback interaction between the system \mathcal{T} to be controlled and the decentralized controller $\langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$.

The feedback interaction between the system \mathcal{T} to be controlled and the decentralized controller $\langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$ is depicted in Figure 5.1 and can be summarized as follows. When the system arrives in a state $\vec{\nu}$, each local controller $\mathcal{C}_i = \langle \mathcal{S}_i, E_i \rangle$ receives the observation $obs_i = M_i(\vec{\nu})$ and computes the set $\mathcal{S}_i(obs_i)$ of actions that it proposes to forbid to avoid the set *Bad* of forbidden states (the computation of \mathcal{S}_i is defined in section 5.1.2.1). Then, the fusion rule $\mathcal{R} = \langle \mathcal{R}^S, \mathcal{R}^E \rangle$ (the computation of \mathcal{R} is defined in section 5.1.2.1) defines, from the control decisions of each local controller \mathcal{C}_i , the actions that the system cannot execute. It thus defines the *global control* to be applied to the system. The controlled system resulting from this interaction is formally defined as follows.

Definition 5.3 (Controlled System)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, n observers $\langle Obs_i, M_i \rangle$ ($\forall i \in [1, n]$), and a decentralized controller $\mathcal{C}_d = \langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$ (where, for each $i \in [1, n]$, $\mathcal{C}_i = \langle \mathcal{S}_i, E_i \rangle$ and $\mathcal{R} = \langle \mathcal{R}^S, \mathcal{R}^E \rangle$), the system \mathcal{T} controlled by \mathcal{C}_d , denoted by $\mathcal{T}/\mathcal{C}_d$, is an STS defined by $\mathcal{T}/\mathcal{C}_d \stackrel{\text{def}}{=} \langle V, (\Theta_0)_{/\mathcal{C}_d}, \Sigma, \Delta_{/\mathcal{C}_d} \rangle$, where:

- $(\Theta_0)_{/\mathcal{C}_d} = \{ \vec{\nu} \mid \vec{\nu} \in (\Theta_0 \setminus \mathcal{R}^E(E_1, \dots, E_n)) \}$ is the set of initial states.
- $\Delta_{/\mathcal{C}_d}$ is defined from Δ as follows: for each transition $\langle \sigma, G, A \rangle \in \Delta$, we define a new transition $\langle \sigma, G_{/\mathcal{C}_d}, A \rangle \in \Delta_{/\mathcal{C}_d}$ where $G_{/\mathcal{C}_d} \stackrel{\text{def}}{=} \{ \vec{\nu} \mid (\vec{\nu} \in G) \wedge (\sigma \notin \mathcal{R}^S(\mathcal{S}_1(M_1(\vec{\nu})), \dots, \mathcal{S}_n(M_n(\vec{\nu})))) \}$.

The decentralized controller \mathcal{C}_d thus restricts the guards of the system. Indeed, a transition $\langle \sigma, G, A \rangle$ can no longer be fired from a state $\vec{\nu}$ in the controlled

system, if the action σ is forbidden by the fusion rule.

Definition of the State Avoidance Decentralized Control Problem. We define two distinct versions of the state avoidance control problem adapted to the decentralized framework:

Problem 5.1 (Basic Decentralized Problem)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, n observers $\langle Obs_i, M_i \rangle$ ($\forall i \in [1, n]$), and a set Bad of forbidden states, the *basic decentralized problem* consists in computing a decentralized controller $\mathcal{C}_d = \langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$ such that (i) $Reachable_{\Delta/\mathcal{C}_d}^{\mathcal{T}/\mathcal{C}_d}((\Theta_0)_{/\mathcal{C}_d}) \neq \emptyset$ and (ii) $Reachable_{\Delta/\mathcal{C}_d}^{\mathcal{T}/\mathcal{C}_d}((\Theta_0)_{/\mathcal{C}_d}) \cap Bad = \emptyset$.

This problem thus consists in synthesizing a *non-trivial* and *valid* decentralized controller.

We are also interested in a second problem where the deadlock free property is ensured:

Problem 5.2 (Deadlock Free Decentralized Problem)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, n observers $\langle Obs_i, M_i \rangle$ ($\forall i \in [1, n]$), and a set Bad of forbidden states, the *deadlock free decentralized problem* consists in computing a decentralized controller $\mathcal{C}_d = \langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$ such that (i) $Reachable_{\Delta/\mathcal{C}_d}^{\mathcal{T}/\mathcal{C}_d}((\Theta_0)_{/\mathcal{C}_d}) \neq \emptyset$, (ii) $Reachable_{\Delta/\mathcal{C}_d}^{\mathcal{T}/\mathcal{C}_d}((\Theta_0)_{/\mathcal{C}_d}) \cap Bad = \emptyset$, and (iii) $\mathcal{T}_{/\mathcal{C}_d}$ is deadlock free.

The decentralized controller \mathcal{C}_d must thus be (i) non-trivial and (ii) valid, and (iii) it must define a deadlock free controlled system.

We extend the concept of *permissiveness* (see Definition 3.12) to the decentralized framework in order to compare the quality of the different solutions of a given problem:

Definition 5.4 (Permissiveness)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$ and n observers $\langle Obs_i, M_i \rangle$ ($\forall i \in [1, n]$), a decentralized controller $\mathcal{C}_d = \langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$ is more *permissive* than a decentralized controller $\mathcal{C}'_d = \langle (\mathcal{C}'_i)_{i=1}^n, \mathcal{R}' \rangle$ (denoted by $\mathcal{C}'_d \preceq_p^d \mathcal{C}_d$) if and only if $Reachable_{\Delta/\mathcal{C}'_d}^{\mathcal{T}/\mathcal{C}'_d}((\Theta_0)_{/\mathcal{C}'_d}) \subseteq Reachable_{\Delta/\mathcal{C}_d}^{\mathcal{T}/\mathcal{C}_d}((\Theta_0)_{/\mathcal{C}_d})$. When the inclusion is strict, we say that \mathcal{C}_d is *strictly more permissive* than \mathcal{C}'_d (denoted by $\mathcal{C}'_d \prec_p^d \mathcal{C}_d$).

Since the centralized control is a particular case of the decentralized control (it corresponds to the case where n is equal to 1), all results proved in sec-

tion 3.3.4 remain valid for the decentralized framework; in particular, computing a decentralized controller for the basic or deadlock free decentralized problem is *undecidable*. Therefore, we still use abstract interpretation techniques to define effective algorithms for these problems.

5.1.2 Computation by Means of Abstract Interpretation of a Decentralized Controller for the Basic Decentralized Problem

In this section, we define our algorithm which synthesizes decentralized controllers for the basic decentralized problem. First, we present a *semi-algorithm* for this problem and next, we explain how to extend it by using abstract interpretation techniques to obtain an *effective* algorithm.

5.1.2.1 Semi-Algorithm for the Basic Decentralized Problem

Our algorithm, which computes decentralized controllers for the basic decentralized problem, is a generalization of the algorithm defined in section 3.4 for the centralized case. It is composed of two parts:

- we compute, using a fixpoint equation, the set $I(Bad)$ of states leading uncontrollably to Bad .
- we compute the decentralized controller \mathcal{C}_d . Each local controller of \mathcal{C}_d forbids, for each observation, the controllable actions that lead to $I(Bad)$ from a state compatible with this observation.

These two steps are detailed below.

Computation of $I(Bad)$. The set $I(Bad)$ of states leading uncontrollably to Bad is given by the fixpoint equation defined in (3.1) which, as a reminder, is equal to $\text{lfp}^{\subseteq}(\lambda B. Bad \cup \text{Pre}_{\Delta_{uc}}^T(B))$. This fixpoint equation is used for the computations of the n local controllers \mathcal{C}_i and of the fusion rule \mathcal{R} .

Computation of the Decentralized Controller \mathcal{C}_d . First, we detail the computation of the n local controllers \mathcal{C}_i and next, we explain how to compute the fusion rule $\mathcal{R} = \langle \mathcal{R}^S, \mathcal{R}^E \rangle$.

The computation of the local controllers \mathcal{C}_i is quite similar to the computation of the centralized controller defined in section 3.4 for the basic problem. More precisely, we first define, for each controller \mathcal{C}_i ($\forall i \in [1, n]$), the function \mathcal{F}_i :

$\Sigma \times 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_{Obs_i}}$ that gives, for each action $\sigma \in \Sigma$ and for each set $B \subseteq \mathcal{D}_V$ of states to be forbidden, the set $\mathcal{F}_i(\sigma, B)$ of observations, for which the controller \mathcal{C}_i proposes to forbid the action σ . This set $\mathcal{F}_i(\sigma, B)$ actually corresponds to the greatest set $\mathcal{O}_i \subseteq \mathcal{D}_{Obs_i}$ of observations such that, for each observation $obs \in \mathcal{O}_i$, there exists a state $\vec{v} \in \mathcal{D}_V$ (with $obs = M_i(\vec{v})$) which leads to B by a transition labeled by σ :

$$\mathcal{F}_i(\sigma, B) \stackrel{\text{def}}{=} \begin{cases} M_i(\text{Pre}_\sigma^T(B) \setminus B) & \text{if } \sigma \in \Sigma_{i,c} \\ \emptyset & \text{otherwise} \end{cases} \quad (5.1)$$

Next, each local controller \mathcal{C}_i ($\forall i \in [1, n]$) is defined, from its function \mathcal{F}_i , in the following way:

$$\mathcal{C}_i \stackrel{\text{def}}{=} \langle \mathcal{S}_i, E_i \rangle \quad (5.2)$$

where (i) the supervisory function \mathcal{S}_i is defined, for each $obs \in \mathcal{D}_{Obs_i}$, by $\mathcal{S}_i(obs) \stackrel{\text{def}}{=} \{\sigma \in \Sigma_{i,c} \mid obs \in \mathcal{F}_i(\sigma, I(\text{Bad}))\}$, and (ii) the set $E_i \stackrel{\text{def}}{=} I(\text{Bad})$.

Then, the fusion rule \mathcal{R} , which defines the global control to be applied to the system, is defined from the control decisions of each local controller \mathcal{C}_i in the following way:

$$\mathcal{R} \stackrel{\text{def}}{=} \langle \mathcal{R}^S, \mathcal{R}^E \rangle \quad (5.3)$$

where the elements \mathcal{R}^S and \mathcal{R}^E are defined as follows:

- Let $B_i \subseteq \Sigma_{i,c}$ ($\forall i \in [1, n]$) be the set of actions that the local controller \mathcal{C}_i proposes to forbid, the function \mathcal{R}^S , which gives the set of actions that the system cannot execute, is defined in the following way:

$$\mathcal{R}^S(B_1, \dots, B_n) \stackrel{\text{def}}{=} \{\sigma \in \Sigma_c \mid \forall i \in \text{In}(\sigma) : \sigma \in B_i\} \quad (5.4)$$

Thus, an action σ cannot be executed by the system, if each local controller, which controls this action, proposes to forbid it. Usually, one uses a conjunctive or disjunctive architecture as defined in [RW92b, YL02] i.e., an action is globally enabled when all local controllers enable it (conjunctive architecture) or at least one local controller enables it (disjunctive architecture). In our framework, an action is enabled whenever at least one local controller, that controls this action, enables it. So, we use a variant of the disjunctive architecture. We will prove that our fusion rule defines correct solutions for the basic and deadlock free decentralized problems (see Propositions 5.1 and 5.3), whereas a disjunctive architecture does not always give correct solutions for

these problems (see Example 5.1). In [TKU94], which is one of the few papers on the decentralized control of discrete event systems with a state-based approach, the authors use a conjunctive architecture to synthesize a decentralized controller that allows the system to exactly reach a set Q of good states. However, they want to obtain a *balanced* decentralized controller (i.e., for each pair of states \vec{v}, \vec{v}' reachable in the controlled system, if there is an action σ which can be fired from \vec{v} to \vec{v}' , then this action must be enabled by each local controller). We do not ensure that our decentralized controllers are balanced, since \preceq_p -maximal decentralized controllers do not always satisfy this property (see Example 5.1).

- Let $E_i \subseteq \mathcal{D}_V$ ($\forall i \in [1, n]$) be the set of states that the local controller \mathcal{C}_i proposes to forbid at the beginning of the execution of the system, the function $\mathcal{R}^{\mathcal{E}}$, which gives the set of states that must effectively be forbidden, is defined as follows:

$$\mathcal{R}^{\mathcal{E}}(E_1, \dots, E_n) \stackrel{\text{def}}{=} \bigcap_{i=1}^n E_i \quad (5.5)$$

A state $\vec{v} \in \mathcal{D}_V$ is thus forbidden at the beginning of the execution of the system, if each local controller \mathcal{C}_i proposes to forbid it.

Finally, the decentralized controller \mathcal{C}_d is defined by the n local controllers \mathcal{C}_i given in (5.2) and by the fusion rule \mathcal{R} given in (5.3):

$$\mathcal{C}_d \stackrel{\text{def}}{=} \langle \langle \mathcal{C}_i \rangle_{i=1}^n, \mathcal{R} \rangle \quad (5.6)$$

The set $I(\text{Bad})$ and the functions \mathcal{F}_i ($\forall i \in [1, n]$) are computed *offline*. Next, during the execution of the system, when the system arrives in a state $\vec{v} \in \mathcal{D}_V$, each local controller \mathcal{C}_i ($\forall i \in [1, n]$) receives the observation $obs_i = M_i(\vec{v}) \in \mathcal{D}_{Obs_i}$ and computes *online* the set $\mathcal{S}_i(obs_i)$ of actions that it proposes to forbid. The fusion rule $\mathcal{R}^{\mathcal{S}}(\mathcal{S}_1(obs_{s_1}), \dots, \mathcal{S}_n(obs_{s_n}))$ is then computed *online* from these control decisions to obtain the set of actions that the system cannot execute.

Example 5.1

We compute a decentralized controller $\mathcal{C}_d = \langle \langle \mathcal{C}_1, \mathcal{C}_2 \rangle, \mathcal{R} \rangle$ for the system depicted in Figure 3.1.

The controller \mathcal{C}_1 has a partial observation modeled by the mask $M_1 : Loc \times \mathbb{Z}^4 \rightarrow 2^{Loc \times \mathbb{Z}^4}$ where, for each state $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, the set of states, that are undistinguishable from \vec{v} , is given by $\{ \langle \ell, x, x'_1, y, y'_1 \rangle \mid x'_1, y'_1 \in \mathbb{Z} \}$ (i.e., \mathcal{C}_1 does not observe the variables x' and y'). \mathcal{C}_1 does not control the actions $Cons, Cons', Stop_prod'$ (i.e., $\Sigma_{1,uc} = \{Cons, Cons', Stop_prod'\}$).

The controller \mathcal{C}_2 has a partial observation modeled by the mask $M_2 : Loc \times \mathbb{Z}^4 \rightarrow 2^{Loc \times \mathbb{Z}^4}$ where, for each state $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, the set of states, that are undistinguishable from \vec{v} , is given by $\{\langle \ell, x_1, x', y_1, y' \rangle \mid x_1, y_1 \in \mathbb{Z}\}$ (i.e., \mathcal{C}_2 does not observe the variables x and y). \mathcal{C}_2 does not control the actions $Cons, Cons'$ (i.e., $\Sigma_{2,uc} = \{Cons, Cons'\}$). The decentralized controller must ensure, during the phase of consumption of the parts X and X' , the presence of a minimal number of parts: $Bad = \{\langle CX, x, x', y, y' \rangle \mid (x \leq 10) \wedge (y \in [0, 2])\} \cup \{\langle CX', x, x', y, y' \rangle \mid (x' \leq 10) \wedge (y' \in [0, 2])\}$. The computation of the set $I(Bad)$ gives:

$$\begin{aligned} I(Bad) = & Bad \cup \{\langle CX, x, x', y, y' \rangle \mid [(x \leq 11) \wedge (y \in [1, 2])] \vee [(x \leq 12) \\ & \wedge (y = 2)]\} \cup \{\langle CX', x, x', y, y' \rangle \mid [(x' \leq 11) \wedge (y' \in [1, 2])] \vee \\ & [(x' \leq 12) \wedge (y' = 2)]\} \end{aligned}$$

The computation of the function \mathcal{F}_1 gives:

$$\mathcal{F}_1(\sigma, I(Bad)) = \begin{cases} M_1(\{\langle PX, x, x', y, y' \rangle \mid x \leq 12\}) & \text{if } \sigma = Stop_prod \\ \emptyset & \text{otherwise} \end{cases}$$

The supervisory function \mathcal{S}_1 thus forbids the action $Stop_prod$ when the system is in a state $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid x \leq 12\}$ and the computation of the function \mathcal{F}_2 gives:

$$\mathcal{F}_2(\sigma, I(Bad)) = \begin{cases} M_2(\{\langle PX, x, x', y, y' \rangle \mid x \leq 12\}) & \text{if } \sigma = Stop_prod \\ M_2(\{\langle PX', x, x', y, y' \rangle \mid x' \leq 12\}) & \text{if } \sigma = Stop_prod' \\ \emptyset & \text{otherwise} \end{cases}$$

Thus, the supervisory function \mathcal{S}_2 forbids the action $Stop_prod'$ when the system is in a state $\vec{v} \in \{\langle PX', x, x', y, y' \rangle \mid x' \leq 12\}$ and it always forbids the action $Stop_prod$ in the location PX . According to the fusion rule \mathcal{R} , the decentralized controller \mathcal{C}_d forbids the action $Stop_prod$ (resp. $Stop_prod'$) in the location PX (resp. PX') when $x \leq 12$ (resp. $x' \leq 12$). Note that \mathcal{C}_d is \preceq_p -maximal.

One can note that with a disjunctive architecture, the decentralized controller always allows the action $Stop_prod'$ in the location PX' , because the local controller \mathcal{C}_1 always allows this action in PX' ($Stop_prod' \in \Sigma_{1,uc}$). Therefore, Bad is reachable with a disjunctive architecture. Moreover, the decentralized controller \mathcal{C}_d , which is \preceq_p -maximal, is not balanced. Indeed, in the controlled system $\mathcal{T}_{/\mathcal{C}_d}$, the state $\vec{v} = \langle CX, 50, 50, 2, 1 \rangle$ is reachable from the state $\vec{v}' = \langle PX, 50, 50, 0, 1 \rangle$ by the action $Stop_prod$, but this action is forbidden by \mathcal{C}_2 in

the state \vec{v}' , whereas it is allowed by \mathcal{C}_1 in this state.

The following property proves that our algorithm synthesizes correct controllers for the basic decentralized problem:

Proposition 5.1

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, n observers $\langle Obs_i, M_i \rangle$ ($\forall i \in [1, n]$) and a predicate Bad , which represents a set of forbidden states, the decentralized controller $\mathcal{C}_d = \langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$ (where, for each $i \in [1, n]$, $\mathcal{C}_i = \langle \mathcal{S}_i, E_i \rangle$ and $\mathcal{R} = \langle \mathcal{R}^S, \mathcal{R}^E \rangle$), defined in (5.6), solves the basic decentralized problem (when $\Theta_0 \not\subseteq \mathcal{R}^E(E_1, \dots, E_n)$).

Proof

We prove by induction on the length ℓ of the sequences of transitions (these sequences begin in an initial state) that $\text{Reachable}_{\Delta/\mathcal{C}_d}^{\mathcal{T}/\mathcal{C}_d}((\Theta_0)_{/\mathcal{C}_d}) \cap I(Bad) = \emptyset$. Since $Bad \subseteq I(Bad)$, this property implies that $\text{Reachable}_{\Delta/\mathcal{C}_d}^{\mathcal{T}/\mathcal{C}_d}((\Theta_0)_{/\mathcal{C}_d}) \cap Bad = \emptyset$:

- *Base case* ($\ell = 0$): the execution of the controlled system $\mathcal{T}/\mathcal{C}_d$ starts in a state which does not belong to $I(Bad)$, since $(\Theta_0)_{/\mathcal{C}_d} = \Theta_0 \setminus \mathcal{R}^E(E_1, \dots, E_n) = \Theta_0 \setminus I(Bad)$.
- *Induction step*: we suppose that the proposition holds for the sequences of transitions of length less than or equal to ℓ and we prove that this property remains true for the sequences of transitions of length $\ell + 1$. By induction hypothesis, each state \vec{v}_1 reachable by a sequence of transitions of length ℓ does not belong to $I(Bad)$ and we show that each transition $\delta = \langle \sigma, G, A \rangle \in \Delta$, which can lead to a state $\vec{v}_2 \in I(Bad)$ from this state \vec{v}_1 in the system \mathcal{T} , cannot be fired from \vec{v}_1 in the controlled system $\mathcal{T}/\mathcal{C}_d$. For that, we consider two cases :
 - if $\delta \in \Delta_c$, then this transition cannot be fired from \vec{v}_1 in the system $\mathcal{T}/\mathcal{C}_d$. Indeed, for each $i \in \text{In}(\sigma)$, $\sigma \in \mathcal{S}_i(M_i(\vec{v}))$ by (5.1) and (5.2), and hence $\sigma \in \mathcal{R}^S(\mathcal{S}_1(M_1(\vec{v})), \dots, \mathcal{S}_n(M_n(\vec{v})))$ by (5.4).
 - if $\delta \in \Delta_{uc}$, then $\vec{v}_2 \in I(Bad)$ (by (3.1)), which is impossible by hypothesis.

Hence, in the controlled system, the forbidden state \vec{v}_2 cannot be reached from the state \vec{v}_1 by the transition δ .

The controlled system $\mathcal{T}_{/c_d} = \langle V, (\Theta_0)_{/c_d}, \Sigma, \Delta_{/c_d} \rangle$ (see Definition 5.3) can be computed for our decentralized controller \mathcal{C}_d , since:

- $(\Theta_0)_{/c_d}$ is obtained by computing the set operation $\Theta_0 \setminus I(Bad)$, and
- for any transition $\langle \sigma, G_{/c_d}, A \rangle \in \Delta_{/c_d}$ (let us suppose that this transition is defined from the transition $\langle \sigma, G, A \rangle \in \Delta$), the restricted guard $G_{/c_d}$ is obtained by computing the set operation $G \setminus \{ \bigcap_{i \in \text{In}(\sigma)} M_i^{-1}(\mathcal{F}_i(\sigma, I(Bad))) \}$. Indeed, the function \mathcal{R}^S forbids an action σ in a state \vec{v} if each local controller, which controls this action, proposes to forbid it i.e., $\forall i \in \text{In}(\sigma) : \vec{v} \in M_i^{-1}(\mathcal{F}_i(\sigma, I(Bad)))$.

5.1.2.2 Effective Algorithm for the Basic Decentralized Problem

The computation of the previous algorithm (in particular, the computation of the fixpoint equation $I(Bad)$) does not always terminate and we use abstract interpretation techniques, as in section 3.4.2, to overcome this obstacle.

More precisely, our effective algorithm for the basic decentralized problem works as follows. First, we compute an overapproximation $I'(Bad)$ of $I(Bad)$ by using abstract interpretation (see section 3.4.2 for the details). Next, we transpose the function \mathcal{F}_i ($\forall i \in [1, n]$), defined in (5.1), into the abstract lattice and we obtain the abstract function \mathcal{F}_i^\sharp (this computation may thus require overapproximations). Finally, we define our decentralized controller $\mathcal{C}_d = \langle (\langle \mathcal{S}_i, I'(Bad) \rangle)_{i=1}^n, \mathcal{R} \rangle$ as in (5.6) by using $I'(Bad)$ and \mathcal{F}_i^\sharp instead of $I(Bad)$ and \mathcal{F}_i .

5.1.3 Computation by Means of Abstract Interpretation of a Decentralized Controller for the Deadlock Free Decentralized Problem

In this section, we define an algorithm which computes decentralized controllers for the deadlock free decentralized problem. First, we present a *semi-algorithm* for this problem and next, we explain how to extend it by means of abstract interpretation to ensure the termination of the computations.

5.1.3.1 Semi-Algorithm for the Deadlock Free Decentralized Problem

Our algorithm, which computes decentralized controllers for the deadlock free decentralized problem, is obtained by generalizing the algorithm defined in sec-

tion 3.5 for the centralized case and is composed of two parts:

- we compute, using a fixpoint equation, the set $I_{df}^d(Bad)$ of states leading uncontrollably to Bad or to deadlocking states.
- we define a decentralized controller \mathcal{C}_d which prevents the system from reaching $I_{df}^d(Bad)$ (this part is similar to the one defined for the basic decentralized problem).

Before formalizing these two steps, we explain how to compute the deadlocking states.

Computation of the Deadlocking States. Computing $I_{df}^d(Bad)$ requires to compute the states that will be in deadlock after control. To obtain these states, we define the function $\text{Pre}_{df}^{d,\mathcal{T}} : 2^{\mathcal{D}_V} \rightarrow 2^{\mathcal{D}_V}$ (this function is given (5.8)) which gives, for each set $B \subseteq \mathcal{D}_V$ of states to be forbidden, the set $\text{Pre}_{df}^{d,\mathcal{T}}(B)$ of states that are in deadlock in the controlled system which avoids B . Now, we explain how to compute this function. Let $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$ be a system under the control of \mathcal{C}_d , which avoids the set $B \subseteq \mathcal{D}_V$, then a state $\vec{v} \in \mathcal{D}_V$ of the system \mathcal{T} will be in deadlock in the controlled system $\mathcal{T}_{/\mathcal{C}_d}$ if and only if the two following conditions are satisfied in the system \mathcal{T} :

- the state \vec{v} has no outgoing uncontrollable transition.
- for each controllable transition $\delta = \langle \sigma, G, A \rangle \in \Delta_c$, (i) this transition δ cannot be fired from \vec{v} (i.e., $\vec{v} \notin G$) or (ii) the action σ is forbidden by the decentralized controller in the state \vec{v} (i.e., $\sigma \in \mathcal{R}^S(\mathcal{S}_1(M_1(\vec{v})), \dots, \mathcal{S}_n(M_n(\vec{v})))$). The condition $\sigma \in \mathcal{R}^S(\mathcal{S}_1(M_1(\vec{v})), \dots, \mathcal{S}_n(M_n(\vec{v})))$ is equivalent to $\forall i \in \text{In}(\sigma) : M_i(\vec{v}) \in \mathcal{F}_i(\sigma, B)$, because the function \mathcal{R}^S forbids an action σ in the state \vec{v} if each local controller, which controls this action, proposes to forbid it.

The computation of the deadlocking states is thus based on the function \mathcal{F}_i ($\forall i \in [1, n]$) defined in (5.1). The fixpoint equation $I_{df}^d(Bad)$ (this function is defined in (5.9)) uses the function $\text{Pre}_{df}^{d,\mathcal{T}}$ and to ensure the continuousness of this fixpoint equation, we must use the continuous function $\widehat{\mathcal{F}}_i$ instead of \mathcal{F}_i in the computation of the deadlocking states:

$$\widehat{\mathcal{F}}_i(\sigma, B) \stackrel{\text{def}}{=} \begin{cases} M_i(\text{Pre}_\sigma^{\mathcal{T}}(B)) & \text{if } \sigma \in \Sigma_{i,c} \\ \emptyset & \text{otherwise} \end{cases} \quad (5.7)$$

Formally, the deadlocking states of the controlled system $\mathcal{T}_{/\mathcal{C}_d}$ can then be characterized as follows:

Definition 5.5 (Deadlocking States of the Controlled System)

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, n observers $\langle Obs_i, M_i \rangle$ ($\forall i \in [1, n]$), a set $B \subseteq \mathcal{D}_V$ of states to be forbidden, and a decentralized controller $\mathcal{C}_d = \langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$ (where, for each $i \in [1, n]$, $\mathcal{C}_i = \langle \mathcal{S}_i, E_i \rangle$, and $\mathcal{R} = \langle \mathcal{R}^S, \mathcal{R}^E \rangle$), then a state $\vec{v} \in \mathcal{D}_V$ of the system \mathcal{T} will be in deadlock in the controlled system $\mathcal{T}/\mathcal{C}_d$ if and only if the two following conditions hold in the system \mathcal{T} :

- $\forall \langle \sigma, G, A \rangle \in \Delta_{uc} : \vec{v} \notin G$
- $\forall \langle \sigma, G, A \rangle \in \Delta_c : (\vec{v} \notin G) \vee (\forall i \in \text{In}(\sigma) : M_i(\vec{v}) \in \widehat{\mathcal{F}}_i(\sigma, B))$

Since, for each $\sigma \in \Sigma_{uc}$, the set $\widehat{\mathcal{F}}(\sigma, B) = \emptyset$, the function $\text{Pre}_{df}^{d, \mathcal{T}}(B)$ (with $B \subseteq \mathcal{D}_V$), which gives the set of states, that are in deadlock in the controlled system avoiding B , can be defined as follows¹:

$$\text{Pre}_{df}^{d, \mathcal{T}}(B) \stackrel{\text{def}}{=} B \cup \left[\bigcap_{\langle \sigma, G, A \rangle \in \Delta} \left(G^c \cup \bigcap_{i \in \text{In}(\sigma)} (M_i^{-1}(\widehat{\mathcal{F}}_i(\sigma, B))) \right) \right] \quad (5.8)$$

Now that we can compute the deadlocking states, we formalize the two steps of our algorithm, which synthesizes controllers for the deadlock free decentralized problem:

Computation of $I_{df}^d(\mathbf{Bad})$. The set $I_{df}^d(\mathbf{Bad})$ corresponds to the set of states leading uncontrollably to \mathbf{Bad} or to deadlocking states. To compute $I_{df}^d(\mathbf{Bad})$, we first compute the set $I(\mathbf{Bad})$ (defined in (3.1)). Then, if we make unreachable these forbidden states by cutting all controllable transitions that lead to a bad state, the corresponding controlled system could have new deadlocking states. These states are given by the function $\text{Pre}_{df}^{d, \mathcal{T}}(I(\mathbf{Bad}))$ and we add them to the set of forbidden states. Adding these deadlocking states to the set of forbidden states can provide new states leading uncontrollably to a forbidden state. Consequently, the set $I_{df}^d(\mathbf{Bad})$ is defined by $\bigcup_{n \geq 0} (\text{Pre}_{df}^{d, \mathcal{T}} \circ I)^n(\mathbf{Bad})$. Since the coreachability is *undecidable* in the model of STS, this set of states cannot always be computed, and, in section 5.1.3.2, we will use abstract interpretation techniques to overapproximate it. However, to compute this overapproximation, we must characterize the set $I_{df}^d(\mathbf{Bad})$ by a fixpoint equation and this equation is defined as follows over the complete lattice $\langle 2^{\mathcal{D}_V}, \subseteq, \cup, \cap, \mathcal{D}_V, \emptyset \rangle$:

$$I_{df}^d(\mathbf{Bad}) \stackrel{\text{def}}{=} \text{lfp}^{\subseteq}(\lambda B. \mathbf{Bad} \cup \text{Pre}_{df}^{d, \mathcal{T}}(I(B))) \quad (5.9)$$

¹We assume that $M_i^{-1}(\emptyset) = \emptyset$ ($\forall i \in [1, n]$).

Since the function $\lambda B. Bad \cup \text{Pre}_{df}^{d,T}(I(B))$ is monotonic, the Knaster-Tarski's theorem [Tar55] ensures that the least fixpoint of this function exists and the following proposition proves that this fixpoint equation gives the set of states leading uncontrollably to Bad or to deadlocking states:

Proposition 5.2

The function $\lambda B. Bad \cup \text{Pre}_{df}^{d,T}(I(B))$ defined over the complete lattice $\langle 2^{\mathcal{D}_V}, \subseteq, \cup, \cap, \mathcal{D}_V, \emptyset \rangle$ is such that $\text{lfp}^{\subseteq}(\lambda B. Bad \cup \text{Pre}_{df}^{d,T}(I(B))) = \bigcup_{n \geq 0} (\text{Pre}_{df}^{d,T} \circ I)^n(Bad)$.

Proof

By the Knaster-Tarski and Kleene's theorems, it suffices to show that the functions I and $\text{Pre}_{df}^{d,T}$ are continuous to prove this proposition. In Proposition 3.9, we have already proven that the function I is continuous. For the second function, we must prove that, for each increasing sequence of sets $B_1 \subseteq B_2 \subseteq \dots \subseteq B_n \subseteq \dots$ (where $\forall i \geq 1 : B_i \in 2^{\mathcal{D}_V}$), the equality $\text{Pre}_{df}^{d,T}(\bigcup_{i \geq 1} B_i) = \bigcup_{i \geq 1} \text{Pre}_{df}^{d,T}(B_i)$ holds. This equality is proven by the following reasoning, where the transition $\delta \in \Delta$ is equal to the triple $\langle \sigma, G, A \rangle$:

$$\begin{aligned} & \text{Pre}_{df}^{d,T} \left(\bigcup_{i \geq 1} B_i \right) \\ = & \left(\bigcup_{i \geq 1} B_i \right) \cup \left[\bigcap_{\delta \in \Delta} \left(G^c \cup \left(\bigcap_{i \in \text{In}(\sigma)} [M_i^{-1}(\widehat{\mathcal{F}}_i(\sigma, \bigcup_{i \geq 1} B_i))] \right) \right) \right], \text{ by} \\ & \text{definition of } \text{Pre}_{df}^{d,T}. \\ = & \left(\bigcup_{i \geq 1} B_i \right) \cup \left[\bigcap_{\delta \in \Delta} \left(G^c \cup \left(\bigcap_{i \in \text{In}(\sigma)} [M_i^{-1}(M_i(\text{Pre}_{\sigma}^T \left(\bigcup_{i \geq 1} B_i \right))] \right) \right) \right], \\ & \text{by definition of } \widehat{\mathcal{F}}_i. \\ = & \left(\bigcup_{i \geq 1} B_i \right) \cup \left[\bigcap_{\delta \in \Delta} \left[G^c \cup \left[\bigcap_{i \in \text{In}(\sigma)} \left(\bigcup_{i \geq 1} [M_i^{-1}(M_i(\text{Pre}_{\sigma}^T(B_i)))] \right) \right] \right] \right], \\ & \text{because } \text{Pre}_{\sigma}^T = \text{Pre}_{\text{Trans}(\sigma)}^T \text{ is continuous (see the proof of} \\ & \text{Proposition 3.9) and because } M_i \text{ and } M_i^{-1} \text{ are} \\ & \text{continuous.} \end{aligned}$$

$$\begin{aligned}
&= \left(\bigcup_{i \geq 1} B_i \right) \cup \left[\bigcap_{\delta \in \Delta} \left[G^c \cup \left[\bigcup_{i \geq 1} \left(\bigcap_{i \in \text{In}(\sigma)} [M_i^{-1}(M_i(\text{Pre}_\sigma^T(B_i)))] \right) \right] \right] \right], \\
&\quad \text{because } \text{In}(\sigma) \text{ is finite and } a \cap \left(\bigcup_i b_i \right) = \bigcup_i (a \cap b_i). \\
&= \left(\bigcup_{i \geq 1} B_i \right) \cup \left[\bigcap_{\delta \in \Delta} \left[\bigcup_{i \geq 1} \left(G^c \cup \left(\bigcap_{i \in \text{In}(\sigma)} [M_i^{-1}(M_i(\text{Pre}_\sigma^T(B_i)))] \right) \right) \right] \right], \\
&\quad \text{because } a \cup \left(\bigcup_i b_i \right) = \bigcup_i (a \cup b_i). \\
&= \left(\bigcup_{i \geq 1} B_i \right) \cup \left[\bigcup_{i \geq 1} \left[\bigcap_{\delta \in \Delta} \left(G^c \cup \left(\bigcap_{i \in \text{In}(\sigma)} [M_i^{-1}(M_i(\text{Pre}_\sigma^T(B_i)))] \right) \right) \right] \right], \\
&\quad \text{because } \Delta \text{ is finite and } a \cap \left(\bigcup_i b_i \right) = \bigcup_i (a \cap b_i). \\
&= \bigcup_{i \geq 1} \left(B_i \cup \left[\bigcap_{\delta \in \Delta} \left(G^c \cup \left(\bigcap_{i \in \text{In}(\sigma)} [M_i^{-1}(M_i(\text{Pre}_\sigma^T(B_i)))] \right) \right) \right] \right) \\
&= \bigcup_{i \geq 1} \text{Pre}_{df}^{d,T}(B_i)
\end{aligned}$$

Computation of the decentralized controller \mathcal{C}_d . The decentralized controller \mathcal{C}_d is defined as in(5.6) by using the set $I_{df}^d(Bad)$ instead of $I(Bad)$.

Example 5.2

We compute a decentralized controller $\mathcal{C}_d = \langle \langle \mathcal{C}_1, \mathcal{C}_2 \rangle, \mathcal{R} \rangle$ for the system depicted in Figure 3.1.

The controller \mathcal{C}_1 has a partial observation modeled by the mask $M_1 : Loc \times \mathbb{Z}^4 \rightarrow 2^{Loc \times \mathbb{Z}^4}$ where, for each state $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, the set of states that are undistinguishable from \vec{v} , is given by $\{ \langle \ell, x_1, x', y_1, y' \rangle \mid x_1, y_1 \in \mathbb{Z} \}$ (i.e., \mathcal{C}_1 does not observe the variables x and y). \mathcal{C}_1 does not control the actions $Cons, Cons'$ (i.e., $\Sigma_{1,uc} = \{Cons, Cons'\}$).

The controller \mathcal{C}_2 has a partial observation modeled by the mask $M_2 : Loc \times \mathbb{Z}^4 \rightarrow 2^{Loc \times \mathbb{Z}^4}$ where, for each state $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, the set of states,

that are undistinguishable from \vec{v} , is given by $\{\langle \ell, x_1, x', y, y' \rangle \mid x_1 \in [x-1, x+1]\}$ (i.e., there is an imprecision regarding the number of parts X). \mathcal{C}_2 does not control the actions $Cons, Cons'$ (i.e., $\Sigma_{2,uc} = \{Cons, Cons'\}$). The decentralized controller must prevent the system from reaching the set $Bad = \{\langle CX, x, x', y, y' \rangle \mid (x \leq 50) \wedge (y \in [0, 2])\} \cup \{\langle PX, x, x', y, y' \rangle \mid (25 \leq x \leq 75)\}$. The computation of the set $I(Bad)$ gives:

$$I(Bad) = Bad \cup \{\langle CX, x, x', y, y' \rangle \mid [(x \leq 51) \wedge (y \in [1, 2])] \vee [(x \leq 52) \wedge (y = 2)]\}$$

The computation of the function $\widehat{\mathcal{F}}_i$ (for $i = 1, 2$) gives:

$$\widehat{\mathcal{F}}_i(\sigma, I(Bad)) = \begin{cases} M_i(\{\langle PX, x, x', y, y' \rangle \mid x \leq 52\}) & \text{if } \sigma = Stop_prod \\ M_i(\{\langle PX, x, x', y, y' \rangle \mid 24 \leq x \leq 74\}) & \text{if } \sigma = Prod \\ M_i(\{\langle Choice, x, x', y, y' \rangle \mid 25 \leq x \leq 75\}) & \text{if } \sigma = Choice_X \\ \emptyset & \text{otherwise} \end{cases}$$

Therefore, to prevent the system from reaching $I(Bad)$, the controller \mathcal{C}_1 proposes to forbid (i) the actions $Stop_prod$ and $Prod$ in the location PX and (ii) the action $Choice_X$ in the location $Choice$, and the controller \mathcal{C}_2 proposes to forbid (i) the action $Stop_prod$ when the system is in a state $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid x \leq 53\}$, (ii) the action $Prod$ when the system is in a state $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid 23 \leq x \leq 75\}$ and (iii) the action $Choice_X$ when the system is in a state $\vec{v} \in \{\langle Choice, x, x', y, y' \rangle \mid 24 \leq x \leq 76\}$. Since, the states $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid 23 \leq x \leq 53\}$ are in deadlock, they are added to the forbidden states i.e., $\text{Pre}_{df}^{d,T}(I(Bad)) = I(Bad) \cup \{\langle PX, x, x', y, y' \rangle \mid 23 \leq x \leq 53\}$. The set of states $I(\text{Pre}_{df}^{d,T}(I(Bad)))$ is equal to $\text{Pre}_{df}^{d,T}(I(Bad))$.

Consequently, to prevent the system from reaching these states, the controller \mathcal{C}_1 proposes to forbid (i) the actions $Stop_prod$ and $Prod$ in the location PX and (ii) the action $Choice_X$ in the location $Choice$, and the controller \mathcal{C}_2 proposes to forbid (i) the action $Stop_prod$ when the system is in a state $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid x \leq 53\}$, (ii) the action $Prod$ when the system is in a state $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid 21 \leq x \leq 75\}$ and (iii) the action $Choice_X$ when the system is in a state $\vec{v} \in \{\langle Choice, x, x', y, y' \rangle \mid 22 \leq x \leq 76\}$. Since the states $\vec{v} \in \{\langle PX, x, x', y, y' \rangle \mid 21 \leq x \leq 53\}$ are in deadlock, they are added to the forbidden states. By continuing the computations in this way, we obtain (i) a set $I_{df}^d(Bad) = I(Bad) \cup \{\langle PX, x, x', y, y' \rangle \mid x \leq 53\}$ and (ii) control functions \mathcal{F}_i (for $i = 1, 2$) defined by:

$$\mathcal{F}_i(\sigma, I_{df}^d(Bad)) = \begin{cases} M_i(\{\langle \text{Choice}, x, x', y, y' \rangle \mid x \leq 75\}) & \text{if } \sigma = \text{Choice_X} \\ \emptyset & \text{otherwise} \end{cases}$$

Thus, the controller \mathcal{C}_1 proposes to forbid the action *Choice_X* in the location *Choice*, and the controller \mathcal{C}_2 proposes to forbid the action *Choice_X* in the location *Choice* when $x \leq 76$. Therefore, the decentralized controller forbids the action *Choice_X* in the location *Choice* when $x \leq 76$.

The following property proves that our algorithm synthesizes correct controllers for the deadlock free decentralized problem:

Proposition 5.3

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, n observers $\langle \text{Obs}_i, M_i \rangle$ ($\forall i \in [1, n]$) and a predicate *Bad*, which represents a set of forbidden states, the decentralized controller $\mathcal{C}_d = \langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$ (where, for each $i \in [1, n]$, $\mathcal{C}_i = \langle \mathcal{S}_i, E_i \rangle$ and $\mathcal{R} = \langle \mathcal{R}^S, \mathcal{R}^E \rangle$), defined in section 5.1.3.1, solves the deadlock free decentralized problem (when $\Theta_0 \not\subseteq \mathcal{R}^E(E_1, \dots, E_n)$).

Proof

Since $I(Bad) \subseteq I_{df}^d(Bad)$, it can be proved, as in the proof of Proposition 5.1, that *Bad* is not reachable in this more restrictive controlled system.

Now, let us suppose that the controlled system $\mathcal{T}/\mathcal{C}_d$ does not satisfy the deadlock free property. Then, there exists at least one deadlocking state $\vec{v} \in \mathcal{D}_V$, which is reachable in this controlled system. By definition of the fixpoint $I_{df}^d(Bad)$ (see (5.9)), this state $\vec{v} \in I_{df}^d(Bad)$, and so is any state $\vec{v}' \in \mathcal{D}_V$ which can lead to \vec{v} by a sequence of uncontrollable transitions. But according to the definition of \mathcal{C}_d , the states \vec{v} and \vec{v}' are both unreachable in the controlled system $\mathcal{T}/\mathcal{C}_d$, which implies a contradiction.

5.1.3.2 Effective Algorithm for the Deadlock Free Decentralized Problem

The computation of the previous algorithm and, in particular, the computation of the fixpoint $I_{df}^d(Bad)$ does not always terminate. To overcome this obstacle, we define an effective algorithm which works like the semi-algorithm defined in the previous section, except that we perform the computations in the abstract lattice and that we compute an overapproximation of $I_{df}^d(Bad)$ to ensure the termination of our algorithm. This overapproximation is obtained by using the instantiation

of the representation framework defined in section 3.4.2 and by transposing the fixpoint equation $I_{df}^d(Bad)$ into the abstract lattice in the following way:

$$I_{df}^{d,\sharp}(Bad) \stackrel{\text{def}}{=} \text{lfp}^{\sqsubseteq}(\lambda \ell. \alpha(Bad) \sqcup (\text{Pre}_{df}^{d,\mathcal{T}})^{\sharp}(I^{\sharp}(\ell))) \quad (5.10)$$

where (i) the fixpoint equation $I^{\sharp}(\ell)$ is defined in (3.7) and (ii) the function $(\text{Pre}_{df}^{d,\mathcal{T}})^{\sharp} : \Lambda \rightarrow \Lambda$ is defined, for each $\ell \in \Lambda$, by:

$$(\text{Pre}_{df}^{d,\mathcal{T}})^{\sharp}(\ell) \stackrel{\text{def}}{=} \ell \sqcup \left[\bigsqcap_{\langle \sigma, G, A \rangle \in \Delta} \left[\alpha(G^c) \sqcup \bigsqcap_{i \in \text{In}(\sigma)} \left(\alpha(M_i^{-1}(M_i(\gamma(\text{Pre}_{\text{Trans}(\sigma)}^{\mathcal{T},\sharp}(\ell)))) \right) \right) \right] \right]$$

where $\text{Pre}_{\text{Trans}(\sigma)}^{\mathcal{T},\sharp}(\ell)$ is defined in (3.5).

Next, we use the fixpoint computation strategy defined in section 3.4.2 to compute a post-fixpoint a_{∞} of $I_{df}^{d,\sharp}(Bad)$ and the element $\gamma(a_{\infty})$ (denoted by $(I_{df}^d)'(Bad)$) is an overapproximation of $I_{df}^d(Bad)$.

Then, we transpose the function \mathcal{F}_i ($\forall i \in [1, n]$), defined in (5.1), into the abstract lattice and we obtain the function \mathcal{F}_i^{\sharp} (the computation of this computation may thus require overapproximations). Finally, we define the decentralized controller $\mathcal{C}_d = \langle \langle \mathcal{S}_i, (I_{df}^d)'(Bad) \rangle_{i=1}^n, \mathcal{R} \rangle$ as in (5.6) by using the $(I_{df}^d)'(Bad)$ and \mathcal{F}_i^{\sharp} .

5.1.4 Comparison Between the Centralized and Decentralized Controllers

In this section, we compare the permissiveness of the decentralized controller $\mathcal{C}_d = \langle \langle \mathcal{C}_i \rangle_{i=1}^n, \mathcal{R} \rangle$ defined in (5.6) and of the centralized controller \mathcal{C} defined in (3.3). Let $\langle \text{Obs}_i^d, M_i^d \rangle$ be the observer of the local controller \mathcal{C}_i ($\forall i \in [1, n]$) and $\Sigma_{i,c}^d$ (resp. $\Sigma_{i,uc}^d$) be the set of actions that \mathcal{C}_i controls (resp. does not control). Then, the set of actions that \mathcal{C} controls is defined by $\Sigma_c^c \stackrel{\text{def}}{=} \bigcup_{i=1}^n \Sigma_{i,c}^d$ (i.e., \mathcal{C} controls an action σ , if this action is controlled by at least one of the controllers \mathcal{C}_i) and the set of actions that \mathcal{C} does not control is defined by $\Sigma_{uc}^c \stackrel{\text{def}}{=} \Sigma \setminus \Sigma_c^c$. Moreover, the partial observation of \mathcal{C} is modeled by the observer $\langle \text{Obs}^c, M^c \rangle$, where (i) the variable $\text{Obs}^c \stackrel{\text{def}}{=} \text{Obs}_1^d \cup \dots \cup \text{Obs}_n^d$ and (ii) the mask $M^c \stackrel{\text{def}}{=} \prod_{i \in [1, n]} M_i^d$ (i.e., \mathcal{C} cannot distinguish two states \vec{v} and \vec{v}' if and only if $\forall i \in [1, n] : M_i^d(\vec{v}) = M_i^d(\vec{v}')$). Thus, the controller \mathcal{C} cannot distinguish a state $\vec{v} \in \mathcal{D}_V$ from the states in the set $\bigcap_{i=1}^n (M_i^d)^{-1}(M_i^d(\vec{v}))$.

The following property shows that \mathcal{C} is more permissive than \mathcal{C}_d .

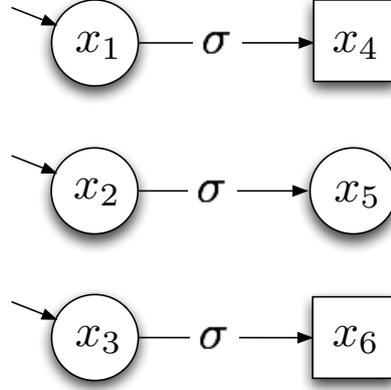


Figure 5.2 - Comparison between centralized and decentralized controllers.

Proposition 5.4

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, n observers $\langle Obs_i^d, M_i^d \rangle$ ($\forall i \in [1, n]$), an observer $\langle Obs^c, M^c \rangle$ (where (i) $Obs^c = Obs_1^d \cup \dots \cup Obs_n^d$ and (ii) $M^c = \prod_{i \in [1, n]} M_i^d$), n partitions $\Sigma_{i,c}^d \uplus \Sigma_{i,uc}^d$ of Σ ($\forall i \in [1, n]$), a partition $\Sigma_c^c \uplus \Sigma_{uc}^c$ of Σ (where $\Sigma_c^c = \bigcup_{i=1}^n \Sigma_{i,c}^d$ and $\Sigma_{uc}^c = \Sigma \setminus \Sigma_c^c$), and a set Bad , which represents a set of forbidden states, the centralized controller \mathcal{C} defined in (3.3) for the basic problem with the observer $\langle Obs^c, M^c \rangle$ and the partition $\Sigma_c^c \uplus \Sigma_{uc}^c$ is more permissive than the decentralized controller \mathcal{C}_d defined in (5.6) for the basic decentralized problem with the observers $\langle Obs_i^d, M_i^d \rangle$ and the partitions $\Sigma_{i,c}^d \uplus \Sigma_{i,uc}^d$ ($\forall i \in [1, n]$).

Proof

First, we can note that the sets $I(Bad)$ computed for the centralized and decentralized cases have the same value.

Next, we show that the centralized controller is as permissive as the decentralized controller by proving that $\forall \vec{v} \in \mathcal{D}_V, \forall \sigma \in \Sigma_c^c : \sigma \in \mathcal{S}(M^c(\vec{v})) \Rightarrow \sigma \in \mathcal{R}^{\mathcal{S}}(\mathcal{S}_1(M_1^d(\vec{v})), \dots, \mathcal{S}_n(M_n^d(\vec{v})))$. We suppose that $\sigma \in \mathcal{S}(M^c(\vec{v}))$. Then, there exists a state $\vec{v}' \notin I(Bad)$, which is undistinguishable from \vec{v} and from which a transition labeled by σ leads to $I(Bad)$. By definition of the mask M^c , \vec{v}' belongs to $\bigcap_{i=1}^n (M_i^d)^{-1}(M_i^d(\vec{v}))$, which implies that each local controller \mathcal{C}_i ($\forall i \in [1, n]$), which controls σ , proposes to forbid this action in the observation $M_i^d(\vec{v})$ (because $\vec{v}' \in (M_i^d)^{-1}(M_i^d(\vec{v}))$). Consequently, by definition of the fusion rule \mathcal{R} , we have that $\sigma \in \mathcal{R}^{\mathcal{S}}(\mathcal{S}_1(M_1^d(\vec{v})), \dots, \mathcal{S}_n(M_n^d(\vec{v})))$.

Finally, the system depicted in Figure 5.2 shows that the centralized controller \mathcal{C} can be strictly more permissive than the decentralized controller \mathcal{C}_d . In this system, (i) the set of states $X = \{x_i \mid i \in [1, 6]\}$, (ii) the set of initial states $X_0 = \{x_1, x_2, x_3\}$, (iii) the set $\Sigma = \{\sigma\}$ (this action can be controlled by all controllers considered in this example). The set of forbidden states Bad is equal to $\{x_4, x_6\}$. The decentralized controller is composed of two controllers \mathcal{C}_1 and \mathcal{C}_2 : the first one does not distinguish x_1 and x_2 , and the second one does not distinguish x_2 and x_3 . By definition of the observer of \mathcal{C} , this controller distinguishes all states.

The decentralized controller \mathcal{C}_d forbids the action σ in the states x_1, x_2 and x_3 , whereas the centralized controller \mathcal{C} only forbids σ in the states x_1 and x_3 , which implies that \mathcal{C} is strictly more permissive than \mathcal{C}_d .

A similar property can be obtained for the deadlock free case.

The following proposition gives a sufficient condition which ensures that the centralized and decentralized controllers have the same permissiveness. This condition is based on the concept of n -observability defined in [TKU94] and it intuitively means that if a controllable action is not forbidden by the centralized controller, then one of the local controllers \mathcal{C}_i of the decentralized controller proposes not to forbid it.

Proposition 5.5

Given an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$, n observers $\langle Obs_i^d, M_i^d \rangle$ ($\forall i \in [1, n]$), an observer $\langle Obs^c, M^c \rangle$ (where (i) $Obs^c = Obs_1^d \cup \dots \cup Obs_n^d$ and (ii) $M^c = \prod_{i \in [1, n]} M_i^d$), n partitions $\Sigma_{i,c}^d \uplus \Sigma_{i,uc}^d$ of Σ ($\forall i \in [1, n]$), a partition $\Sigma_c^c \uplus \Sigma_{uc}^c$ of Σ (where $\Sigma_c^c = \bigcup_{i=1}^n \Sigma_{i,c}^d$ and $\Sigma_{uc}^c = \Sigma \setminus \Sigma_c^c$), and a set Bad , which represents a set of forbidden states, the centralized controller, \mathcal{C} defined in (3.3) for the basic problem with the observer $\langle Obs^c, M^c \rangle$ and the partition $\Sigma_c^c \uplus \Sigma_{uc}^c$, and the decentralized controller \mathcal{C}_d , defined in (5.6) for the basic decentralized problem with the observers $\langle Obs_i^d, M_i^d \rangle$ and the partitions $\Sigma_{i,c}^d \uplus \Sigma_{i,uc}^d$ ($\forall i \in [1, n]$), have the same permissiveness if $\forall \vec{v} \notin I(Bad), \forall \sigma \in \Sigma_c^c : [\text{Post}_\sigma^T((M^c)^{-1}(M^c(\vec{v})) \setminus I(Bad)) \cap I(Bad) = \emptyset] \Rightarrow [\exists i \in \text{In}(\sigma) : \text{Post}_\sigma^T((M_i^d)^{-1}(M_i^d(\vec{v})) \setminus I(Bad)) \cap I(Bad) = \emptyset]$.

Proof

In this proof, we use the term *equivalence condition* to denote the above condition. By Proposition 5.4, we must only prove that the decentralized controller

is as permissiveness as the centralized controller. For that, we show that if the centralized controller allows an action σ ($\forall \sigma \in \Sigma$) in a state \vec{v} ($\forall \vec{v} \notin I(Bad)$), then this action is also allowed by the decentralized controller in the state \vec{v} . Two cases must be considered:

- $\sigma \in \Sigma_{uc}^c$: no local controller \mathcal{C}_i ($\forall i \in [1, n]$) controls σ and hence the decentralized controller allows the action σ in the state \vec{v} .
- $\sigma \in \Sigma_c^c$: by (3.2) and (3.4), an action σ' is forbidden by the centralized controller \mathcal{C} in the state \vec{v} if and only if there exists a state $\vec{v}' \notin I(Bad)$ such that (i) $M^c(\vec{v}) = M^c(\vec{v}')$ and (ii) $I(Bad)$ is reachable from \vec{v}' by a transition labeled by σ' . Therefore, since \mathcal{C} allows σ in the state \vec{v} , we have that $\text{Post}_\sigma^T((M^c)^{-1}(M^c(\vec{v})) \setminus I(Bad)) \cap I(Bad) = \emptyset$. By the equivalence condition, at least one local controller \mathcal{C}_i , which controls σ , knows that $I(Bad)$ is not reachable from $(M_i^d)^{-1}(M_i^d(\vec{v})) \setminus I(Bad)$ by a transition labeled by σ . Consequently, by (5.1) and (5.2), \mathcal{C}_i proposes to allow σ in the state \vec{v} , which implies that the fusion rule allows this action σ in the state \vec{v} .

Note that a similar property can be defined for the deadlock free case.

5.2 Synthesis of Modular Controllers

The decentralized framework, that we introduced in the previous section, does not take into account the fact that, in many applications, systems are often modeled by several components acting concurrently. This entails that the computation of the decentralized controller is performed on the whole system. In this section, we will show how, in some cases, the concurrent structure of the system can be exploited to locally perform the computation of a solution by avoiding the computation of the global system. To illustrate this point, let us consider the system \mathcal{T} depicted in Figure 3.1. This system has actually the same behavior as the parallel composition of the subsystems \mathcal{T}_1 and \mathcal{T}_2 depicted in Figure 5.3, and the corresponding controlled system could have been obtained by performing only local computations on each of these subsystems. As in the previous section, we focus on the state avoidance control problem, but now we assume that the system is given by a collection of subsystems acting concurrently. A solution to this control problem, exploiting the modularity of the system, has already

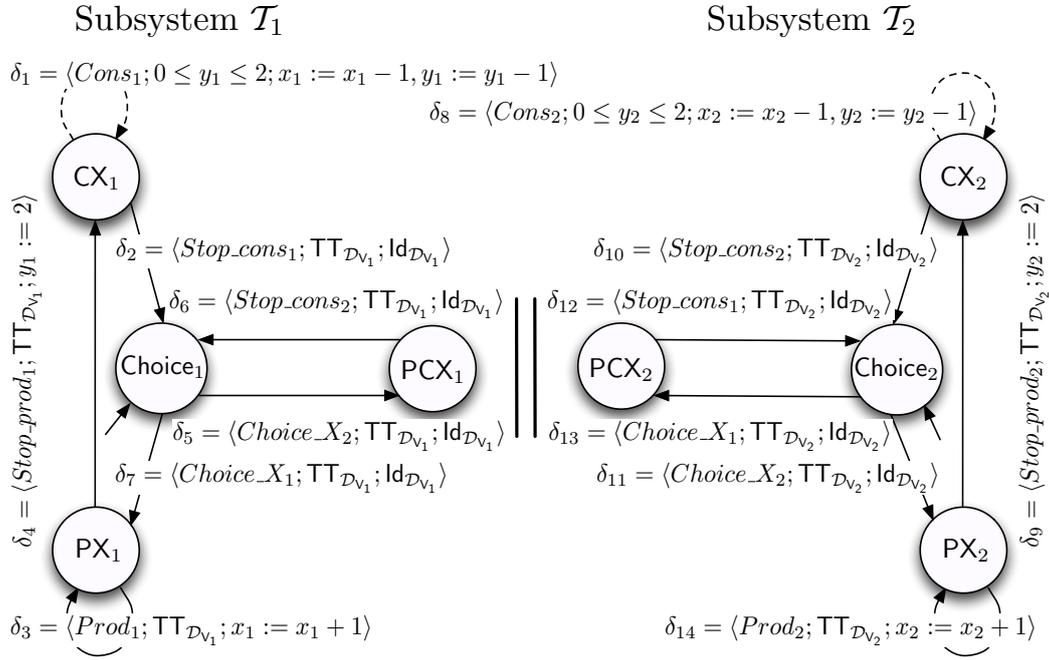


Figure 5.3 - Modular version of the producer and consumer example.

been given in [GM05]. However, this approach only holds for finite systems. We extend it to the case of infinite state systems in order to solve more efficiently our state avoidance control problem. Compared to [GM05], which computes a centralized controller, we keep the decentralized framework and compute a decentralized controller where each local controller has its own partial view of the system and its own set of controllable actions.

The remainder of this section is structured as follows. First, in section 5.2.1, we formally define the modular framework and the problem that we want to solve. Next, in section 5.2.2, we define a semi-algorithm for this problem. Finally, in section 5.2.3, we explain how to extend it by using of abstract interpretation techniques to obtain an effective algorithm.

5.2.1 Framework and State Avoidance Control Problem

Model of the System. The system to be controlled is given by a collection of n subsystems modeled by STS $\mathcal{T}_i = \langle V_i, \Theta_{0,i}, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) that interact with each other by synchronizing some actions, called *common (or shared) actions*. The global behavior resulting from this interaction is defined by an STS $\mathcal{T} =$

$\langle V, \Theta_0, \Sigma, \Delta \rangle$ obtained by composing the STS \mathcal{T}_i with the parallel composition operator. This operation gives the concurrent behavior of these STS with a synchronization on the common actions. The parallel composition of two STS is defined as follows:

Definition 5.6 (Parallel Composition of STS)

Let $\mathcal{T}_1 = \langle V_1, \Theta_{0,1}, \Sigma_1, \Delta_1 \rangle$ and $\mathcal{T}_2 = \langle V_2, \Theta_{0,2}, \Sigma_2, \Delta_2 \rangle$ be two STS such that $V_1 \cap V_2 = \emptyset$. The set Σ_s of shared actions is defined by $\Sigma_s \stackrel{\text{def}}{=} \Sigma_1 \cap \Sigma_2$. The parallel composition of \mathcal{T}_1 and \mathcal{T}_2 , denoted by $\mathcal{T}_1 || \mathcal{T}_2$, is given by the STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$ where (i) $V \stackrel{\text{def}}{=} V_1 \cup V_2$, (ii) $\Theta_0 \stackrel{\text{def}}{=} \Theta_{0,1} \times \Theta_{0,2}$, (iii) $\Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \Sigma_2$, and (iv) Δ is defined as follows for each $\sigma \in \Sigma$:

- if $\sigma \in \Sigma_1 \setminus \Sigma_s$, then, for each transition $\delta_1 = \langle \sigma, G_1, A_1 \rangle \in \Delta_1$, we define a new transition $\delta = \langle \sigma, G_1, A \rangle \in \Delta$ where the update function $A : \mathcal{D}_{V_1} \times \mathcal{D}_{V_2} \rightarrow \mathcal{D}_{V_1} \times \mathcal{D}_{V_2}$ is defined, for each $\vec{v}_1 \in \mathcal{D}_{V_1}$ and $\vec{v}_2 \in \mathcal{D}_{V_2}$, by $A(\langle \vec{v}_1, \vec{v}_2 \rangle) \stackrel{\text{def}}{=} \langle A_1(\vec{v}_1), \vec{v}_2 \rangle$.
- if $\sigma \in \Sigma_2 \setminus \Sigma_s$, then, for each transition $\delta_2 = \langle \sigma, G_2, A_2 \rangle \in \Delta_2$, we define a new transition $\delta = \langle \sigma, G_2, A \rangle \in \Delta$ where the update function $A : \mathcal{D}_{V_1} \times \mathcal{D}_{V_2} \rightarrow \mathcal{D}_{V_1} \times \mathcal{D}_{V_2}$ is defined, for each $\vec{v}_1 \in \mathcal{D}_{V_1}$ and $\vec{v}_2 \in \mathcal{D}_{V_2}$, by $A(\langle \vec{v}_1, \vec{v}_2 \rangle) \stackrel{\text{def}}{=} \langle \vec{v}_1, A_2(\vec{v}_2) \rangle$.
- if $\sigma \in \Sigma_s$, then, for each $\delta_1 = \langle \sigma, G_1, A_1 \rangle \in \Delta_1$ and $\delta_2 = \langle \sigma, G_2, A_2 \rangle \in \Delta_2$, we define a new transition $\delta = \langle \sigma, G_1 \cap G_2, A \rangle \in \Delta$ where the update function $A : \mathcal{D}_{V_1} \times \mathcal{D}_{V_2} \rightarrow \mathcal{D}_{V_1} \times \mathcal{D}_{V_2}$ is defined, for each $\vec{v}_1 \in \mathcal{D}_{V_1}$ and $\vec{v}_2 \in \mathcal{D}_{V_2}$, by $A(\langle \vec{v}_1, \vec{v}_2 \rangle) \stackrel{\text{def}}{=} \langle A_1(\vec{v}_1), A_2(\vec{v}_2) \rangle$.

Given an STS $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n$ (where $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle$ and $\mathcal{T}_i = \langle V_i, \Theta_{0,i}, \Sigma_i, \Delta_i \rangle$, for each $i \in [1, n]$), a state $\vec{v} \in \mathcal{D}_V$ of \mathcal{T} is thus given by an n -tuple $\langle \vec{v}_1, \dots, \vec{v}_n \rangle \in \mathcal{D}_{V_1} \times \dots \times \mathcal{D}_{V_n}$ (where each $\vec{v}_i \in \mathcal{D}_{V_i}$ gives the value of the variables of the system \mathcal{T}_i). The variables of the subsystems \mathcal{T}_i are not shared, which means that, for each $\delta_i = \langle \sigma_i, G_i, A_i \rangle \in \Delta_i$, the guard G_i and the assignment A_i of δ_i only use variables of the module \mathcal{T}_i ; in fact, the synchronization between the subsystems is achieved through the set Σ_s of shared actions where $\Sigma_s \stackrel{\text{def}}{=} \bigcup_{i \neq j \in [1, n]} (\Sigma_i \cap \Sigma_j)$. In the sequel, we call *product set* a product $\prod_{i=1}^n B_i$ of local sets $B_i \subseteq \mathcal{D}_{V_i}$.

Means of Control. The alphabet Σ_i of each subsystem \mathcal{T}_i ($\forall i \in [1, n]$) is partitioned into the set $\Sigma_{i,c}$ of controllable actions and the $\Sigma_{i,uc}$ of uncontrollable actions i.e., $\Sigma_i = \Sigma_{i,uc} \uplus \Sigma_{i,c}$. We also assume that the shared actions can be controlled by all controllers \mathcal{C}_i that share them, namely $\Sigma_s \cap \Sigma_{i,uc} = \emptyset$. This assumption allows us to fully exploit the structure of the system to be controlled by performing all computations locally (i.e., on each subsystem). The set Σ_c of actions, that can be controlled by at least one controller, is defined by $\Sigma_c \stackrel{\text{def}}{=} \bigcup_{i=1}^n \Sigma_{i,c}$ and the set $\Sigma_{i,uc}$ of actions, that cannot be controlled, is defined by $\Sigma_{uc} \stackrel{\text{def}}{=} \Sigma \setminus \Sigma_c = \bigcup_{i=1}^n \Sigma_{i,uc}$ (indeed, with the above assumptions, we have that $\Sigma \setminus \Sigma_c = \bigcup_{i=1}^n \Sigma_{i,uc}$). That also induces a partition of the set of transitions Δ into the set Δ_c (resp. $\Delta_{i,c}$) and the set Δ_{uc} (resp. $\Delta_{i,uc}$) in the following way: $\langle \sigma, G, A \rangle \in \Delta_c$ (resp. $\Delta_{i,c}$) if $\sigma \in \Sigma_c$ (resp. $\Sigma_{i,c}$), and $\langle \sigma, G, A \rangle \in \Delta_{uc}$ (resp. $\Delta_{i,uc}$) if $\sigma \in \Sigma_{uc}$ (resp. $\Sigma_{i,uc}$).

Means of Observation. In the decentralized framework, each local controller has a partial view of the global system. Here, we assume that there is one controller per module. We thus consider a special case of partial observation where each local controller \mathcal{C}_i has only a local view of the module, that it controls, and a perfect observation of all local variables of this subsystem (partial local observation could be considered, but gives an even more elaborate solution). Therefore, we associate the following observer with each local controller \mathcal{C}_i :

$$\langle Obs_i, M_i \rangle \tag{5.11}$$

where (i) $Obs_i \stackrel{\text{def}}{=} V_i$, and (ii) the mask $M_i : \mathcal{D}_V \rightarrow \mathcal{D}_{V_i}$ is defined, for each state $\langle \vec{v}_1, \dots, \vec{v}_i, \dots, \vec{v}_n \rangle \in \mathcal{D}_V$, by $M_i(\langle \vec{v}_1, \dots, \vec{v}_i, \dots, \vec{v}_n \rangle) \stackrel{\text{def}}{=} \vec{v}_i$. In fact, here we have not a partial observation, but a *shared* observation, where each state $\vec{v} = \langle \vec{v}_1, \dots, \vec{v}_i, \dots, \vec{v}_n \rangle \in \mathcal{D}_V$ is equal to the tuple of its observations $\langle M_1(\vec{v}), \dots, M_n(\vec{v}) \rangle$.

Definition of the State Avoidance Modular Control Problem. Again, we want to solve a state avoidance control problem, but a modular one. Therefore, knowing the modular structure of the system, we assume that the set Bad of forbidden states is decomposed according to this structure. This set is thus defined by a finite union of product sets:

$$Bad \stackrel{\text{def}}{=} \bigcup_{j=1}^m \left(\prod_{i=1}^n B_i^j \right) \tag{5.12}$$

where $B_i^j \subseteq \mathcal{D}_{V_i}$. Bad can therefore be seen as a disjunction of conjunctions of local predicates (i.e., of forbidden local states). This definition of Bad does not cover all possible subsets of states of the system, but it covers a lot of practical cases that one could want to handle.

Our aim is to synthesize a decentralized controller for the basic decentralized problem with the additional assumption that we know the structure of the system, and we want to exploit this knowledge to perform most of the computations locally. To avoid confusions, this problem is called *basic modular problem* and the decentralized controller, that we want to compute, is called *modular controller*.

Problem 5.3 (Basic Modular Problem)

Given n STS $\mathcal{T}_i = \langle V_i, \Theta_{0,i}, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$), n observers $\langle Obs_i, M_i \rangle$ ($\forall i \in [1, n]$) (defined as in (5.11)), an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle = \mathcal{T}_1 || \dots || \mathcal{T}_n$, and a set of states Bad (defined as in (5.12)), the basic modular problem consists in computing a modular controller $\mathcal{C}_M = \langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$ such that (i) $\text{Reachable}_{\Delta/\mathcal{C}_M}^{\mathcal{T}/\mathcal{C}_M}((\Theta_0)_{/\mathcal{C}_M}) \neq \emptyset$, and (ii) $\text{Reachable}_{\Delta/\mathcal{C}_M}^{\mathcal{T}/\mathcal{C}_M}((\Theta_0)_{/\mathcal{C}_M}) \cap Bad = \emptyset$.

Note that to fit the modular structure of the system the fusion rule \mathcal{R} will be different from the one defined in (5.3).

5.2.2 Semi-Algorithm for the Basic Modular Problem

Our algorithm, which computes modular controllers for the basic modular problem, is composed of two parts:

- we compute, using a fixpoint equation, the set $I(Bad)$ of states leading uncontrollably to Bad .
- we compute the modular controller \mathcal{C}_M . Each local controller of \mathcal{C}_M forbids, for each observation, the controllable actions that lead to $I(Bad)$ from a state compatible with this observation.

Within the modular framework, we want to find a way to compute the set $I(Bad)$ and the modular controller without having to compute the whole system. We detail these two steps below, but before that, we present some properties regarding the backward and coreachability operators.

Local Computation of the Backward and Coreachability Operators.

The fixpoint equation defining the set $I(Bad)$ (see (3.1)) is based on the function Pre and we present a property which allows us to locally compute this function (see [GM05]).

Proposition 5.6 ([GM05])

Given n STS $\mathcal{T}_i = \langle V_i, \Theta_{0,i}, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$), an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle = \mathcal{T}_1 || \dots || \mathcal{T}_n$, and a set of states $B_1 \times \dots \times B_n$ (where, for each $i \in [1, n]$, $B_i \subseteq \mathcal{D}_{V_i}$), we have that:

$$\text{Pre}_A^{\mathcal{T}}(B_1 \times \dots \times B_n) = \bigcup_{\sigma \in A} \text{Cpre}_{\sigma}^{\mathcal{T}_1}(B_1) \times \dots \times \text{Cpre}_{\sigma}^{\mathcal{T}_n}(B_n) \quad (5.13)$$

where, for each $i \in [1, n]$:

$$\text{Cpre}_{\sigma}^{\mathcal{T}_i}(B_i) = \begin{cases} \text{Pre}_{\sigma}^{\mathcal{T}_i}(B_i) & \text{if } \sigma \in \Sigma_i \\ B_i & \text{otherwise} \end{cases} \quad (5.14)$$

The distributivity of the function $\text{Pre}_A^{\mathcal{T}}$ is a direct consequence of the assumptions that we made on the parallel composition of the subsystems (there is no shared state variable). This property means that the $\text{Pre}_A^{\mathcal{T}}$ operator on the product set $B_1 \times \dots \times B_n$ can locally be computed on each set B_i .

By using this result, one can locally compute the set $\text{Coreach}_A^{\mathcal{T}}(B_1 \times \dots \times B_n)$, thanks to the following proposition:

Proposition 5.7 ([GM05])

Given n STS $\mathcal{T}_i = \langle V_i, \Theta_{0,i}, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$), an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle = \mathcal{T}_1 || \dots || \mathcal{T}_n$, a set of states $B_1 \times \dots \times B_n$ (where, for each $i \in [1, n]$, $B_i \subseteq \mathcal{D}_{V_i}$), and a set of actions $A \subseteq \Sigma \setminus \Sigma_s$, we have that:

$$\text{Coreach}_A^{\mathcal{T}}(B_1 \times \dots \times B_n) = \text{Coreach}_{A \cap \Sigma_1}^{\mathcal{T}_1}(B_1) \times \dots \times \text{Coreach}_{A \cap \Sigma_n}^{\mathcal{T}_n}(B_n) \quad (5.15)$$

Local Computation of the set $I(\text{Bad})$. The following proposition allows us to locally compute the set $I(\text{Bad})$ of states leading uncontrollably to Bad :

Proposition 5.8 ([GM05])

Given n STS $\mathcal{T}_i = \langle V_i, \Theta_{0,i}, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$), an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle = \mathcal{T}_1 || \dots || \mathcal{T}_n$ with the assumptions defined in section 5.2.1, a set of states $\bigcup_{j=1}^m B_1^j \times \dots \times B_n^j$ (where, for each $j \in [1, m]$ and for each $i \in [1, n]$, $B_i^j \subseteq \mathcal{D}_{V_i}$), we have that:

$$I(\text{Bad}) = \bigcup_{j=1}^m \left(\prod_{i=1}^n I_i(B_i^j) \right) \quad (5.16)$$

where $I_i(B_i^j) \stackrel{\text{def}}{=} \text{Coreach}_{\Sigma_{i,uc}}^{\mathcal{T}_i}(B_i^j)$. By the proof of Proposition 3.9, we also have that $I_i(B_i^j) = \text{lfp}^{\subseteq}(\lambda B. B_i^j \cup \text{Pre}_{\Sigma_{i,uc}}^{\mathcal{T}_i}(B_i^j))$.

Proof

This property is proved as follows :

$$\begin{aligned} & I(\text{Bad}) \\ &= \text{Coreach}_{\Sigma_{uc}}^{\mathcal{T}} \left(\bigcup_{j=1}^m B_1^j \times \dots \times B_n^j \right) \\ &= \bigcup_{j=1}^m \text{Coreach}_{\Sigma_{uc}}^{\mathcal{T}}(B_1^j \times \dots \times B_n^j), \\ & \quad \text{because } \text{Coreach}_{\Sigma_{uc}}^{\mathcal{T}}(B_1 \cup B_2) = \text{Coreach}_{\Sigma_{uc}}^{\mathcal{T}}(B_1) \cup \text{Coreach}_{\Sigma_{uc}}^{\mathcal{T}}(B_2) \\ &= \bigcup_{j=1}^m (\text{Coreach}_{\Sigma_{uc} \cap \Sigma_1}^{\mathcal{T}_1}(B_1^j) \times \dots \times \text{Coreach}_{\Sigma_{uc} \cap \Sigma_n}^{\mathcal{T}_n}(B_n^j)), \\ & \quad \text{by Proposition 5.7 (it can be used, because } \Sigma_{uc} \subseteq \Sigma \setminus \Sigma_s) \\ &= \bigcup_{j=1}^m (\text{Coreach}_{\Sigma_{1,uc}}^{\mathcal{T}_1}(B_1^j) \times \dots \times \text{Coreach}_{\Sigma_{n,uc}}^{\mathcal{T}_n}(B_n^j)), \text{ because, for each} \\ & \quad i \in [1, n], \Sigma_{i,uc} = \Sigma_i \cap \Sigma_{uc} \text{ (indeed, } \Sigma_{uc} = \bigcup_{i=1}^n \Sigma_{i,uc}) \\ &= \bigcup_{j=1}^m (I_1(B_1^j) \times \dots \times I_n(B_n^j)), \text{ because, for each } j \in [1, m] \text{ and for each} \\ & \quad i \in [1, n], I_i(B_i^j) = \text{Coreach}_{\Sigma_{i,uc}}^{\mathcal{T}_i}(B_i^j) \\ &= \bigcup_{j=1}^m \left(\prod_{i=1}^n I_i(B_i^j) \right) \end{aligned}$$

This proposition means that the fixpoint equation $I(\text{Bad})$ can be obtained by computing the fixpoint equation $I_i(B_i^j)$ (for each $j \in [1, m]$ and for each $i \in [1, n]$).

Local Computation of the Modular Controller $\mathcal{C}_M = \langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle$. We recall that each local controller \mathcal{C}_i ($\forall i \in [1, n]$) is only able to observe the variables V_i of the subsystem \mathcal{T}_i . Following the construction of the local controllers \mathcal{C}_i defined in subsections 5.1.2.1 and 5.1.3.1 for the decentralized case, we first investigate how the function $\widehat{\mathcal{F}}_i$, defined in (5.7), can locally be computed². For a set of states $B = \bigcup_{j=1}^m B^j$ (where $B^j = B_1^j \times \dots \times B_n^j$ is a product set), the following proposition explains how to locally compute (i.e., on \mathcal{T}_i) the function $\widehat{\mathcal{F}}_i$ for each product set B^j of B :

Proposition 5.9

Given n STS $\mathcal{T}_i = \langle V_i, \Theta_{0,i}, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$), an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle = \mathcal{T}_1 \parallel \dots \parallel \mathcal{T}_n$ with the assumptions defined in section 5.2.1, a set of states $B^j = B_1^j \times \dots \times B_n^j$ (where, for each $i \in [1, n]$, $B_i^j \subseteq \mathcal{D}_{V_i}$), and an action $\sigma \in \Sigma_i$, we have that:

$$\widehat{\mathcal{F}}_i(\sigma, B^j) = \begin{cases} \text{Pre}_\sigma^{\mathcal{T}_i}(B_i^j) & \text{if } \sigma \in \Sigma_{i,c} \\ \emptyset & \text{otherwise} \end{cases} \quad (5.17)$$

Proof

If $\sigma \notin \Sigma_{i,c}$, then $\widehat{\mathcal{F}}_i(\sigma, B^j) = \emptyset$ by (5.7). If $\sigma \in \Sigma_{i,c}$, then:

$$\begin{aligned} \widehat{\mathcal{F}}_i(\sigma, B^j) &= M_i(\text{Pre}_\sigma^{\mathcal{T}}(B^j)), \text{ by (5.7)} \\ &= M_i(\text{Cpre}_\sigma^{\mathcal{T}_1}(B_1^j) \times \dots \times \text{Cpre}_\sigma^{\mathcal{T}_i}(B_i^j) \times \dots \times \text{Cpre}_\sigma^{\mathcal{T}_n}(B_n^j)), \\ &\quad \text{by Proposition 5.6} \\ &= \text{Cpre}_\sigma^{\mathcal{T}_i}(B_i^j), \text{ by definition of } M_i \\ &= \text{Pre}_\sigma^{\mathcal{T}_i}(B_i^j), \text{ by (5.14) (because } \sigma \in \Sigma_i) \end{aligned}$$

We now explain how each local controller $\mathcal{C}_i = \langle \mathcal{S}_i, E_i \rangle$ of the modular controller \mathcal{C}_M is computed. According to Proposition 5.8, we write $I(\text{Bad}) = \bigcup_{j=1}^m I^j$ where $I^j = I_1^j \times \dots \times I_n^j$. To define \mathcal{S}_i and E_i , we take into account the decomposition of the system into subsystems and the decomposition of $I(\text{Bad})$ into a union of product sets. This leads us to a new characterization of these two elements. Indeed, let us consider the product set $I^j = I_1^j \times \dots \times I_n^j$ of $I(\text{Bad})$, a controllable

²We recall that $\widehat{\mathcal{F}}_i(\sigma, B)$ gives the set of observations for which \mathcal{C}_i proposes to forbid σ in order to prevent the system from reaching B . We use $\widehat{\mathcal{F}}_i$ instead of \mathcal{F}_i , because the first function can locally be computed unlike the second one.

action $\sigma \in \Sigma_c$ should be disabled in a state $\vec{v} = \langle \vec{v}_1, \dots, \vec{v}_n \rangle$ due to this product set whenever:

1. $\forall i \in [1, n]$ such that $\sigma \in \Sigma_i : \text{Post}_{\sigma}^{\mathcal{I}_i}(\vec{v}_i) \in I_i^j$, and
2. $\forall i \in [1, n]$ such that $\sigma \notin \Sigma_i : \vec{v}_i \in I_i^j$.

i.e., each subsystem, which has σ in its alphabet, can reach a forbidden state from \vec{v}_i by firing a transition labeled by σ and each subsystem, which has not σ in its alphabet, is already in a forbidden state. Thus, whenever $\sigma \notin \Sigma_i$, the supervisory function \mathcal{S}_i ($\forall i \in [1, n]$) should indicate whether $\vec{v}_i \in I_i^j$ in order to determine the global control to be applied to the system. In consequence, we formally define the local controller \mathcal{C}_i ($\forall i \in [1, n]$) as follows:

$$\mathcal{C}_i \stackrel{\text{def}}{=} \langle \mathcal{S}_i, E_i \rangle \quad (5.18)$$

where the elements \mathcal{S}_i and E_i are defined in the following way:

- For each state $\vec{v} = \langle \vec{v}_1, \dots, \vec{v}_i, \dots, \vec{v}_n \rangle \in \mathcal{D}_V$, the supervisory function \mathcal{S}_i only observes $\vec{v}_i \in \mathcal{D}_{Obs_i}$ and is given by the pair $\mathcal{S}_i(\vec{v}_i) \stackrel{\text{def}}{=} \langle \mathcal{P}_i(\vec{v}_i), \mathcal{B}_i(\vec{v}_i) \rangle$ where:
 - the function $\mathcal{P}_i(\vec{v}_i)$ gives the set of pairs $\langle j, \sigma \rangle$ such that \mathcal{C}_i proposes to forbid the action σ in the state \vec{v}_i because of the product set I^j ; compared to the decentralized approach, we memorize the index of the product set for which σ must be forbidden in the state \vec{v}_i :

$$\mathcal{P}_i(\vec{v}_i) \stackrel{\text{def}}{=} \{ \langle j, \sigma \rangle \mid (\sigma \in \Sigma_{i,c}) \wedge (j \in [1, m]) \wedge (\vec{v}_i \in \widehat{\mathcal{F}}_i(\sigma, I^j)) \} \quad (5.19)$$

- the function $\mathcal{B}_i(\vec{v}_i)$ gives the set of indices j such that \vec{v}_i belongs to the forbidden set I_i^j :

$$\mathcal{B}_i(\vec{v}_i) \stackrel{\text{def}}{=} \{ j \mid (j \in [1, m]) \wedge (\vec{v}_i \in I_i^j) \} \quad (5.20)$$

As explained above, this information will be used by the fusion rule to know whether $\vec{v}_i \in I_i^j$.

- The set $E_i \stackrel{\text{def}}{=} \{ \langle j, E_i^j \rangle \mid (j \in [1, m]) \wedge (E_i^j = I_i^j) \}$.

We also need to redefine the elements \mathcal{R}^S and \mathcal{R}^E of the fusion rule \mathcal{R} (denoted by \mathcal{R}_M^S and \mathcal{R}_M^E hereafter to stress the modular architecture) to take into account the decomposition of $I(\text{Bad})$ into product sets as well as the new definition of the supervisory function. The fusion rule \mathcal{R} is defined by:

$$\mathcal{R} \stackrel{\text{def}}{=} \langle \mathcal{R}_M^S, \mathcal{R}_M^E \rangle \quad (5.21)$$

where, for each $\vec{v} = \langle \vec{v}_1, \dots, \vec{v}_n \rangle \in \mathcal{D}_V$, we define:

- the fusion rule \mathcal{R}_M^S by:

$$\mathcal{R}_M^S(\mathcal{S}_1(\vec{v}_1), \dots, \mathcal{S}_n(\vec{v}_n)) \stackrel{\text{def}}{=} \{ \sigma \in \Sigma_c \mid \exists j \in [1, m] : [(\forall i \in [1, n] \text{ such that } \sigma \in \Sigma_i : \langle j, \sigma \rangle \in \mathcal{P}_i(\vec{v}_i)) \wedge (\forall i \in [1, n] \text{ such that } \sigma \notin \Sigma_i : j \in \mathcal{B}_i(\vec{v}_i))] \} \quad (5.22)$$

Thus, as explained above, a controllable action σ is forbidden in the state \vec{v} if there exists a product set $I^j = I_1^j \times \dots \times I_n^j$ such that (i) each subsystem \mathcal{T}_i , which has σ in its alphabet, can reach I_i^j from \vec{v}_i by firing a transition labeled by σ , and (ii) each subsystem \mathcal{T}_i , which has not σ in its alphabet, is in a state of I_i^j .

- the fusion rule $\mathcal{R}_M^\mathcal{E}$ as follows: $\vec{v} \in \mathcal{R}_M^\mathcal{E}(E_1, \dots, E_n)$ if and only if $\exists j \in [1, m] : \vec{v} \in E_1^j \times \dots \times E_n^j$ (where, for each $i \in [1, n]$, $\langle j, E_i^j \rangle \in E_i$). We can remark that, as expected, $\mathcal{R}_M^\mathcal{E}$ forbids the states of $I(\text{Bad})$.

Finally, the modular controller \mathcal{C}_M is defined by the n local controllers \mathcal{C}_i given in (5.18) and by the fusion rule given in (5.21):

$$\mathcal{C}_M \stackrel{\text{def}}{=} \langle (\mathcal{C}_i)_{i=1}^n, \mathcal{R} \rangle \quad (5.23)$$

The set $I(\text{Bad})$ and the control functions $\widehat{\mathcal{F}}_i$ are computed *offline*. Next, during the execution of the system, when the system arrives in a state $\vec{v} = \langle \vec{v}_1, \dots, \vec{v}_n \rangle \in \mathcal{D}_V$, each local controller \mathcal{C}_i receives the observation \vec{v}_i and computes *online* the pair $\mathcal{S}_i(\vec{v}_i) = \langle \mathcal{P}_i(\vec{v}_i), \mathcal{B}_i(\vec{v}_i) \rangle$; this computation is based on the functions $\widehat{\mathcal{F}}_i$. The fusion rule $\mathcal{R}_M^S(\mathcal{S}_1(\vec{v}_1), \dots, \mathcal{S}_n(\vec{v}_n))$ is then computed *online* from these control decisions and gives the actions that the system cannot execute.

Example 5.3

We compute a modular controller $\mathcal{C}_M = \langle (\mathcal{C}_1, \mathcal{C}_2), \mathcal{R} \rangle$ for the subsystems \mathcal{T}_1 and \mathcal{T}_2 depicted in Figure 5.3. The system $\mathcal{T}_i = \langle V_i, \Theta_{0,i}, \Sigma_i, \Delta_i \rangle$ (for $i = 1, 2$) has a tuple $V_i = \langle \ell_i, x_i, y_i \rangle$ of three variables where (i) $\ell_i \in \{\text{Choice}_i, \text{PX}_i, \text{CX}_i, \text{PCX}_i\}$ gives the current location of \mathcal{T}_i , (ii) $x_i \in \mathbb{Z}$ gives the number of parts X_i , and (iii) $y_i \in \mathbb{Z}$ gives the number of parts X_i that can be produced. The system \mathcal{T}_i does not control the action Cons_i (i.e., $\Sigma_{i,uc} = \{\text{Cons}_i\}$) and its initial state is $\langle \text{Choice}_i, 0, 0 \rangle$. The global system $\mathcal{T} = \mathcal{T}_1 \parallel \mathcal{T}_2 = \langle V, \Theta_0, \Sigma, \Delta \rangle$ has thus a tuple $V = \langle \langle \ell_1, x_1, y_1 \rangle, \langle \ell_2, x_2, y_2 \rangle \rangle$ of six variables. The modular controller must prevent the system \mathcal{T} from reaching the set Bad defined by the product set $B_1^1 \times B_2^1$ where $B_i^1 = \{ \langle \text{CX}_i, x_i, y_i \rangle \mid (x_i \leq 10) \wedge (y_i \in [0, 2]) \}$ (for $i = 1, 2$). The computation of the set $I(\text{Bad}) = I_1(B_1^1) \times I_2(B_2^1)$ gives (for $i = 1, 2$):

$$I_i(B_i^1) = B_i^1 \cup \{\langle \mathbf{CX}_i, x_i, y_i \rangle \mid [(x_i \leq 11) \wedge (y_i \in [1, 2])] \vee [(x_i \leq 12) \wedge (y_i = 2)]\}$$

Next, the computation of the control function $\widehat{\mathcal{F}}_i$ (for $i = 1, 2$) gives:

$$\widehat{\mathcal{F}}_i(\sigma, I(\text{Bad})) = \begin{cases} \{\langle \mathbf{PX}_i, x_i, y_i \rangle \mid x_i \leq 12\} & \text{if } \sigma = \text{Stop_prod}_i \\ \emptyset & \text{otherwise} \end{cases}$$

The supervisory function $\mathcal{S}_i = \langle \mathcal{P}_i, \mathcal{B}_i \rangle$ (for $i = 1, 2$) is then defined as follows:

$$\mathcal{P}_i(\vec{v}_i) = \begin{cases} \{\langle 1, \text{Stop_prod}_i \rangle\} & \text{if } \vec{v}_i \in \{\langle \mathbf{PX}_i, x_i, y_i \rangle \mid x_i \leq 12\} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathcal{B}_i(\vec{v}_i) = \begin{cases} \{1\} & \text{if } \vec{v}_i \in I_i(B_i^1) \\ \emptyset & \text{otherwise} \end{cases}$$

The fusion rule $\mathcal{R}_M^{\mathcal{S}}$ forbids:

- the action Stop_prod_1 when the subsystem \mathcal{T}_1 is in a state $\vec{v}_1 \in \{\langle \mathbf{PX}_1, x_1, y_1 \rangle \mid x_1 \leq 12\}$ and the subsystem \mathcal{T}_2 is in a state $\vec{v}_2 \in \{\langle \mathbf{CX}_2, x_2, y_2 \rangle \mid [(x_2 \leq 10) \wedge (y_2 \in [0, 2])] \vee [(x_2 \leq 11) \wedge (y_2 \in [1, 2])] \vee [(x_2 \leq 12) \wedge (y_2 = 2)]\}$.
- the action Stop_prod_2 when the subsystem \mathcal{T}_1 is in state $\vec{v}_1 \in \{\langle \mathbf{CX}_1, x_1, y_1 \rangle \mid [(x_1 \leq 10) \wedge (y_1 \in [0, 2])] \vee [(x_1 \leq 11) \wedge (y_1 \in [1, 2])] \vee [(x_1 \leq 12) \wedge (y_1 = 2)]\}$ and the subsystem \mathcal{T}_2 is in a state $\vec{v}_2 \in \{\langle \mathbf{PX}_2, x_2, y_2 \rangle \mid x_2 \leq 12\}$.

Proposition 5.10

Given n STS $\mathcal{T}_i = \langle V_i, \Theta_{0,i}, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$), an STS $\mathcal{T} = \langle V, \Theta_0, \Sigma, \Delta \rangle = \mathcal{T}_1 \parallel \dots \parallel \mathcal{T}_n$ with the assumptions defined in section 5.2.1, and a set of states $\bigcup_{j=1}^m B_1^j \times \dots \times B_n^j$ (where, for each $j \in [1, m]$ and for each $i \in [1, n]$, $B_i^j \subseteq \mathcal{D}_{V_i}$), the modular controller \mathcal{C}_M , defined in (5.23), solves the basic modular problem (when $\Theta_0 \not\subseteq \mathcal{R}_M^{\mathcal{E}}(E_1, \dots, E_n)$).

Proof

We prove by induction on the length ℓ of the sequences of transitions (these sequences begin in an initial state) that $\text{Reachable}_{\Delta/\mathcal{C}_M}^{\mathcal{T}/\mathcal{C}_M}((\Theta_0)_{/\mathcal{C}_M}) \cap I(\text{Bad}) = \emptyset$. Since $\text{Bad} \subseteq I(\text{Bad})$, this property implies that $\text{Reachable}_{\Delta/\mathcal{C}_M}^{\mathcal{T}/\mathcal{C}_M}((\Theta_0)_{/\mathcal{C}_M}) \cap \text{Bad} = \emptyset$:

- *Base case* ($\ell = 0$): the controlled system $\mathcal{T}/\mathcal{C}_M$ starts its execution in a state \vec{v} , which belongs to Θ_0 and which is not in $\mathcal{R}_M^{\mathcal{E}}(E_1, \dots, E_n)$. Thus, this state does not belong to $I(\text{Bad})$.
- *Induction case*: we suppose that the proposition holds for the sequences of transitions of length less than or equal to ℓ and we prove that this property remains true for the sequences of transitions of length $\ell + 1$. By induction hypothesis, each state $\vec{v} = \langle \vec{v}_1, \dots, \vec{v}_n \rangle$ reachable by a sequence of transitions of length ℓ does not belong to $I(\text{Bad})$ and we show that each transition $\delta = \langle \sigma, G, A \rangle \in \Delta$, which can lead to a state $\vec{v}' = \langle \vec{v}'_1, \dots, \vec{v}'_n \rangle \in I(\text{Bad})$ from this state \vec{v} in the system \mathcal{T} , cannot be fired from \vec{v} in the controlled system $\mathcal{T}/\mathcal{C}_M$. For that, we consider two cases; let $I^j = I_1^j \times \dots \times I_n^j$ be a product set of $I(\text{Bad})$ such that $\vec{v}' \in I^j$:
 - either $\delta \in \Delta_{uc}$, then $\vec{v} \in I(\text{Bad})$ (by (3.1)), which is impossible by hypothesis.
 - or $\delta \in \Delta_c$, then we show that this transition is forbidden by the fusion rule. We can first remark that, for any $i \in [1, n]$, if $\sigma \in \Sigma_i$ and if $\sigma \in \Sigma_c$, then $\sigma \in \Sigma_{i,c}$. Next, we prove the desired property by reasoning as follows:
 - for each $i \in [1, n]$ such that $\sigma \notin \Sigma_i$, we have that $\vec{v}_i = \vec{v}'_i$ (by definition of the parallel composition of STS), which implies that $\vec{v}_i \in I_i^j$, because $\vec{v}' \in I^j$. Therefore, by (5.20), we have that $\forall i \in [1, n]$ such that $\sigma \notin \Sigma_i : j \in \mathcal{B}_i(\vec{v}_i)$.
 - for each $i \in [1, n]$ such that $\sigma \in \Sigma_i$, we have that $\vec{v}_i \in \text{Pre}_{\sigma}^{\mathcal{T}_i}(\vec{v}'_i)$ (by definition of the parallel composition of STS), which implies that $\vec{v}_i \in \text{Pre}_{\sigma}^{\mathcal{T}_i}(I_i^j)$, because $\vec{v}' \in I^j$. Therefore, by (5.17), and (5.19) we have that $\forall i \in [1, n]$ such that $\sigma \in \Sigma_i : \langle j, \sigma \rangle \in \mathcal{P}_i(\vec{v}_i)$.

Consequently, by (5.22), we have that σ is forbidden by the fusion rule $\mathcal{R}_M^{\mathcal{S}}$ in the state \vec{v} .

As explained in the introduction of this chapter, the advantage of the modular approach w.r.t. the decentralized approach is that we exploit the structure of the distributed system to be controlled to efficiently compute controllers. To exploit this structure, we defined the controller and the fusion rule differently from the decentralized approach. This implies that the controlled behaviors resulting from

the decentralized and modular controls are not necessarily identical and an open question consists in comparing the permissiveness of these approaches.

5.2.3 Effective Algorithm for the Basic Modular Problem

The effective computation is here straightforward. We compute, for each $i \in [1, n]$ and for each $j \in [1, m]$, an overapproximation $I'_i(B_i^j)$ of the fixpoint equation $I_i(B_i^j)$ by using the approach described in section 3.4.2. Next, we transpose the control function $\widehat{\mathcal{F}}_i$ ($\forall i \in [1, n]$) into the abstract lattice and we obtain the abstract function $\widehat{\mathcal{F}}_i^\sharp$ (the computation of this function may require overapproximations). Finally, we define our modular controller \mathcal{C}_M as in (5.23) by using $I'_i(B_i^j)$ ($\forall i \in [1, n]$ and $\forall j \in [1, m]$) and $\widehat{\mathcal{F}}_i^\sharp$ ($\forall i \in [1, n]$) instead of $I_i(B_i^j)$ and $\widehat{\mathcal{F}}_i$.

5.3 Experimental Evaluation

Our algorithms, that synthesize decentralized and modular controllers, have respectively been implemented in our tool SMACS (Symbolic MAsked Controller Synthesis) and Mod-SMACS (Modular Symbolic MAsked Controller Synthesis). A detailed description of SMACS is given in appendix A. The tool Mod-SMACS has a lot of similarities with SMACS and is thus not described.

We define several examples and present the solutions computed by SMACS or Mod-SMACS for these systems in order to experimentally evaluate our method. Our experiments are performed with the lattice of convex polyhedra.

Toy Example. We consider the system depicted in Figure 3.14. As a reminder, this STS has explicit locations $\ell \in \{\ell_i \mid i \in [0, 8]\}$ and two integer variables x and y . A state of the system is a triple $\langle \ell, x, y \rangle$. The actions c_i (for $i \in [0, 3]$) are controllable and the actions u_i (for $i \in [0, 1]$) are uncontrollable. The initial state is $\langle \ell_0, 0, 0 \rangle$ and the set *Bad* is defined by $\{\langle \ell_6, k_1, k_2 \rangle \mid k_1, k_2 \in \mathbb{Z}\}$. We consider several cases:

- The local controller \mathcal{C}_1 has a full observation of the system and the local controller \mathcal{C}_2 does not distinguish the locations ℓ_3 and ℓ_4 (i.e., $\forall x, y \in \mathbb{Z} : M_2(\langle \ell_3, x, y \rangle) = M_2(\langle \ell_4, x, y \rangle)$). We first suppose that the deadlock free property must not be ensured. SMACS computes a set $I(\text{Bad}) = \text{Bad} \cup \{\langle \ell_2, x, y \rangle \mid 10 \leq x \leq 1000\}$. To prevent the system from reaching this set, it defines a controller \mathcal{C}_1 that disables the action c_0 in the location ℓ_3 when $(10 \leq x \leq 1000) \wedge (y \geq x)$ and a controller \mathcal{C}_2 that disables

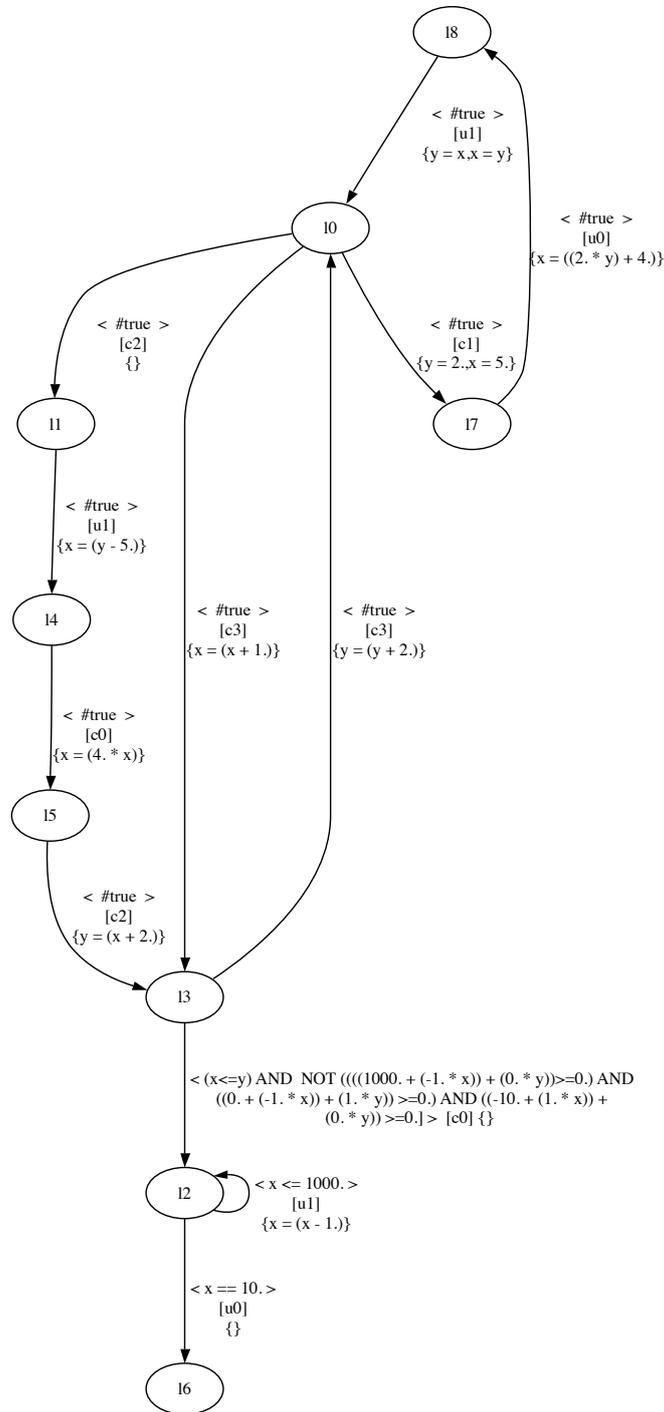


Figure 5.4 - Controlled system generated by SMACS for the toy example when \mathcal{C}_1 has a full observation of the system, \mathcal{C}_2 does not distinguish the locations ℓ_3 and ℓ_4 , and the deadlock free property is not ensured.

the action c_0 in the locations ℓ_3 and ℓ_4 when $(10 \leq x \leq 1000) \wedge (y \geq x)$. The global control is similar to the control policy of \mathcal{C}_1 . The graphical output of the controlled system generated by SMACS is depicted in Figure 5.4. When the deadlock free property must be ensured, SMACS computes a set $I_{df}(Bad) = Bad \cup \{\langle \ell_2, x, y \rangle \mid x \geq 10\}$. The states in the set $\{\langle \ell_2, x, y \rangle \mid 10 \leq x \leq 1000\}$ are forbidden, because they uncontrollably lead to Bad and the states in the set $\{\langle \ell_2, x, y \rangle \mid x > 1000\}$ are forbidden, because they are in deadlock. SMACS defines a controller \mathcal{C}_1 that forbids the action c_0 in the location ℓ_3 when $(x \geq 10) \wedge (y \geq x)$ and a controller \mathcal{C}_2 that forbids the action c_0 in the locations ℓ_3 and ℓ_4 when $(x \geq 10) \wedge (y \geq x)$. Again, the global control is similar to the control policy of \mathcal{C}_1 .

- The local controller \mathcal{C}_1 has a full observation of the system and the local controller \mathcal{C}_2 does not observe the variable x (i.e., for each $\vec{v} = \langle \ell, x, y \rangle \in \mathcal{D}_V$, the set of states that \mathcal{C}_2 does not distinguish from \vec{v} is given by $\{\langle \ell_2, x', y \rangle \mid x' \in \mathbb{Z}\}$). We first suppose that the deadlock free property must not be ensured. SMACS computes a set $I(Bad) = Bad \cup \{\langle \ell_2, x, y \rangle \mid 10 \leq x \leq 1000\}$. Next, it defines a controller \mathcal{C}_1 that forbids the action c_0 in the location ℓ_3 when $(10 \leq x \leq 1000) \wedge (y \geq x)$ and a controller \mathcal{C}_2 that forbids the action c_0 in the location ℓ_3 when $y \geq 10$. The global control is similar to the control policy of \mathcal{C}_1 . When the deadlock free property must be ensured, SMACS computes a set $I_{df}(Bad) = Bad \cup \{\langle \ell_2, x, y \rangle \mid x \geq 10\}$. Next, SMACS defines a controller \mathcal{C}_1 that forbids the action c_0 in the location ℓ_3 when $(x \geq 10) \wedge (y \geq x)$ and a controller \mathcal{C}_2 which forbids the action c_0 in the location ℓ_3 when $(y \geq 10)$. The global control is similar to the control policy of \mathcal{C}_1 .
- The local controller \mathcal{C}_1 does not distinguish the locations ℓ_3 and ℓ_4 , and the local controller \mathcal{C}_2 does not observe the variable x . We first suppose that the deadlock free property must not be ensured. SMACS computes a set $I(Bad) = Bad \cup \{\langle \ell_2, x, y \rangle \mid 10 \leq x \leq 1000\}$. Next, it defines a controller \mathcal{C}_1 that forbids the action c_0 in the locations ℓ_3 and ℓ_4 when $(10 \leq x \leq 1000) \wedge (y \geq x)$ and a controller \mathcal{C}_2 that forbids the action c_0 in the location ℓ_3 when $y \geq 10$. The global control consists in forbidding the action c_0 in the location ℓ_3 when $(10 \leq x \leq 1000) \wedge (y \geq x)$. When the deadlock free property must be ensured, SMACS computes a set $I_{df}(Bad) = Bad \cup \{\langle \ell_2, x, y \rangle \mid x \geq 10\}$. To prevent the system from reaching this set, SMACS defines a controller \mathcal{C}_1 that forbids the action c_0 in the locations ℓ_3 and ℓ_4 when $(x \geq 10) \wedge (y \geq x)$ and a controller \mathcal{C}_2 that forbids the action c_0 in the location ℓ_3 when $(y \geq 10)$. The global control

System	Number of Variables	Time (seconds)	Maximal Memory (MB)
Toy example	130	9.4	72
	170	21.1	84
	210	35.9	96
	250	58.9	108
	290	93.3	144

Table 5.1 - Time (in seconds) and memory (in MB) used by SMACS to synthesize a decentralized controller (composed of two local controllers) for modified versions of the toy example.

consists in forbidding the action c_0 in the location ℓ_3 when $(x \geq 10) \wedge (y \geq x)$.

All these computations are done in few ms. Next, we consider modified versions of the toy example where we add variables to make more complex the system to be controlled. The set of forbidden states is defined over these new variables and SMACS must compute a local controller \mathcal{C}_1 (this controller does not distinguish the locations ℓ_3 and ℓ_4) and a local controller \mathcal{C}_2 (this controller does not observe the variable x) which satisfy this specification. The results are reported in Table 5.1. For example, when the system has 170 variables, SMACS computes local controllers \mathcal{C}_1 and \mathcal{C}_2 which fulfill the requirements in 21.1 seconds and uses, for that, 84 MB of RAM.

Producer and Consumer. We consider a modified version of the producer and consumer example depicted in Figure 3.1, where the transitions δ_1 , δ_4 , δ_5 and δ_8 are respectively replaced by $\langle \text{Cons}; 0 \leq y \leq 500; x := x - 1, y := y - 1 \rangle$, $\langle \text{Stop_prod}; \top\top_{\mathcal{D}_V}; y := 500 \rangle$, $\langle \text{Cons}' ; 0 \leq y' \leq 500; x' := x' - 1, y' := y' - 1 \rangle$, and $\langle \text{Stop_prod}' ; \top\top_{\mathcal{D}_V}; y' := 500 \rangle$, which makes the system more complex to be controlled. The set *Bad* of forbidden states is defined by $\{ \langle \text{CX}, x, x', y, y' \rangle \mid (x \leq 10) \wedge (y \in [0, 500]) \} \cup \{ \langle \text{CX}', x, x', y, y' \rangle \mid (x' \leq 10) \wedge (y' \in [0, 500]) \}$. The aim is to synthesize two local controllers \mathcal{C}_1 and \mathcal{C}_2 which satisfy the control specification. \mathcal{C}_1 does not control the actions *Cons*, *Cons'* and *Stop_prod'*, and \mathcal{C}_2 does not control the actions *Cons*, *Cons'* and *Stop_prod*. We consider several cases:

- The local controller \mathcal{C}_1 does not observe the variables x and y (i.e., for each $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, the set of states that \mathcal{C}_1 does not distinguish from \vec{v} is given by $\{ \langle \ell, x_1, x', y_1, y' \rangle \mid x_1, y_1 \in \mathbb{Z} \}$), the local controller \mathcal{C}_2 has a full observation of the system, and the deadlock free property is not ensured.

SMACS defines a controller \mathcal{C}_1 which always forbids the action *Stop_prod* in the location PX and a controller \mathcal{C}_2 which forbids the action *Stop_prod'* in the location PX' when $x' \leq 510$. The global control consists in forbidding the action *Stop_prod* in the location PX and the action *Stop_prod'* in the location PX' when $x' \leq 510$.

- The local controller \mathcal{C}_1 has a full observation of the system, the controller local \mathcal{C}_2 does not observe the variable x' in the interval $[500, 600]$ (i.e., for each state $\vec{v} = \langle \ell, x, x', y, y' \rangle \in \mathcal{D}_V$, if $x' \notin [500, 600]$, then \mathcal{C}_2 perfectly observes this state, otherwise the set of states that \mathcal{C}_2 does not distinguish from \vec{v} is given by $\{\langle \ell, x, x'_1, y, y' \rangle \mid x'_1 \in [500, 600]\}$), and the deadlock free property is not ensured. SMACS defines a controller \mathcal{C}_1 which forbids the action *Stop_prod* in the location PX when $x \leq 510$ and a controller \mathcal{C}_2 which forbids the action *Stop_prod'* in the location PX' when $x' \leq 600$. The global control consists in forbidding the action *Stop_prod* in the location PX when $x \leq 510$ and the action *Stop_prod'* in the location PX' when $x' \leq 600$.
- The local controller \mathcal{C}_1 does not observe the variables x and y , the local controller \mathcal{C}_2 does not observe the variables x' in the interval $[500, 600]$, and the deadlock free property is not ensured. SMACS defines a controller \mathcal{C}_1 which always forbids the action *Stop_prod* in the location PX and a controller \mathcal{C}_2 which forbids the action *Stop_prod'* in the location PX' when $x' \leq 600$. The global control consists in forbidding the action *Stop_prod* in the location PX and the action *Stop_prod'* in the location PX' when $x' \leq 600$.

All these computations are done in few ms. Then, we consider modified versions of the producer and consumer example. The system is made more complex by producing n (where $n > 0$) kinds of parts. The production of each kind of parts requires the definition of two variables x_i and y_i . The control requirements consist in ensuring that there are at least 11 parts of each kind. SMACS must compute n local controllers \mathcal{C}_i ($\forall i \in [1, n]$). Each controller \mathcal{C}_i does not observe the value of the variable x_i when it belongs to the interval $[400, 600]$. The results are reported in Table 5.2. For example, when the system has 80 variables (i.e., there are 40 kinds of parts), SMACS computes a solution which fulfills the requirements in 50.2 seconds and uses, for that, 168 MB of RAM.

Comparison Between the Modular and Decentralized Controllers.

Now, we compare our modular and decentralized controllers. The experiments have been done on the producer and consumer example and are reported in

System	Number of Variables	Time (seconds)	Maximal Memory (MB)
Producer and consumer example	40	4.4	84
	80	50.2	168
	120	231.6	348
	140	404.4	456
	160	686.9	660

Table 5.2 - Time (in seconds) and memory (in MB) used by SMACS to synthesize a decentralized controller (composed of two local controllers) for modified versions of the producer and consumer example.

Table 5.3. For the decentralized case, we consider the producer and consumer system described above and producing n kinds of parts X_i ($\forall i \in [1, n]$). The decentralized controller must ensure that there are at least 11 parts of each kind and is composed of n local controllers \mathcal{C}_i . Each local controller \mathcal{C}_i can only observe the variables ℓ, x_i and y_i involved in the production of the parts of kind X_i . Note that these masks are different from the ones used in the experiments reported in Table 5.2. This explains why the time performance is not identical in Tables 5.2 and 5.3. For the modular case, the system to be controlled is composed of n subsystems \mathcal{T}_i . Each subsystem \mathcal{T}_i produces the parts of kind X_i and the modular controller must ensure that there are at least 11 parts of each kind. Our experiments show that the performance (time and memory) of the modular controller is better than the one of the decentralized controller (see Table 5.3) and that both controllers are always \preceq_p -maximal. In this case the controlled behaviors resulting from the decentralized and modular approaches are identical. For example, when 100 kinds of parts must be produced, the modular controller computes the solution 50 times faster and uses 100 times less memory.

System	Number of parts to be produced	Decentralized		Modular	
		Time (seconds)	Maximal Memory (MB)	Time (seconds)	Maximal Memory (MB)
Producer and consumer example	20	1.3	84	0.2	< 12
	30	5	108	0.5	< 12
	40	12.7	168	0.7	< 12
	50	28	228	1.2	< 12
	60	53.5	300	1.5	< 12
	70	93.7	444	2.2	< 12
	80	153.8	612	3.2	< 12
	90	240.8	852	6.2	< 12
	100	362.7	1128	7.1	< 12

Table 5.3 - Time (in seconds) and memory (in MB) used by SMACS and Mod-SMACS to control modified versions of the producer and consumer example.

Chapter 6

Distributed Supervisory Control of FIFO Systems

 In chapter 5, we were interested in the decentralized and modular controls of infinite state systems. These methods are useful for the control of distributed systems with synchronous communications. Both methods assume the existence of a synchronization mechanism (fusion rule) which allows the decentralized (resp. modular) controller to collect the control decisions of each local controller and to define the global control policy from these local decisions.

In this chapter, we consider the control of a more complex class of distributed systems: the distributed systems with asynchronous communications. These systems are generally composed of several subsystems that can be located at different places in such a way that communications between them can no longer be assumed as instantaneous. In this context, one has to take into account the duration of the communications and thus the asynchronous nature of the communication events. The decentralized and modular frameworks cannot then be used for the control of these systems, since their synchronization mechanism, which defines the global control policy, is no more valid in this context.

In this chapter, we thus propose a method for the control of distributed systems with asynchronous communications. In this method the system to be controlled is composed of n subsystems that asynchronously communicate

through reliable *unbounded* FIFO channels. These subsystems are modeled by *communicating finite state machines* [BZ83a] that explicitly represent the communications of a distributed system. Each subsystem is controlled by a *local* controller. The local controllers can communicate with each other through reliable *unbounded* FIFO channels. These communications allow them to exchange information to refine their knowledge of the global state of the system and hence their control policy. The state space of the distributed system to be controlled can be *infinite*, because the FIFO channels are unbounded. Moreover, the state avoidance control problem that we want to solve is *undecidable* and hence we use abstract interpretation techniques to ensure the termination of the computations. The used techniques to obtain an effective algorithm are actually the same as the ones used in chapters 3, 4, and 5, except that we perform the computations in a different abstract lattice. This lattice is the one of *regular languages*, which allows us to abstract the content of FIFO channels by regular languages.

Related Works. In [Tri02], Tripakis studies the time complexity of some problems related to the decentralized control with communication. The specification is given by a set ϕ of *responsiveness* properties over an alphabet Σ . A responsiveness property is a formula of the form $a \rightsquigarrow b$ (where $a, b \in \Sigma$) and it is satisfied by a (finite or infinite) string ℓ over Σ if an action b eventually occurs after every action a in ℓ . The problem under consideration consists in deciding if there exists a decentralized controller \mathcal{C}_d such that the *maximal* language L_{max} of the controlled system $\mathcal{T}/\mathcal{C}_d$ satisfies ϕ . This language L_{max} is defined by all infinite strings of the controlled system and all finite strings that cannot be extended by an action of Σ (i.e., the system is in a deadlocking state). The author proves that the problem is undecidable when there is no communication or when the communication delays are unbounded. He conjectures that the problem is decidable when the communication delays are bounded. In [XK09], Kumar and Xu propose a distributed algorithm which computes an estimate of the current state of a system. More precisely, local sites maintain and update local state estimates from their local observation of the system and information received from the other sites. This information is exchanged through reliable unbounded FIFO channels. The control of distributed systems also received attention in the domain of Petri nets, but these works generally use a synchronization mechanism to coordinate the control decisions of the local controllers. In [HB91], Haoxun and Baosheng define an algorithm for the state avoidance control problem. Their

method consists in augmenting the system with *coordination places*, that act as semaphores and are used by the local controllers to coordinate their control decisions. In [Dar05], Darondeau synthesizes distributed controllers by using Petri nets during the computations. More precisely, he states a sufficient condition which allows us to decide if a controller can be implemented by a particular class of Petri nets; this class is a strict subclass of finite automata. Then, he imposes a constraint on the produced Petri net, which ensures that it can be translated into several communicating automata $\mathcal{C}_1, \dots, \mathcal{C}_n$. These controllers can communicate through bounded channels, for which no assumption is made regarding the order and the delay of delivery of the messages. Finally, they are used to control the distributed system which is also composed of n communicating automata $\mathcal{T}_1, \dots, \mathcal{T}_n$.

The remainder of this chapter is structured as follows. In section 6.1, we present an overview of our control method. Next, in section 6.2, we define the formalism of *communicating finite state machines*, that we use to model distributed systems. In section 6.3, we formally define the control mechanisms and the state avoidance control problem that we want to solve. Then, in section 6.4, we provide an algorithm, that allows us to compute an estimate of the current state of a distributed system. In section 6.5, we define a control algorithm, based on this state estimate algorithm, for our state avoidance control problem and we explain how we can ensure the termination of this control algorithm by using abstract interpretations techniques. Finally, in section 6.6, we prove some properties presented in section 6.4.

6.1 Overview of the Method

We are interested in the distributed supervisory control of infinite state systems, consisting of *asynchronous networks of subsystems* where communication is possible through *reliable unbounded FIFO channels*. We associate a local controller with each subsystem. Each local controller can observe the last action fired by its subsystem¹ and disable some of its actions. Our goal is that the global system under the control of these local controllers satisfies a *global safety property*. In order to achieve a more permissive control, the controllers have a *memory* and can asynchronously communicate through the existing FIFO channels.

¹A controller observes the actions fired by its subsystem in order to have a better knowledge about this system. Note that since we consider, in the sequel, deterministic systems, the controller can infer the current location of its subsystem.

In this section, we present, with an example, the *communicating finite state machines*, which model the distributed systems to be controlled, as well as the kind of properties we want to ensure and the main idea of our method.

Introductory Example. We illustrate our method on a faulty communication protocol (see Figure 6.1 for the structure of the system). In this example, the subsystem \mathcal{T}_2 can send two kinds of messages to the subsystem \mathcal{T}_1 : a and b . The subsystem \mathcal{T}_1 can receive those messages at any time. However, an error occurs when the subsystem \mathcal{T}_1 receives the message a in the location A_0 . The protocol depicted in Figure 6.1 tries to avoid this error by using an additional subsystem \mathcal{T}_3 and synchronization messages c and d . The subsystem \mathcal{T}_1 sends the message c to \mathcal{T}_2 to start the communications. \mathcal{T}_1 can then read the messages a and b sent by \mathcal{T}_2 . When \mathcal{T}_2 has sent all messages a and b , it sends a message d to \mathcal{T}_1 , which passes through the subsystem \mathcal{T}_3 . After the reception of this message d , \mathcal{T}_1 also sends a message d to \mathcal{T}_2 to inform it that it has read this message. There are four FIFO channels: (i) channel 1: $\mathcal{T}_1 \rightarrow \mathcal{T}_2$, (ii) channel 2: $\mathcal{T}_2 \rightarrow \mathcal{T}_1$, (iii) channel 3: $\mathcal{T}_2 \rightarrow \mathcal{T}_3$, and (iv) channel 4: $\mathcal{T}_3 \rightarrow \mathcal{T}_1$. The action of sending the message c into the channel 1 is denoted by $1!c$, and the action of receiving the message c from the channel 1 is denoted by $1?c$. However, this protocol does not ensure that the subsystem \mathcal{T}_1 does not receive the message a in the location A_0 . Indeed, a possible counter-example is given by the following sequence of actions that starts in the initial location $\langle A_0, B_0, D_0 \rangle$ with all channels empty:

- The subsystem \mathcal{T}_1 sends the message c : $A_0 \xrightarrow{1!c} A_1$
- The subsystem \mathcal{T}_2 receives the message c : $B_0 \xrightarrow{1?c} B_1$
- The subsystem \mathcal{T}_2 sends the message a : $B_1 \xrightarrow{2!a} B_2$
- The subsystem \mathcal{T}_2 sends the message d : $B_2 \xrightarrow{3!d} B_3$
- The subsystem \mathcal{T}_3 receives the message d : $D_0 \xrightarrow{3?d} D_1$
- The subsystem \mathcal{T}_3 sends the message d : $D_1 \xrightarrow{4!d} D_0$
- The subsystem \mathcal{T}_1 receives the message d : $A_1 \xrightarrow{4?d} A_2$
- The subsystem \mathcal{T}_1 sends the message d : $A_2 \xrightarrow{1!d} A_0$
- The subsystem \mathcal{T}_1 receives the message a in the location A_0 : $A_0 \xrightarrow{2?a} A_{error}$

We would like to add three controllers $\mathcal{C}_1, \mathcal{C}_2$ and \mathcal{C}_3 to ensure that the subsystem \mathcal{T}_1 cannot receive the message a in the location A_0 . The controller \mathcal{C}_i ($\forall i \in [1, 3]$) controls the subsystem \mathcal{T}_i .

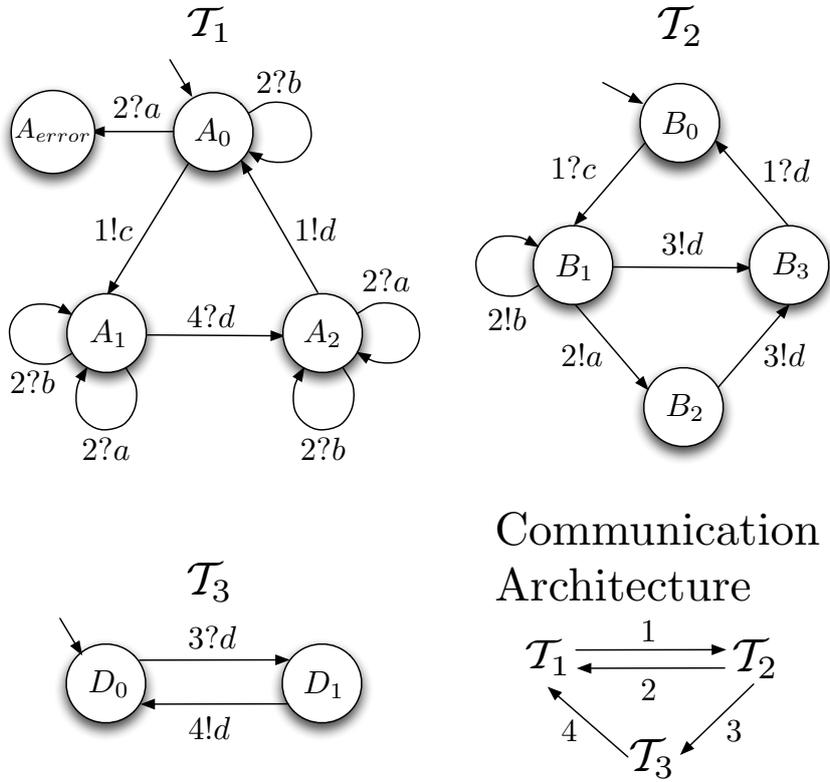


Figure 6.1 - Introductory example

Control Problem. We want to disable some actions to ensure that this error state cannot be reached. We assume that inputs are uncontrollable and outputs are controllable (i.e., they may be disabled by the local controllers). Thus, there is a trivial way to ensure the given property: \mathcal{C}_1 disables the action $A_0 \xrightarrow{1!c} A_1$. But if \mathcal{C}_1 disables this action, the system cannot perform any action and stays in its initial state. There are valid solutions that are more permissive than this trivial solution and our aim is to automatically compute one of them.

In our example, the most permissive solution is defined as follows: \mathcal{C}_1 always allows the action $A_0 \xrightarrow{1!c} A_1$, and disables the action $A_2 \xrightarrow{1!d} A_0$ when there is a message a in the channel 2. However, the local controllers do not observe the content of the FIFO channels. Therefore, to obtain *good* solutions, the communication between the local controllers must provide enough information to have a good knowledge of the content of the FIFO channels. We now present an overview of our method that automatically computes the local controllers and the kind of information they share.

Computation of the Set of Forbidden Global States. A state of the global system is given by a tuple $\langle x_1, x_2, x_3, w_1, w_2, w_3, w_4 \rangle$ where x_i ($\forall i \in [1, 3]$) gives the current location of the subsystem \mathcal{T}_i and w_i (for $i \in [1, 4]$) gives the content of the queue i . We want to ensure that the system does not reach the set of global states $Bad = \{ \langle x_1, x_2, x_3, M^*, M^*, M^*, M^* \rangle \mid x_1 = A_{error} \}$, where $M = \{a, b, c, d\}$ is the set of messages of the system. Bad represents the set of states in which the first subsystem has read a message a in the location A_0 . Since the inputs are uncontrollable, we must first compute the set $I(Bad)$ of global states that lead to Bad by a sequence of inputs. In our example, $I(Bad) = Bad \cup \{ \langle x_1, x_2, x_3, M^*, b^*.a.M^*, M^*, M^* \rangle \mid x_1 = A_0 \}$, since an arbitrary large number of b can be read in the location A_0 before reading a message a .

Since the coreachability problem is generally undecidable², we cannot exactly compute $I(Bad)$. We thus compute an *overapproximation* of $I(Bad)$ by using *abstract interpretation* techniques. The same problem arises when we have to compute $Reachable(X)$ for any set of states X and we also overcome this obstacle by using abstract interpretation techniques.

Control Decisions Based on Partial Observation. When $I(Bad)$ is computed, there is a simple way to prevent the system from reaching Bad :

²The communicating finite state machine model is Turing-complete. Thus we cannot decide when a given global state is reachable or coreachable from another one.

we forbid all controllable transitions that lead to $I(Bad)$. However, each local controller \mathcal{C}_i has only a partial observation of the global system, since it cannot observe the content of the queues and the control states of the other subsystems. If the control decisions of \mathcal{C}_i are only based on the observation of \mathcal{T}_i , this controller disables any action that *may* lead to $\text{Proj}_{\mathcal{T}_i}(I(Bad))$, where $\text{Proj}_{\mathcal{T}_i}$ is the projection on the locations of the subsystem \mathcal{T}_i . In this example, the local controller \mathcal{C}_1 always disables the action $A_2 \xrightarrow{1!d} A_0$ since it does not know whether there is a message a in the queue 2. To obtain more permissive solutions, we must consider control decisions based on a more accurate *knowledge* of the current global state of the system.

State Estimates and Communication Between Controllers. We consider local controllers \mathcal{C}_i that keep their own estimate of the current global state of the system and use this information to define their control policy. The estimate of the controller \mathcal{C}_i is updated from its observation of the local subsystem \mathcal{T}_i . Moreover, the local controllers asynchronously exchange their state estimates; this allows them to refine their own state estimate from the estimates of the other controllers. In our model, we consider controllers that communicate with each other by adding their state estimate to the messages normally exchanged by the subsystems.

In our example, if the subsystem \mathcal{T}_1 has not read the message a in the queue 2, then the controller \mathcal{C}_1 of \mathcal{T}_1 cannot know by itself if there is a message a in this queue. However, when the subsystem \mathcal{T}_2 sends the message d to the subsystem \mathcal{T}_3 , its controller \mathcal{C}_2 knows whether a message a has been sent. \mathcal{C}_3 can then forward this information to \mathcal{C}_1 . So, when the subsystem \mathcal{T}_1 is in the location A_2 , its controller \mathcal{C}_1 knows whether there is a message a in the queue 2 and it can then define the right control policy i.e., it disables the transition $A_2 \xrightarrow{1!d} A_0$ if and only if there is a message a in the queue 2.

Effective Algorithm. The control problem, that we want to solve, is generally *undecidable*. We then use abstract interpretation techniques to ensure, at the price of some overapproximations, that the computations of our control algorithm always terminate. In our case, the abstractions consist in substituting the queue contents by *regular languages*.

6.2 Model of the System

Communicating Finite State Machines. A distributed system is generally composed of several subsystems acting in parallel. In our framework, we model each subsystem of a distributed system by a *communicating finite state machine* [BZ83b]. A communicating finite state machine has (potentially) unbounded FIFO channels, that is can use to communicate with other subsystems. This formalism is defined as follows:

Definition 6.1 (Communicating Finite State Machines)

A *communicating finite state machine* (or *CFSM* for short) \mathcal{T} is defined by a 6-tuple $\langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, where :

- L is a finite set of locations
- $\ell_0 \in L$ is the initial location
- N is a set of queues that \mathcal{T} can use
- M is a finite set of messages
- $\Sigma \subseteq N \times \{!, ?\} \times M$ is a finite set of actions³, which are:
 - either an output $i!m$: the message $m \in M$ is written on the queue $i \in N$
 - or an input $i?m$: the message $m \in M$ is read on the queue $i \in N$
- $\Delta \subseteq L \times \Sigma \times L$ is a finite set of transitions. A transition $\delta \in \Delta$ is either an input $\langle \ell, i?m, \ell' \rangle$ or an output $\langle \ell, i!m, \ell' \rangle$.

A transition $\langle \ell, i!m, \ell' \rangle$ indicates that, when the system moves from the location ℓ to the location ℓ' , a message m must be added at the end of the queue i . This transition is called an *output transition*. A transition $\langle \ell, i?m, \ell' \rangle$ indicates that, when the system moves from the location ℓ to the location ℓ' , a message m must be present at the beginning of the queue i and must be removed from this queue. This transition is called an *input transition*. To remain concise, we do not consider internal actions. But the results presented in this chapter can easily be extended to the case where the CFSM can emit internal actions.

³The set Σ is redundant, but we define it, because it will often be used in this chapter.

In the sequel, the occurrence of a transition is called an *event*. One can note that two different events can correspond to a same transition. For example, two occurrences of the transition $\langle A_0, 2?b, A_0 \rangle$ of the system \mathcal{T}_1 depicted in Figure 6.1 give two different events e and e' .

Example 6.1

We consider our running example: the three CFSM depicted in Figure 6.1 which model the three subsystems \mathcal{T}_1 , \mathcal{T}_2 and \mathcal{T}_3 . The CFSM $\langle L_1, \ell_{0,1}, N_1, M_1, \Sigma_1, \Delta_1 \rangle$ modeling the subsystem \mathcal{T}_1 is defined as follows:

- $L_1 = \{A_0, A_1, A_2, A_{error}\}$
- $\ell_{0,1} = A_0$
- $N_1 = \{1, 2, 4\}$
- $M_1 = \{a, b, c, d\}$
- $\Delta_1 = \{\langle A_0, 1!c, A_1 \rangle, \langle A_1, 4?d, A_2 \rangle, \langle A_2, 1!d, A_0 \rangle, \langle A_0, 2?a, A_{error} \rangle, \langle A_0, 2?b, A_0 \rangle, \langle A_1, 2?a, A_1 \rangle, \langle A_1, 2?b, A_1 \rangle, \langle A_2, 2?a, A_2 \rangle, \langle A_2, 2?b, A_2 \rangle\}$.

The semantics of a CFSM can be defined as follows:

Definition 6.2 (Semantics of a CFSM)

The semantics of a CFSM $\mathcal{T} = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$ is given by an LTS $\llbracket \mathcal{T} \rrbracket = \langle X, x_0, \Sigma, \rightarrow \rangle$, where :

- $X \stackrel{\text{def}}{=} L \times (M^*)^{|N|}$ is the set of states
- $x_0 \stackrel{\text{def}}{=} \langle \ell_0, \epsilon, \dots, \epsilon \rangle$ is the initial state
- Σ is the set of actions (see Definition 6.1)
- \rightarrow is the transition relation defined as follows:

$$\frac{\delta = \langle \ell, i!m, \ell' \rangle \in \Delta \quad w'_i = w_i \cdot m}{\langle \ell, w_1, \dots, w_i, \dots, w_{|N|} \rangle \rightarrow \langle \ell', w_1, \dots, w'_i, \dots, w_{|N|} \rangle}$$

$$\frac{\delta = \langle \ell, i?m, \ell' \rangle \in \Delta \quad w_i = m \cdot w'_i}{\langle \ell, w_1, \dots, w_i, \dots, w_{|N|} \rangle \rightarrow \langle \ell', w_1, \dots, w'_i, \dots, w_{|N|} \rangle}$$

For a CFSM, the notion of determinism and the functions **Pre**, **Post**, **Reachable** and **Coreach** are defined by the same concepts on the corresponding

LTS (see section 1.2). In the sequel, we assume that \mathcal{T} is deterministic.

Asynchronous Product. A distributed system \mathcal{T} is generally composed of several subsystems \mathcal{T}_i ($\forall i \in [1, n]$) acting in parallel. In our case, this global system \mathcal{T} is defined by a CFSM resulting from the asynchronous product of the n subsystems \mathcal{T}_i , also modeled by CFSM.

Definition 6.3 (Asynchronous Product)

Given n CFSM $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M_i, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) with $L_i \cap L_j = \emptyset$ ($\forall i \neq j \in [1, n]$), their asynchronous product, denoted by $\mathcal{T}_1 || \dots || \mathcal{T}_n$, is defined by a CFSM $\mathcal{T} = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, where:

- $L \stackrel{\text{def}}{=} L_1 \times \dots \times L_n$
- $\ell_0 \stackrel{\text{def}}{=} \ell_{0,1} \times \dots \times \ell_{0,n}$
- $N \stackrel{\text{def}}{=} N_1 \cup \dots \cup N_n$. One can note that N_1, \dots, N_n are not necessarily disjoint; this allows the subsystems to communicate between them via common queues.
- $M \stackrel{\text{def}}{=} M_1 \cup \dots \cup M_n$
- $\Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \dots \cup \Sigma_n$
- Δ is defined as follows: for each $i \in [1, n]$, for each transition $\delta_i = \langle \ell_i, \sigma_i, \ell'_i \rangle \in \Delta_i$, and for each location $\ell_j \in L_j$ ($\forall j \neq i \in [1, n]$), we create a new transition $\langle \langle \ell_1, \dots, \ell_{i-1}, \ell_i, \ell_{i+1}, \dots, \ell_n \rangle, \sigma_i, \langle \ell_1, \dots, \ell_{i-1}, \ell'_i, \ell_{i+1}, \dots, \ell_n \rangle \rangle \stackrel{\text{def}}{\in} \Delta$.

Only the reachable part of $\mathcal{T}_1 || \dots || \mathcal{T}_n$ is considered⁴.

Communication Architecture. The system \mathcal{T} , that we want to control, is defined by the asynchronous product of n CFSM \mathcal{T}_i ($\forall i \in [1, n]$) acting in parallel and exchanging information through FIFO channels. In the sequel, to simplify the notations, we suppose that the subsystems \mathcal{T}_i use the same set M of messages. We also assume that the communication architecture is *point-to-point*, meaning that any subsystem \mathcal{T}_i can send messages to any subsystem \mathcal{T}_j (with $\mathcal{T}_j \neq \mathcal{T}_i$) through a queue $Q_{i,j}$. Thus, only \mathcal{T}_i can write on $Q_{i,j}$ and only \mathcal{T}_j can read on

⁴Note that the operator $||$ can be seen as a commutative and associative binary operator.

this queue⁵. With this architecture, the set N_i of \mathcal{T}_i ($\forall i \in [1, n]$) is defined by $N_i = \{Q_{i,j}, Q_{j,i} \mid (j \neq i) \wedge (1 \leq j \leq n)\}$ and the set of actions Σ_i of \mathcal{T}_i is disjoint from the set of actions Σ_j of \mathcal{T}_j ($\forall j \neq i \in [1, n]$). Moreover, we suppose that the message transfers between the subsystems are reliable and may suffer from arbitrary non-zero delays, and that a *global clock* or *perfectly synchronized local clocks* are not available.

Notations and Definitions. Given n CFSM $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined above, and a CFSM $\mathcal{T} = \mathcal{T}_1 \parallel \dots \parallel \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, we define the following notations:

- The projection $P_i : \Sigma^* \rightarrow \Sigma_i^*$ is defined as in (1.1).
- Let $\sigma \in \Sigma$ be an action of \mathcal{T} , $\text{Trans}(\sigma) \stackrel{\text{def}}{=} \{\langle \ell, \sigma_1, \ell' \rangle \in \Delta \mid \sigma_1 = \sigma\}$ gives the set of transitions of \mathcal{T} labeled by the action σ .
- Let $\bar{\sigma} = \sigma_1.\sigma_2 \dots \sigma_n$ be a sequence of actions and $\vec{v}, \vec{v}' \in L \times (M^*)^{|N|}$ be two states of \mathcal{T} , $\Delta(\vec{v}, \bar{\sigma})$ gives the state which is reachable from \vec{v} by executing the sequence $\bar{\sigma}$. In other words, $\Delta(\vec{v}, \bar{\sigma}) = \vec{v}'$ if and only if $\exists \vec{v}_0, \vec{v}_1, \dots, \vec{v}_n \in L \times (M^*)^{|N|}$ such that (i) $\vec{v}_0 = \vec{v}$, (ii) $\vec{v}_n = \vec{v}'$, and (iii) $\forall i \in [1, n] : \vec{v}_i = \text{Post}_{\sigma_i}^{\mathcal{T}}(\vec{v}_{i-1})$. Since \mathcal{T}_i is deterministic, only one state is reachable from the state \vec{v}_i by firing the action σ_i .
- Let e be an event occurring in \mathcal{T} and δ_e be the transition corresponding to this event, $\vec{v} \xrightarrow{e} \vec{v}'$ means that the state \vec{v}' can be reached from \vec{v} by firing the transition δ_e .
- Let $\delta_i = \langle \ell_i, \sigma_i, \ell'_i \rangle \in \Delta_i$ be a transition of the subsystem \mathcal{T}_i , $\text{global}(\delta_i) \stackrel{\text{def}}{=} \{\langle \langle \ell_1, \dots, \ell_{i-1}, \ell_i, \ell_{i+1}, \dots, \ell_n \rangle, \sigma_i, \langle \ell_1, \dots, \ell_{i-1}, \ell'_i, \ell_{i+1}, \dots, \ell_n \rangle \rangle \in \Delta \mid \forall j \neq i \in [1, n] : \ell_j \in L_j\}$ gives the set of transitions of Δ that are built from δ_i in the asynchronous product $\mathcal{T}_1 \parallel \dots \parallel \mathcal{T}_n$ (see the construction of Δ in Definition 6.3). We naturally extend this definition to sets of transitions: let $D \subseteq \Delta_i$ be a set of transitions, $\text{global}(D) \stackrel{\text{def}}{=} \bigcup_{\delta_i \in D} \text{global}(\delta_i)$.
- Let $B \subseteq L \times (M^*)^{|N|}$ be a subset of states of \mathcal{T} , and $\delta_i \in \Delta_i$ be a transition of the subsystem \mathcal{T}_i , by abuse of notation, we write $\Delta \setminus \Delta_i$ instead of $\Delta \setminus \text{global}(\Delta_i)$ and $\text{Post}_{\delta_i}^{\mathcal{T}}(B)$ (resp. $\text{Pre}_{\delta_i}^{\mathcal{T}}(B)$) instead of $\text{Post}_{\text{global}(\delta_i)}^{\mathcal{T}}(B)$ (resp. $\text{Pre}_{\text{global}(\delta_i)}^{\mathcal{T}}(B)$).

⁵When \mathcal{T}_i writes a message m on the queue $Q_{i,j}$ (i.e., it performs the action $Q_{i,j}!m$), \mathcal{T}_j can read this message by performing the action $Q_{i,j}?m$.

Moreover, we define the concept of *execution* as follows:

Definition 6.4 (Execution)

Given a CFSM $\mathcal{T} = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, m events e_i ($\forall i \in [1, m]$), $m + 1$ states \vec{v}_i ($\forall i \in [0, m]$), and the transition δ_{e_i} ($\forall i \in [1, m]$) corresponding to the event e_i , then an execution of \mathcal{T} is a sequence $s = \vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{v}_m$ where $\vec{v}_0 = \langle \ell_0, \epsilon, \dots, \epsilon \rangle$ is the initial state of \mathcal{T} . In other words, for every $i \in [1, m]$, the state \vec{v}_i is reachable from the state \vec{v}_{i-1} by the transition δ_{e_i} .

6.3 Framework and State Avoidance Control Problem

In this section, we define the framework, that we use, and the problem that we want to solve. More precisely, in section 6.3.1, we define the control mechanisms. Next, in section 6.3.2, we formally define the concepts of controller and of controlled execution. Finally, in section 6.3.3, we define the distributed state avoidance control problem.

6.3.1 Means of Control

The system to be controlled is composed of n subsystems \mathcal{T}_i ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2. Classically, we want to associate a local controller \mathcal{C}_i ($\forall i \in [1, n]$) with each subsystem \mathcal{T}_i in order to satisfy some control requirements. We suppose that the controllers can communicate with each other by adding some information (e.g., their current state, their state estimates, ...) to the messages exchanged by the subsystems (see Figure 6.2, where \mathcal{C}_1 and \mathcal{C}_2 use the queues $Q_{1,2}$ and $Q_{2,1}$ of \mathcal{T}_1 and \mathcal{T}_2 to exchange information⁶).

Following the Ramadge & Wonham's theory [RW89], \mathcal{C}_i ($\forall i \in [1, n]$) interacts with \mathcal{T}_i in a feedback manner: the controller \mathcal{C}_i observes the last action⁷ fired by \mathcal{T}_i and computes, from this observation and the information received from the other controllers (this information is actually state estimates), a set of actions that the subsystem \mathcal{T}_i cannot fire in order to ensure the desired properties on the global system. Moreover, we suppose that the local controller \mathcal{C}_i synchronously communicates with \mathcal{T}_i . This assumption is realistic, since they generally reside in the same site.

⁶We duplicate the queues in Figure 6.2 to emphasize the fact that the controllers can exchange information.

⁷ \mathcal{C}_i observes the actions fired by \mathcal{T}_i in order to have a better knowledge about \mathcal{T}_i .

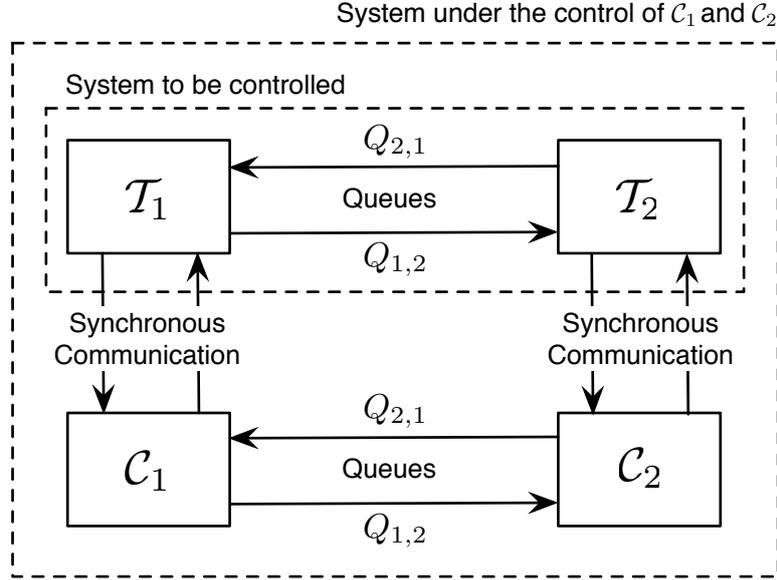


Figure 6.2 - Control architecture of a distributed system composed of two sub-systems \mathcal{T}_1 and \mathcal{T}_2 .

Moreover, as usual, the set of actions Σ_i of \mathcal{T}_i is partitioned into the set of controllable actions $\Sigma_{i,c}$, that can be forbidden by \mathcal{C}_i , and the set of uncontrollable actions $\Sigma_{i,uc}$, that cannot be forbidden by \mathcal{C}_i . The subsets $\Sigma_{1,c}, \dots, \Sigma_{n,c}$ are disjoint, because $\Sigma_i \cap \Sigma_j = \emptyset$ ($\forall i \neq j \in [1, n]$). Without loss of generality, we assume that:

- inputs are uncontrollable, and
- outputs are controllable.

Indeed, in a reactive system, the inputs cannot generally be controlled, whereas the outputs can be controlled.

The set of actions, that can be controlled by at least one controller, is denoted by Σ_c and is defined by $\Sigma_c \stackrel{\text{def}}{=} \bigcup_{i=1}^n \Sigma_{i,c}$ and the set of actions, that cannot be controlled, is denoted by Σ_{uc} and is defined by $\Sigma_{uc} \stackrel{\text{def}}{=} \Sigma \setminus \Sigma_c = \bigcup_{i=1}^n \Sigma_{i,uc}$. That also induces a partition of the set of transitions Δ_i into the set $\Delta_{i,c}$ and the set $\Delta_{i,uc}$: the transition $\langle \ell, \sigma, \ell' \rangle \in \Delta_{i,c}$, if the action $\sigma \in \Sigma_{i,c}$, and the transition $\langle \ell, \sigma, \ell' \rangle \in \Delta_{i,uc}$, if the action $\sigma \in \Sigma_{i,uc}$. The set of transitions Δ is similarly partitioned into the set Δ_c and the set Δ_{uc} .

Partial Observation. In the architecture we propose, the local controllers have only a partial observation of the global system. It is due to the fact that a local controller \mathcal{C}_i cannot observe the internal state of the subsystems \mathcal{T}_j ($\forall j \neq i \in [1, n]$) and the content of the FIFO channels.

6.3.2 Distributed Controller and Controlled Execution

The local controllers aim at restricting the behavior of the subsystems to ensure the desired properties. We formally define the local controllers \mathcal{C}_i ($\forall i \in [1, n]$) as follows:

Definition 6.5 (Local Controller)

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, and a CFM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, a *local controller* \mathcal{C}_i , which interacts with \mathcal{T}_i in order to restrict the behavior of this subsystem, is a pair $\mathcal{C}_i \stackrel{\text{def}}{=} \langle \mathcal{S}_i, E_i \rangle$, where⁸:

- $\mathcal{S}_i : 2^{L \times (M^*)^{|N|}} \rightarrow 2^{\Sigma_c}$ is a supervisory function which defines, for each $P \in 2^{L \times (M^*)^{|N|}}$, a set $\mathcal{S}_i(P)$ of controllable actions that \mathcal{T}_i cannot execute when P is the estimate of the current state of \mathcal{T} computed by \mathcal{C}_i .
- $E_i \subseteq L \times (M^*)^{|N|}$ is a set of states, in which the system \mathcal{T} cannot begin its execution.

In our control algorithm, we shall use the method defined in section 6.4 to compute, during the execution of the system, the state estimate P of \mathcal{C}_i .

Based on Definition 6.5, a *distributed controller* is defined as follows:

Definition 6.6 (Distributed Controller)

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, and a CFM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, a *distributed controller* \mathcal{C}_{di} is defined by a tuple $\mathcal{C}_{\text{di}} \stackrel{\text{def}}{=} \langle \mathcal{C}_i \rangle_{i=1}^n$ where \mathcal{C}_i ($\forall i \in [1, n]$) is a local controller (see Definition 6.5).

⁸Since the system \mathcal{T} has only one initial state, we could define a local controller \mathcal{C}_i without the component E_i , but we keep it to remain coherent with the definitions of controller given in the previous chapters.

The concept of *controlled execution* characterizes the executions that can occur in a system \mathcal{T} under the control of a distributed controller \mathcal{C}_{di} .

Definition 6.7 (Controlled Execution)

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, a CFSM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, a distributed controller $\mathcal{C}_{\text{di}} = \langle \mathcal{C}_i \rangle_{i=1}^n$ (for each $i \in [1, n]$, $\mathcal{C}_i = \langle \mathcal{S}_i, E_i \rangle$), an execution $s = \vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{v}_m$ of \mathcal{T} , and the transition $\delta_{e_j} = \langle \ell, \sigma_{e_j}, \ell' \rangle$ ($\forall j \in [1, m]$) corresponding to the event e_j , then s is a *controlled execution* of the system \mathcal{T} under the control of \mathcal{C}_{di} if it satisfies the following conditions:

- the initial state $\vec{v}_0 = \langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle$ does not belong to E_i ($\forall i \in [1, n]$).
- for every $k \in [1, m]$, if $\delta_{e_k} \in \Delta_i$, then, when the system \mathcal{T} reaches the state \vec{v}_{k-1} , the controller \mathcal{C}_i has an estimate P of the current state of \mathcal{T} such that $\sigma_{e_k} \notin \mathcal{S}_i(P)$. In other words, \mathcal{C}_i allows the action σ_{e_k} when the system \mathcal{T} reaches \vec{v}_{k-1} .

We shall see later on, how such a state estimate P can be computed.

6.3.3 Definition of the Control Problem

We define the problem we are interested in as follows:

Problem 6.1 (Distributed State Avoidance Control Problem)

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, a CFSM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, and a set of forbidden states $Bad \subseteq L \times (M^*)^{|N|}$, the *distributed state avoidance control problem* (*distributed problem* for short) consists in synthesizing a distributed controller $\mathcal{C}_{\text{di}} = \langle \mathcal{C}_i \rangle_{i=1}^n = \langle \langle \mathcal{S}_i, E_i \rangle \rangle_{i=1}^n$ such that (i) the initial state $\vec{v}_0 = \langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle$ of \mathcal{T} does not belong to E_i ($\forall i \in [1, n]$), and (ii) each controlled execution $\vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{v}_m$ of the system \mathcal{T} under the control of \mathcal{C}_{di} does not reach Bad i.e, the state $\vec{v}_m \notin Bad$.

The first condition ensures that the distributed controller is not trivial and the second one ensures that Bad is not reachable.

Proposition 6.1

The distributed problem is undecidable.

Intuitively, this proposition holds, because the reachability is undecidable in the model of CFSM.

As explained in section 6.1 and as shown in Definition 6.7, the local controllers \mathcal{C}_i define their control policy from an estimate of the current state of the global system \mathcal{T} that they compute during the execution of this system. In the next section, we explain how they compute this state estimate.

6.4 State Estimates of Distributed Systems

In this section, we present our algorithm that computes estimates of the current state of a distributed system. This algorithm is used, in section 6.5, by our control algorithm which synthesizes distributed controllers for the distributed problem. But, before presenting this algorithm, we recall the concept of *vector clocks* [Mat89] that we use to compute state estimates.

6.4.1 Instrumentation

In a centralized system, it is quite easy to determine if an event precedes another one, since all events are emitted by the same process. A logical clock, which counts the number of events, can be used to time-stamp the events and the order of two events can then be determined by comparing the value of their respective time-stamp.

In a distributed system, events emitted by a same process are ordered, while events emitted by different processes are generally not. However, when the concurrent processes communicate, additional ordering information can be obtained. In this case, the communication scheme can be used to obtain a partial order on the events of the system. In practice, vectors of logical clocks, called *Vector clocks* [Lam78], can be used to time-stamp the events of a distributed system. The order of two events can then be determined by comparing the value of their respective vector clocks. However, these vector clocks can be incomparable. In this case, the exact order in which the events occur cannot be determined. Vector clocks are formally defined as follows:

Definition 6.8 (Vector Clocks)

Let $\langle X, \sqsubseteq \rangle$ be a partially ordered set, a *vector clock mapping* of width n is a function $V : X \rightarrow (\mathbb{N}_0)^n$ such that:

$$\forall x_1, x_2 \in X : (x_1 \sqsubseteq x_2) \Leftrightarrow (V(x_1) \leq V(x_2)) \quad (6.1)$$

In general, for a distributed system composed of n processes, the partial order on events is represented by a vector clock mapping of width n . The method for computing this vector clock mapping depends on the communication scheme of the distributed system. CFSM communicate via message passing and hence we present, below, an algorithm [Mat89] that computes vector clock mappings for this communication scheme. We also present some properties [Mat89] related to this concept of vector clocks.

Computation of a Vector Clock Mapping. In a distributed system, when the processes asynchronously communicate via message passing, the sending of a message always precedes its reception. This information can then be used to determine a partial order, called *causality (or happened-before) relation*, on the events of the system. This relation is formally defined as follows:

Definition 6.9 (Causality Relation)

Let E_i ($\forall i \in [1, n]$) be the set of events occurring in an execution of \mathcal{T}_i and $E = \bigcup_{i \in [1, n]} E_i$, the *causality (or happened-before) relation* $\prec_c \subseteq E \times E$ is the smallest transitive relation satisfying the following conditions:

- if $e_i \neq e_j \in E_i$ (i.e., these events occur in the same process \mathcal{T}_i) and if e_i comes before e_j in the execution, then $e_i \prec_c e_j$.
- if $e_i \in E_i$ is an output event and if $e_j \in E_j$ is the corresponding input event, then $e_i \prec_c e_j$.

One can note that \prec_c is irreflexive. When $e_i \prec_c e_j$, we say that e_j *causally depends* on e_i (or that e_i *happened-before* e_j). Notice that a total order on the events of a distributed system can be obtained when one supposes the existence of a global clock or perfectly synchronized local clocks. But, in our framework, we have not this assumption.

The vector clock mapping for this relation can be obtained by using the

Algorithm 5: $\text{MatternAlgorithm}(\mathcal{T}_i, V_i, e)$

input : A subsystem \mathcal{T}_i (with $i \in [1, n]$), the vector clock V_i of \mathcal{T}_i , and an event e occurring in \mathcal{T}_i .

output: The vector clock V_i after the occurrence of e .

```

1 begin
2    $V_i[i] \leftarrow V_i[i] + 1$ 
3   if  $e$  corresponds to an output transition sending the message  $m$  to  $\mathcal{T}_j$  then
4      $\lfloor$  send  $\langle m, V_i \rangle$  to  $\mathcal{T}_j$  instead of  $m$ 
5   else if  $e$  corresponds to an input transition reading the message  $\langle m, V_j \rangle$ 
6     coming from  $\mathcal{T}_j$  then
7     for  $k \leftarrow 1$  to  $n$  do
8        $\lfloor$   $V_i[k] \leftarrow \max(V_i[k], V_j[k])$ 
9   return ( $V_i$ )
10 end

```

Mattern's algorithm [Mat89] (see Algorithm 5). In this algorithm, each process \mathcal{T}_i ($\forall i \in [1, n]$) has a vector clock $V_i \in (\mathbb{N}_0)^n$ of width n . Each element $V_i[j]$ ($\forall j \in [1, n]$) is a counter which represents the knowledge of \mathcal{T}_i regarding \mathcal{T}_j and which can roughly be interpreted as follows: \mathcal{T}_i knows that \mathcal{T}_j has executed at least $V_i[j]$ events. More precisely, $V_i[j]$ gives the number of events of \mathcal{T}_j that happened-before the most recent event of \mathcal{T}_i . Initially, each component of the vector V_i ($\forall i \in [1, n]$) is set to 0. Next, when an event e occurs in \mathcal{T}_i , the vector clock V_i is updated as follows (see Algorithm 5): first, $V_i[i]$ is incremented (i.e., $V_i[i] \leftarrow V_i[i] + 1$) to indicate that a new event occurred in \mathcal{T}_i and next two cases are considered:

- if e consists in sending the message m to \mathcal{T}_j , the vector clock V_i is attached to m and this information is sent to \mathcal{T}_j (see lines 3 and 4).
- if e corresponds to the reception of the message m tagged with the vector clock V_j , then V_i is set to the component-wise maximum of V_i and V_j . This allows us to take into account the fact that any event, that precedes the sending of m , should also precede the event e (see lines 5, 6 and 7).

In the sequel, the vector clock V_i , computed after the occurrence of an event e in \mathcal{T}_i , is denoted by $V_i(e)$.

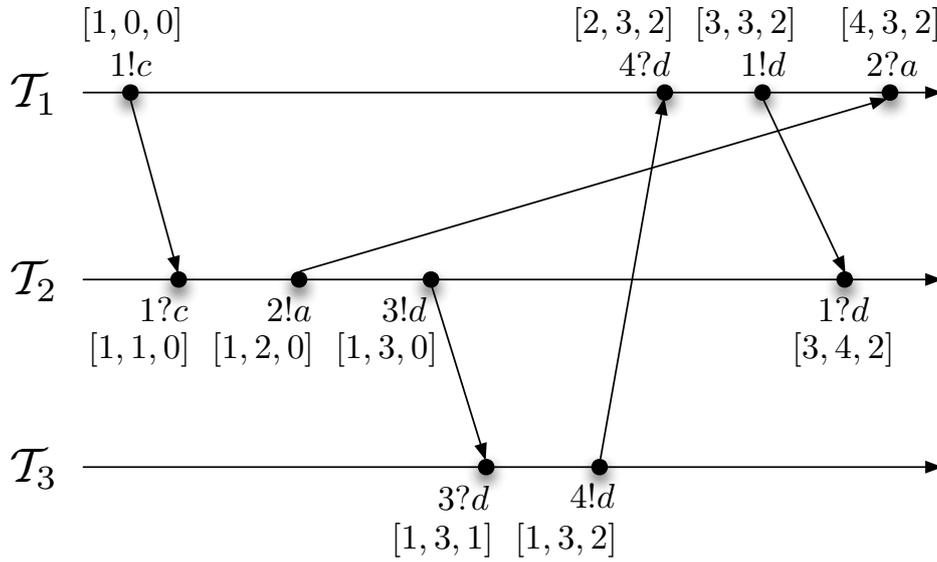


Figure 6.3 - Illustration of the concept of vector clocks.

Example 6.2

Let us illustrate this mechanism on our running example (see Figure 6.1). We consider the following sequence of actions (see Figure 6.3):

- The subsystem \mathcal{T}_1 performs the action $A_0 \xrightarrow{1!c} A_1$ and attaches its vector clock $V_1 = [1, 0, 0]$ to the message c .
- The subsystem \mathcal{T}_2 performs the action $B_0 \xrightarrow{1?c} B_1$ and updates its vector clock: $V_2 = [1, 1, 0]$.
- The subsystem \mathcal{T}_2 performs the action $B_1 \xrightarrow{2!a} B_2$ and attaches its vector clock $V_2 = [1, 2, 0]$ to the message a .
- The subsystem \mathcal{T}_2 performs the action $B_2 \xrightarrow{3!d} B_3$ and attaches its vector clock $V_2 = [1, 3, 0]$ to the message d .
- The subsystem \mathcal{T}_3 performs the action $D_0 \xrightarrow{3?d} D_1$ and updates its vector clock: $V_3 = [1, 3, 1]$.
- The subsystem \mathcal{T}_3 performs the action $D_1 \xrightarrow{4!d} D_0$ and attaches its vector clock $V_3 = [1, 3, 2]$ to the message d .

- The subsystem \mathcal{T}_1 performs the action $A_1 \xrightarrow{4?d} A_2$ and updates its vector clock: $V_1 = [2, 3, 2]$.
- The subsystem \mathcal{T}_1 performs the action $A_2 \xrightarrow{1!d} A_0$ and attaches its vector clock $V_1 = [3, 3, 2]$ to the message d .
- The subsystem \mathcal{T}_2 performs the action $B_3 \xrightarrow{1?d} B_0$ and updates its vector clock: $V_2 = [3, 4, 2]$.
- The subsystem \mathcal{T}_1 performs the action $A_0 \xrightarrow{2?a} A_{error}$ and updates its vector clock: $V_1 = [4, 3, 2]$. Note that the vector clock V_2 attached to this input is out of date, since $V_2 = [1, 2, 0]$ and $V_1 = [3, 3, 2]$ before this input.

Propositions. The following proposition proves the correctness of the vector clock mapping computed by the Mattern's algorithm for the relation \prec_c :

Proposition 6.2 ([Mat89])

Given n subsystems \mathcal{T}_i ($\forall i \in [1, n]$) and two events $e_1 \neq e_2$ occurring respectively in \mathcal{T}_i and \mathcal{T}_j (i can be equal to j), we have the following equivalence:

$$e_1 \prec_c e_2 \text{ if and only if } V_i(e_1) \leq V_j(e_2)$$

In the sequel, we will need the following proposition:

Proposition 6.3

Given n subsystems \mathcal{T}_i ($\forall i \in [1, n]$) and three events $e_i \neq e_j \neq e_k$ occurring respectively in \mathcal{T}_i , \mathcal{T}_j and \mathcal{T}_k (we can have $\mathcal{T}_i = \mathcal{T}_j$), if $e_k \not\prec_c e_j$ and $e_i \prec_c e_j$, then $e_k \not\prec_c e_i$.

Proof

We suppose that $e_k \prec_c e_i$. Since $e_k \not\prec_c e_j$, there exists a number $\ell \in [1, n]$ such that $V_k(e_k)[\ell] > V_j(e_j)[\ell]$. Moreover, $V_k(e_k)[\ell] > V_i(e_i)[\ell]$, because $V_i(e_i)[m] \leq V_j(e_j)[m]$ for each $m \in [1, n]$ (due to $e_i \prec_c e_j$). But it is a contradiction with $e_k \prec_c e_i$, because this relation implies that $V_k(e_k)[m] \leq V_i(e_i)[m]$ for each $m \in [1, n]$.

6.4.2 Computation of State Estimates

In our control algorithm, defined in section 6.5, each local controller must compute, during the execution of the distributed system, an estimate of the current state of the system to define its control policy. In this section, we explain how this state estimate is computed.

Let $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) be n CFSM communicating according to the architecture defined in section 6.2, and $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$ be a CFSM. Our state estimate algorithm computes, for each local controller \mathcal{C}_i and for each event occurring in the subsystem \mathcal{T}_i , a vector clock V_i and a state estimate CS_i that contains the current state of the global system \mathcal{T} and any future state that can be reached from this current state by firing actions that do not belong to \mathcal{T}_i . This computation obviously depends on the information that \mathcal{C}_i receives: as a reminder, \mathcal{C}_i observes the last action fired by \mathcal{T}_i (note that since \mathcal{T}_i is deterministic, \mathcal{C}_i can infer the fired transition) and can receive from the other controllers \mathcal{C}_j their state estimate CS_j and their vector clock V_j . Our state estimate algorithm proceeds as follows. When the subsystem \mathcal{T}_i performs an action, the following operations are performed:

- If it is an output message m , \mathcal{T}_i attaches the vector clock V_i and the state estimate CS_i of \mathcal{C}_i to this message. Next, \mathcal{C}_i receives the action fired by \mathcal{T}_i and infers the fired transition. It then uses this information to update its state estimate CS_i .
- If it is an input message m sent by \mathcal{T}_j , \mathcal{C}_i receives the action fired by \mathcal{T}_i and the information sent by \mathcal{T}_j i.e., the state estimate CS_j and the vector clock V_j of \mathcal{C}_j . It computes its new state estimate from these elements.

In both cases, the computation of the new state estimate CS_i depends on the computation of reachable states. In this section, we assume that we have an operator that can compute an *approximation* of the reachable states (as a reminder, the computation of the reachable states is *undecidable* in the model of CFSM). We will explain in section 6.5 how such an operator can be obtained.

6.4.2.1 State Estimate Algorithm

Our algorithm, that computes estimates of the current state of a distributed system, is composed of three subalgorithms:

- **initialEstimate**: computes, for each local controller, its initial state estimate. It is only used when the system starts its execution.

- **outputTransition**: computes online the new state estimate of \mathcal{C}_i after an output of \mathcal{T}_i .
- **inputTransition**: computes online the new state estimate of \mathcal{C}_i after an input of \mathcal{T}_i .

In the sequel, this state estimate algorithm is called SE-algorithm.

Algorithm 6: initialEstimate(\mathcal{T})

input : A CFSM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$ where, for each $i \in [1, n]$, $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$.

output: The initial state estimate CS_i of the local controller \mathcal{C}_i ($\forall i \in [1, n]$).

```

1 begin
2   for  $i \leftarrow 1$  to  $n$  do
3      $CS_i \leftarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle)$ 
4   return  $(CS_1, \dots, CS_n)$ 
5 end
```

initialEstimate Algorithm. The initial state of the system \mathcal{T} is $\langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle$. Before that the subsystem \mathcal{T}_i ($\forall i \in [1, n]$) executes its first action, the other subsystems \mathcal{T}_j ($\forall j \neq i \in [1, n]$) could perform inputs and outputs. Therefore, to take into account this possibility, the initial state estimate of \mathcal{T}_i is defined by $CS_i = \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle)$.

Algorithm 7: outputTransition($\mathcal{T}_i, V_i, CS_i, \delta$)

input : A subsystem $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ of the global system $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, the vector clock V_i of \mathcal{C}_i , the current state estimate CS_i of \mathcal{C}_i , and a transition

$\delta = \langle \ell_1, Q_{i,j}!m, \ell_2 \rangle \in \Delta_i$.

output: The state estimate CS_i after the output transition δ .

```

1 begin
2    $V_i[i] \leftarrow V_i[i] + 1$ 
3    $\mathcal{T}_i$  tags the message  $m$  with  $\langle CS_i, V_i, \delta \rangle$ 
4    $\mathcal{T}_i$  writes this tagged message on  $Q_{i,j}$ 
5    $CS_i \leftarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta}^{\mathcal{T}}(CS_i))$ 
6   return  $(CS_i)$ 
7 end
```

outputTransition Algorithm. Let CS_i be the current state estimate of \mathcal{C}_i . When \mathcal{T}_i ($\forall i \in [1, n]$) wants to execute a transition $\delta = \langle \ell_1, Q_{i,j}!m, \ell_2 \rangle \in \Delta_i$ corresponding to an output on the queue $Q_{i,j}$, the following instructions are computed to update the state estimate CS_i :

- the vector clock V_i of \mathcal{C}_i is updated as follows: $V_i[i] \leftarrow V_i[i] + 1$.
- \mathcal{T}_i tags the message m with $\langle CS_i, V_i, \delta \rangle$ and writes this information on the queue $Q_{i,j}$. The state estimate CS_i tagging m contains the set of states in which the system \mathcal{T} can be *before* the execution of δ . The additional information $\langle CS_i, V_i, \delta \rangle$ will be used by \mathcal{T}_j to refine its state estimate.
- the state estimate CS_i is updated as follows to contain the current state of the global system \mathcal{T} and any future state that can be reached from this current state by firing actions that do not belong to \mathcal{T}_i : $CS_i \leftarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta}^{\mathcal{T}}(CS_i))$. More precisely, $\text{Post}_{\delta}^{\mathcal{T}}(CS_i)$ gives the set of states in which the system \mathcal{T} can be after the execution of δ . However, after the execution of this transition, the subsystems \mathcal{T}_j ($\forall j \neq i \in [1, n]$) could read and write on their queues. Therefore, to take into account this possibility, we define the state estimate CS_i by $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta}^{\mathcal{T}}(CS_i))$.

inputTransition Algorithm. Let CS_i be the current state estimate of \mathcal{C}_i . When \mathcal{T}_i ($\forall i \in [1, n]$) executes a transition $\delta = \langle \ell_1, Q_{j,i}?m, \ell_2 \rangle \in \Delta_i$, corresponding to an input on the queue $Q_{j,i}$, it also reads the information $\langle CS_j, V_j, \delta' \rangle$ (where (i) CS_j is the state estimate of \mathcal{C}_j before the execution of δ' by \mathcal{T}_j , (ii) V_j is the vector clock of \mathcal{C}_j after the execution of δ' by \mathcal{T}_j , and (iii) $\delta' = \langle \ell'_1, Q_{j,i}!m, \ell'_2 \rangle \in \Delta_j$ is the output corresponding to δ) tagging m and the following operations are performed to update CS_i :

- we update the state estimate CS_j of \mathcal{C}_j (this update is denoted by $Temp_1$) by using the vector clocks to guess the possible behaviors of the system \mathcal{T} between the execution of the transition δ' and the execution of δ . For that, we consider three cases⁹:
 - if $V_j[i] = V_i[i]$, then $Temp_1 \leftarrow \text{Post}_{\delta}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta'}^{\mathcal{T}}(CS_j)))$. In this case, thanks to the vector clocks, we know that \mathcal{T}_i has executed no transition between the execution of δ' by \mathcal{T}_j and the execution of δ by \mathcal{T}_i . Only

⁹The first and second cases are not exclusive, but since the first case provides a better update of CS_j than the second one, we do not consider this latter case when the first one holds.

Algorithm 8: inputTransition($\mathcal{T}_i, V_i, CS_i, \delta$)

input : A subsystem $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ of the global system $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, the vector clock V_i of \mathcal{C}_i , the current state estimate CS_i of \mathcal{C}_i and a transition $\delta = \langle \ell_1, Q_{j,i}!m, \ell_2 \rangle \in \Delta_i$. The message m is tagged with the triple $\langle CS_j, V_j, \delta' \rangle$ where (i) CS_j is the state estimate of \mathcal{C}_j before the execution of δ' by \mathcal{T}_j , (ii) V_j is the vector clock of \mathcal{C}_j after the execution of δ' by \mathcal{T}_j , and (iii) $\delta' = \langle \ell'_1, Q_{j,i}!m, \ell'_2 \rangle \in \Delta_j$ is the output corresponding to δ .

output: The state estimate CS_i after the input transition δ .

```

1 begin
2   \ \ We consider three cases to update  $CS_j$ 
3   if  $V_j[i] = V_i[i]$  then
4      $Temp_1 \leftarrow Post_{\delta}^{\mathcal{T}}(Reachable_{\Delta \setminus \Delta_i}^{\mathcal{T}}(Post_{\delta'}^{\mathcal{T}}(CS_j)))$ 
5   else if  $V_j[j] > V_i[j]$  then
6      $Temp_1 \leftarrow Post_{\delta}^{\mathcal{T}}(Reachable_{\Delta \setminus \Delta_i}^{\mathcal{T}}(Reachable_{\Delta \setminus \Delta_j}^{\mathcal{T}}(Post_{\delta'}^{\mathcal{T}}(CS_j))))$ 
7   else
8      $Temp_1 \leftarrow Post_{\delta}^{\mathcal{T}}(Reachable_{\Delta}^{\mathcal{T}}(Post_{\delta'}^{\mathcal{T}}(CS_j)))$ 
9    $CS_i \leftarrow Post_{\delta}^{\mathcal{T}}(CS_i)$  \ \ We update  $CS_i$ 
10   $CS_i \leftarrow CS_i \cap Temp_1$ 
11  \ \ We intersect the updates of  $CS_i$  and  $CS_j$  (i.e.,  $Temp_1$ ) to obtain the
12  \ \ new state estimate  $CS_i$ 
13   $V_i[i] \leftarrow V_i[i] + 1$ 
14  for  $k \leftarrow 1$  to  $n$  do
15     $V_i[k] \leftarrow \max(V_i[k], V_j[k])$ 
16  return ( $CS_i$ )
17 end

```

transitions in $\Delta \setminus \Delta_i$ could thus have occurred during this period and we then update CS_j as follows. We compute $\text{Post}_{\delta'}^{\mathcal{T}}(CS_j)$ to take into account the execution of δ' by \mathcal{T}_j , $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta'}^{\mathcal{T}}(CS_j))$ to take into account the transitions that could occur between the execution of δ' and the execution of δ , and finally $\text{Post}_{\delta}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta'}^{\mathcal{T}}(CS_j)))$ to take into account the execution of δ by \mathcal{T}_i .

- else if $V_j[j] > V_i[j]$, then $Temp_1 \leftarrow \text{Post}_{\delta}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\text{Post}_{\delta'}^{\mathcal{T}}(CS_j))))$. Indeed, in this case, we can prove (see the proof of Proposition 6.4) that the transitions executed between the execution of δ' and the execution of δ can be reordered in order to execute the transitions of Δ_i before the ones of Δ_j . So, in this reordered sequence, we know that, after the execution of δ , only transitions in $\Delta \setminus \Delta_j$ could occur followed by transitions in $\Delta \setminus \Delta_i$. Intuitively, this reordering is possible, because there is no causal relation between the events of \mathcal{T}_i and the events of \mathcal{T}_j , that have occurred between δ' and δ .
- else $Temp_1 \leftarrow \text{Post}_{\delta}^{\mathcal{T}}(\text{Reachable}_{\Delta}^{\mathcal{T}}(\text{Post}_{\delta'}^{\mathcal{T}}(CS_j)))$. Indeed, in this case, the vector clocks do not allow us to deduce information regarding the behavior of the system \mathcal{T} between the execution of the transition δ' and the execution of δ . Therefore, to have a correct state estimate, we update CS_j by taking into account all possible behaviors of the system \mathcal{T} between the execution of δ' and the execution of δ . One can note that Lemma 6.1 (see below) shows that the set $Temp_1$ computed in the second case is better than the one computed in this third case.
- we update the state estimate CS_i to take into account the execution of the input transition δ : $CS_i \leftarrow \text{Post}_{\delta}^{\mathcal{T}}(CS_i)$.
- we have two different state estimates: $Temp_1$ and CS_i . Thus, we intersect them to obtain a better state estimate: $CS_i \leftarrow CS_i \cap Temp_1$.
- we update the vector clock V_i to take into account the execution of δ : $V_i[i] \leftarrow V_i[i] + 1$.
- we update the vector clock V_i by using the information given by V_j : $V_i[k] \leftarrow \max(V_i[k], V_j[k])$ for each $k \in [1, n]$.

Now, we prove that the set $Temp_1$ computed at the line 6 of Algorithm 8 is more accurate than the one computed at the line 8 of Algorithm 8.

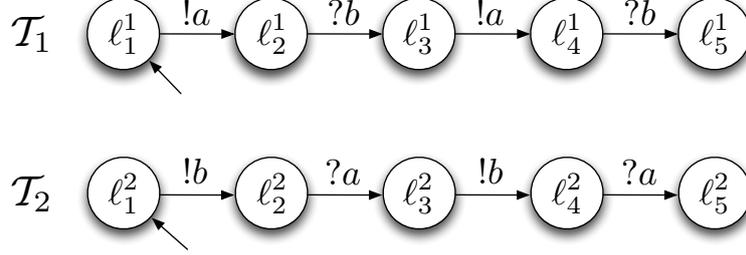


Figure 6.4 - System where $\text{Reachable}_{\Delta \setminus \Delta_1}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_2}^{\mathcal{T}}(B)) \subset \text{Reachable}_{\Delta}^{\mathcal{T}}(B)$.

Lemma 6.1

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, a CFSM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, $i \neq j \in [1, n]$, and a set $B \subseteq L \times (M^*)^{|N|}$, then $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(B)) \subseteq \text{Reachable}_{\Delta}^{\mathcal{T}}(B)$.

Proof

This proposition holds because $\Delta \setminus \Delta_i \subseteq \Delta$ and $\Delta \setminus \Delta_j \subseteq \Delta$.

Moreover, the following example shows that, in some cases, we can have $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(B)) \subset \text{Reachable}_{\Delta}^{\mathcal{T}}(B)$.

Example 6.3

Let $\mathcal{T}_1 = \langle L_1, \ell_{0,1}, N_1, M, \Sigma_1, \Delta_1 \rangle$ and $\mathcal{T}_2 = \langle L_2, \ell_{0,2}, N_2, M, \Sigma_2, \Delta_2 \rangle$ be two subsystems communicating according to the architecture defined in section 6.2 (see Figure 6.4), and $\mathcal{T} = \mathcal{T}_1 || \mathcal{T}_2 = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$. Given a state $\langle \ell_1, \ell_2, w_1, w_2 \rangle$ of \mathcal{T} , (i) ℓ_1 (resp. ℓ_2) gives the current location of \mathcal{T}_1 (resp. \mathcal{T}_2) and (ii) w_1 (resp. w_2) gives the content of the queue $Q_{1,2}$ (resp. $Q_{2,1}$). We define B as follows: $B = \{ \langle \ell_1^1, \ell_1^2, \epsilon, \epsilon \rangle \}$. The set $\text{Reachable}_{\Delta \setminus \Delta_1}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_2}^{\mathcal{T}}(B)) = \{ \langle \ell_1^1, \ell_1^2, \epsilon, \epsilon \rangle, \langle \ell_1^1, \ell_2^2, \epsilon, b \rangle, \langle \ell_2^1, \ell_1^2, a, \epsilon \rangle, \langle \ell_2^1, \ell_2^2, a, b \rangle, \langle \ell_2^1, \ell_3^2, \epsilon, b \rangle, \langle \ell_2^1, \ell_4^2, \epsilon, bb \rangle \}$. This set is strictly included in $\text{Reachable}_{\Delta}^{\mathcal{T}}(B) = \{ \langle \ell_1^1, \ell_1^2, \epsilon, \epsilon \rangle, \langle \ell_2^1, \ell_1^2, a, \epsilon \rangle, \langle \ell_1^1, \ell_2^2, \epsilon, b \rangle, \langle \ell_2^1, \ell_2^2, a, b \rangle, \langle \ell_3^1, \ell_2^2, a, \epsilon \rangle, \langle \ell_4^1, \ell_2^2, aa, \epsilon \rangle, \langle \ell_2^1, \ell_3^2, \epsilon, b \rangle, \langle \ell_2^1, \ell_4^2, \epsilon, bb \rangle, \langle \ell_3^1, \ell_3^2, \epsilon, \epsilon \rangle, \langle \ell_4^1, \ell_3^2, a, \epsilon \rangle, \langle \ell_3^1, \ell_4^2, \epsilon, b \rangle, \langle \ell_4^1, \ell_4^2, a, b \rangle, \langle \ell_5^1, \ell_4^2, a, \epsilon \rangle, \langle \ell_4^1, \ell_5^2, \epsilon, b \rangle, \langle \ell_5^1, \ell_5^2, \epsilon, \epsilon \rangle \}$.

6.4.2.2 Properties

Notations. Let $s = \vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{v}_m$ be an execution of the global system \mathcal{T} . When the subsystem \mathcal{T}_i executes an event e_k (with $k \in [1, m]$) of this sequence,

the state estimate, computed by \mathcal{C}_i at this step, is denoted by CS_i^t , where t is the number of events executed by \mathcal{T}_i in the subsequence $\vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_k} \vec{v}_k$. However, to be more concise in our proofs, we use an abuse of notation: when an event e_k is executed in the sequence s , the state estimate of *each* subsystem \mathcal{T}_i is denoted by CS_i^k (in this way, we do not need to introduce, for each subsystem, a parameter giving the number of events that it has executed). This state estimate is then defined in the following way:

- if e_k has not been executed by \mathcal{T}_i , then $CS_i^k \stackrel{\text{def}}{=} CS_i^{k-1}$,
- otherwise, the value of CS_i^k is computed by Algorithms 7 or 8 from the state estimate CS_i^{k-1} .

Definitions. We define the concepts of *complete* and *sound* algorithms as follows:

Definition 6.10 (Complete Algorithm)

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, a CFSSM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, and an execution $\vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{v}_m$ of \mathcal{T} (as a reminder, the initial state $\vec{v}_0 = \langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle$), a state estimate algorithm for \mathcal{T} is *complete* if and only if the current state \vec{v}_m belongs to the state estimate CS_i^m ($\forall i \in [1, n]$).

The completeness of an algorithm refers to the fact that the current state of the global system is always included in the state estimate.

Definition 6.11 (Sound Algorithm)

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, a CFSSM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, an execution $s = \vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{v}_m$ of \mathcal{T} , and the transition $\langle \ell_{e_k}, \sigma_{e_k}, \ell'_{e_k} \rangle$ ($\forall k \in [1, m]$) corresponding to the event e_k , a state estimate algorithm for \mathcal{T} is *sound* if and only if $\forall i \in [1, n] : CS_i^m \subseteq \bigcup_{\bar{\sigma} \in P_i^{-1}(P_i(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m}))} \Delta(\vec{v}_0, \bar{\sigma})$, where $\vec{v}_0 = \langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle$ is the initial state of \mathcal{T} .

When \mathcal{T}_i executes an action, \mathcal{C}_i receives the fired action. Therefore, the observation of the system \mathcal{T}_i allows the local controller \mathcal{C}_i to know that the system \mathcal{T} has executed a sequence of actions $\bar{\sigma} \in P_i^{-1}(P_i(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m}))$ to reach the state \vec{v}_m . The soundness of an algorithm refers to the fact that a state

is included in the state estimate of \mathcal{C}_i only if it can be reached by one of these sequences $\bar{\sigma}$.

Properties. As explained above, we assume that we can compute an approximation of the reachable states. In this part, we present the properties of our state estimate algorithm w.r.t. the kind of used approximations.

Proposition 6.4

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, and a CFMSM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, SE-algorithm is complete, if we compute an overapproximation of the reachable states.

The proof is given in section 6.6. If we compute an underapproximation of the reachable states, our state estimate algorithm is not complete and hence, in section 6.5, we define an effective algorithm for the distributed problem by computing overapproximations of the reachable states.

Proposition 6.5

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, and a CFMSM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, SE-algorithm is sound, if we compute an underapproximation of the reachable states.

The proof is given in section 6.6. If we compute an overapproximation of the reachable states, our state estimate algorithm is not sound.

Remark 6.1 (Improvements of the State Estimates)

Our state estimate algorithm can be improved by informing the subsystems of the reading of a message. More precisely, when a subsystem \mathcal{T}_j executes an input transition $\delta = \langle \ell_j, Q_{i,j}?m, \ell'_j \rangle$ (with $i \neq j \in [1, n]$), it sends an odd message tagged with this transition δ , the state estimate CS_j and the vector clock V_j on the queue $Q_{j,k}$ ($\forall k \neq j \in [1, n]$). When the subsystem \mathcal{T}_k reads this message, its local controller \mathcal{C}_k can refine its state estimate from this information. For that, it first updates its state estimate CS_k by taking into account the reading of m on the queue $Q_{i,j}$: $CS_k \leftarrow \text{Post}_\delta^{\mathcal{T}}(CS_k)$. Next, it updates CS_j by using the method based on vector clocks which is defined in

Algorithm 8. Finally, it intersects these two sets to obtain the new state estimate CS_k . It can be proved that the properties of soundness and completeness remain valid with this improvement.

6.5 Computation by Means of Abstract Interpretation of Distributed Controllers for the Distributed Problem

In this section, we first define a semi-algorithm for the distributed problem. Next, we explain how to extend it by using abstract interpretation techniques to obtain an effective algorithm.

6.5.1 Semi-algorithm for the Distributed Problem

Our algorithm, which synthesizes a distributed controller \mathcal{C}_{di} for the distributed problem, is composed of two parts:

1. **Offline part:** we compute, using a fixpoint equation, the set $I(Bad)$ of states of the global system \mathcal{T} that can lead to Bad by a sequence of uncontrollable transitions. Next, we compute, for each local controller \mathcal{C}_i , a control function \mathcal{F}_i which gives, for each action σ of \mathcal{T}_i , the set of states of \mathcal{T} that can lead to $I(Bad)$ by a transition labeled by σ . This information is used by \mathcal{C}_i , in the online part, to define its control policy. This part is computed *offline*.
2. **Online part:** During the execution of the system \mathcal{T} , each local controller \mathcal{C}_i uses SE-algorithm to obtain its own state estimate CS_i and computes, from this information and its control function \mathcal{F}_i (computed in the offline part), the actions to be forbidden. Roughly, \mathcal{C}_i forbids an action σ in a state \vec{v} of \mathcal{T} if \vec{v} belongs to CS_i and if \mathcal{F}_i asks to forbid σ in \vec{v} .

These two parts are formalized as follows.

Offline Part. The set $I(Bad)$ of states of \mathcal{T} leading uncontrollably to Bad is given by $\text{Coreach}_{\Delta_{uc}}^{\mathcal{T}}(Bad)$ which, as a reminder, is defined by $\text{Coreach}_{\Delta_{uc}}^{\mathcal{T}}(Bad) = \bigcup_{n \geq 0} (\text{Pre}_{\Delta_{uc}}^{\mathcal{T}})^n(Bad)$ (see Definition 1.9). This set of states cannot always be computed, because the coreachability is undecidable in the model of CFSM. So, we use abstract interpretation techniques to overapproximate it. However, to compute this overapproximation, we must characterize the set $I(Bad)$ by a

fixpoint equation. This equation is defined as follows over the complete lattice $\langle 2^{L \times (M^*)^{|N|}}, \subseteq, \cup, \cap, L \times (M^*)^{|N|}, \emptyset \rangle$:

$$I(Bad) \stackrel{\text{def}}{=} \text{lfp}^{\subseteq}(\lambda B. Bad \cup \text{Pre}_{\Delta_{uc}}^{\mathcal{T}}(B)) \quad (6.2)$$

Since the function $\lambda B. Bad \cup \text{Pre}_{\Delta_{uc}}^{\mathcal{T}}(B)$ is continuous [Cou81], the Knaster-Tarski and Kleene's theorems [Tar55, Mar97] ensure that the least fixpoint of this function actually exists and $I(Bad) = \text{Coreach}_{\Delta_{uc}}^{\mathcal{T}}(Bad)$.

Next, we define, for each local controller \mathcal{C}_i , the control function $\mathcal{F}_i : \Sigma_i \times 2^{L \times (M^*)^{|N|}} \rightarrow 2^{L \times (M^*)^{|N|}}$, which gives, for each action $\sigma \in \Sigma_i$ and set $B \subseteq L \times (M^*)^{|N|}$ of states to be forbidden, the set $\mathcal{F}_i(\sigma, B)$ of global states in which the action σ must be forbidden. This set corresponds, more precisely, to the greatest set \mathcal{O} of states of \mathcal{T} such that, for each state $\vec{v} \in \mathcal{O}$, there exists a transition labeled by σ which leads to B from \vec{v} :

$$\mathcal{F}_i(\sigma, B) \stackrel{\text{def}}{=} \begin{cases} \text{Pre}_{\text{Trans}(\sigma)}^{\mathcal{T}}(B) & \text{if } \sigma \in \Sigma_{i,c} \\ \emptyset & \text{otherwise} \end{cases} \quad (6.3)$$

We compute, for each transition $\sigma \in \Sigma_i$, the set $\mathcal{F}_i(\sigma, I(Bad))$ ($\forall i \in [1, n]$). This information will be used, during the execution of the system, by the local controller \mathcal{C}_i to compute the actions to be forbidden.

Online Part. During the execution of the system, when the subsystem \mathcal{T}_i ($\forall i \in [1, n]$) executes a transition $\delta = \langle \ell_i, \sigma, \ell'_i \rangle$, the local controller \mathcal{C}_i receives the following information:

- if $\sigma = Q_{i,j}!m$ (with $j \neq i \in [1, n]$) is an output, it receives σ .
- if $\sigma = Q_{j,i}?m$ (with $j \neq i \in [1, n]$) is an input, it receives σ and the triple $\langle CS_j, V_j, \delta' \rangle$ tagging m .

In both cases, since the system is deterministic and since the local controller \mathcal{C}_i knows that \mathcal{T}_i was in the location ℓ_i before triggering σ , this controller can infer the fired transition δ . \mathcal{C}_i then uses SE-algorithm with this information to update its state estimate CS_i and computes, from this state estimate, the set $\mathcal{S}_i(CS_i)$ of actions that \mathcal{T}_i cannot execute (the computation of \mathcal{S}_i is based on \mathcal{F}_i and is given below).

Now, we explain how we define our local controller \mathcal{C}_i ($\forall i \in [1, n]$), which gives us the function \mathcal{S}_i , and our distributed controller \mathcal{C}_{di} . The local controller \mathcal{C}_i ($\forall i \in [1, n]$) is formally defined as follows:

$$\mathcal{C}_i \stackrel{\text{def}}{=} \langle \mathcal{S}_i, E_i \rangle \quad (6.4)$$

where the elements \mathcal{S}_i and E_i are defined in the following way:

- the supervisory function \mathcal{S}_i is given, for each $P \in 2^{L \times (M^*)^{|N|}}$, by:

$$\mathcal{S}_i(P) = \{\sigma \in \Sigma_i \mid \mathcal{F}_i(\sigma, I(Bad)) \cap P \neq \emptyset\} \quad (6.5)$$

Thus, if P is the state estimate of \mathcal{C}_i , the supervisory function \mathcal{S}_i of \mathcal{C}_i forbids an action $\sigma \in \Sigma_i$ if and only if there exists a state $\vec{v} \in P$ in which the action σ must be forbidden in order to prevent the system \mathcal{T} from reaching $I(Bad)$ (i.e., $\exists \vec{v} \in P : \vec{v} \in \mathcal{F}_i(\sigma, I(Bad))$).

- the set $E_i \stackrel{\text{def}}{=} I(Bad)$. It prevents the system from beginning its execution in a state which can lead to Bad by a sequence of uncontrollable transitions.

Note that the sets E_i are identical for all $i \in [1, n]$.

Our distributed controller \mathcal{C}_{di} is then defined as follows:

$$\mathcal{C}_{\text{di}} \stackrel{\text{def}}{=} \langle \mathcal{C}_i \rangle_{i=1}^n \quad (6.6)$$

where the local controller \mathcal{C}_i ($\forall i \in [1, n]$) is defined as in (6.4).

The following proposition proves that this algorithm synthesizes correct distributed controllers for the distributed problem.

Proposition 6.6

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, a CFSM $\mathcal{T} = \mathcal{T}_1 \parallel \dots \parallel \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, and a set of forbidden states $Bad \subseteq L \times (M^*)^{|N|}$, the distributed controller $\mathcal{C}_{\text{di}} = \langle \mathcal{C}_i \rangle_{i=1}^n = \langle \langle \mathcal{S}_i, E_i \rangle \rangle_{i=1}^n$, defined in (6.6), solves the distributed problem when the initial state $\langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle$ of \mathcal{T} does not belong to E_i ($\forall i \in [1, n]$).

Proof

We prove by induction on the length m of the sequences of transitions (these sequences begin in an initial state) that $I(Bad)$ is not reachable in the system \mathcal{T} under the control of \mathcal{C}_{di} , which implies that Bad is not reachable, because $Bad \subseteq I(Bad)$:

- *Base case* ($m = 0$): Since $\langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle \notin E_i$ ($\forall i \in [1, n]$), the execution of the system \mathcal{T} under the control of \mathcal{C}_{di} starts in a state which does not belong to $I(Bad)$ ($E_i = I(Bad)$ for each $i \in [1, n]$).

• *Induction step:* we suppose that the proposition holds for the sequences of transitions of length less than or equal to m and we prove that this property remains true for the sequences of transitions of length $m+1$. By induction hypothesis, each state \vec{v}_1 reachable by a sequence of transitions of length m does not belong to $I(Bad)$ and we show that each transition $\delta \in \Delta$, which can lead to a state $\vec{v}_2 \in I(Bad)$ from this state \vec{v}_1 in the system \mathcal{T} , cannot be fired from \vec{v}_1 in the system \mathcal{T} under the control of \mathcal{C}_{di} . For that, we consider two cases and we suppose that δ is executed by \mathcal{T}_i and is labeled by σ :

- if δ is controllable, then the action σ is forbidden by \mathcal{C}_i in the state \vec{v}_1 and hence δ cannot be fired from \vec{v}_1 . Indeed, the state estimate CS_i^m of \mathcal{C}_i contains \vec{v}_1 , because SE-algorithm is complete. Moreover, we have that $\vec{v}_1 \in \mathcal{F}_i(\sigma, I(Bad))$, because $\vec{v}_1 \in \text{Pre}_\delta^{\mathcal{T}}(\vec{v}_2)$ and $\vec{v}_2 \in I(Bad)$. Therefore, $\sigma \in \mathcal{S}_i(CS_i^m)$ (by (6.5)), which implies that δ cannot be fired from \vec{v}_1 .
- if δ is uncontrollable, then $\vec{v}_2 \in I(Bad)$ (by (6.2)), which is impossible by hypothesis.

Hence, in the system \mathcal{T} under the control of \mathcal{C}_{di} , the forbidden state \vec{v}_2 cannot be reached from the state \vec{v}_1 by the transition δ .

Example 6.4

We consider the sequence of actions of our running example described in section 6.1. At the beginning of the execution of \mathcal{T} , the state estimates of the three subsystems are:

- $CS_1 = \{\langle A_0, B_0, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle\}$
- $CS_2 = \{\langle A_0, B_0, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_0, D_0, c, \epsilon, \epsilon, \epsilon \rangle\}$
- $CS_3 = \{\langle A_0, B_0, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_0, D_0, c, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_1, D_0, \epsilon, b^*, \epsilon, \epsilon \rangle, \langle A_1, B_2, D_0, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle, \langle A_1, B_3, D_0, \epsilon, b^*(a + \epsilon), d, \epsilon \rangle\}$

After the first action $A_0 \xrightarrow{1lc} A_1$, the state estimate of the controller \mathcal{C}_1 is not really precise, because a lot of things may happen without the controller \mathcal{C}_1 being informed: the estimate $CS_1 = \{\langle A_1, B_0, D_0, c, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_1, D_0, \epsilon, b^*, \epsilon, \epsilon \rangle,$

$\langle A_1, B_2, D_0, \epsilon, b^*a, \epsilon, \epsilon \rangle$, $\langle A_1, B_3, D_0, \epsilon, b^*(a + \epsilon), d, \epsilon \rangle$, $\langle A_1, B_3, D_1, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle$, $\langle A_1, B_3, D_0, \epsilon, b^*(a + \epsilon), \epsilon, d \rangle$. However, after the second action $B_0 \xrightarrow{1?c} B_1$, the controller \mathcal{C}_2 has an accurate state estimate: $CS_2 = \{\langle A_1, B_1, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle\}$. We skip a few steps and consider the state estimates before the sixth action $D_1 \xrightarrow{4!d} D_0$: CS_1 is still the same, because the subsystem \mathcal{T}_1 did not perform any action, $CS_3 = \{\langle A_1, B_3, D_1, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle\}$, and we do not give CS_2 , because \mathcal{T}_2 is no longer involved. When the subsystem \mathcal{T}_3 sends the message d to \mathcal{T}_1 , it tags it with CS_3 . Thus, the controller \mathcal{C}_1 knows, after receiving this message, that there may be a message a in the queue 2. It thus disables the action $A_2 \xrightarrow{1!d} A_0$, as long as this message a is not read, to prevent the system from reaching the forbidden states.

6.5.2 Effective Algorithm for the Distributed Problem

The algorithm described in the previous section requires the computation of reachability and coreachability operators, but they cannot always be computed for undecidability (or complexity) reasons. Again, we overcome this obstacle by using *abstract interpretation* techniques. The method to compute these abstractions is the same as the one used in the previous chapters for STS: we define a representation framework to formalize the abstractions and we perform the computations in an abstract lattice by using a widening operator to ensure the termination of our algorithm. The only difference is that we use a different abstract lattice and a different widening operator.

Since the CFSM model is Turing-powerful, the language which represents all possible contents of the FIFO channels may be recursively enumerable. We would like to abstract the contents of the queues by a simpler class of languages: the regular languages, which can be represented by finite automata. This issue was already discussed in [LGJJ06] and, in the sequel, we remind the main ideas of this abstraction.

Definition of the Elements of the Representation Framework. In order to clearly explain the abstractions, we assume that there is a single queue; we explain further how to do with several queues. With one queue, the concrete domain of the system $\mathcal{T} = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$ is defined by $2^{L \times M^*}$. Instead of considering unstructured sets of states of the domain $2^{L \times M^*}$, we prefer to associate sets of queue contents with the locations. For this, we can observe that $2^{L \times M^*}$ is isomorphic to $L \rightarrow 2^{M^*}$. Thus, a set of states $B \in 2^{L \times M^*}$ can be viewed

as a map $B : L \rightarrow 2^{M^*}$ that associates a language $B(\ell)$ with each location $\ell \in L$; in this case, $B(\ell)$ represents the possible contents of the queue in the location ℓ . In the sequel, we represent sets of states in this way. We substitute the concrete lattice $\langle L \rightarrow 2^{M^*}, \subseteq, \cup, \cap, L \times M^*, \emptyset \rangle$ by the abstract lattice $\langle L \rightarrow \text{Reg}(M), \subseteq, \cup, \cap, L \times M^*, \emptyset \rangle$, where $\text{Reg}(M)$ is the set of *regular languages* over the alphabet M . This substitution consists thus in abstracting, for each location, the possible contents of the queue by a regular language. Regular languages have a canonical representation given by finite automata and, in the abstract lattice, the regular language associated with a location is actually represented by its canonical automaton. So, each operation in the abstract lattice is performed on automata. For example, let B_1 and B_2 be two sets of $L \rightarrow \text{Reg}(M)$, then $B_1 \cap B_2$ is obtained by computing, for each location $\ell \in L$, the intersection of the automata representing $B_1(\ell)$ and $B_2(\ell)$.

Now, we define the widening operator [LGJJ06]. A widening operator on regular languages is sufficient, because, given two sets B_1 and B_2 of $L \rightarrow \text{Reg}(M)$, we can apply this operator to $B_1(\ell)$ and $B_2(\ell)$ for each $\ell \in L$. The widening operator, that we use, depends on the *k-bounded bisimulation relation* \equiv_k [LGJJ06] where $k > 0$ is a parameter which allows us to be more or less precise in the abstractions (increasing k improves the quality of the abstractions). Let $\mathcal{A} = \langle X, X_0, X_m, \Sigma, \Delta \rangle$ be an automaton, this equivalence relation is defined over \mathcal{A} as follows:

- if $k = 0$, the relation \equiv_0 is defined by four equivalence classes: (i) initial states, (ii) marked states, (iii) states that are both initial and marked, and (iv) all other states.
- if $k > 0$, the relation \equiv_k is recursively defined in the following way: $\forall x_1, x_2 \in X, x_1 \equiv_k x_2$ if and only if:
 - $x_1 \equiv_{k-1} x_2$,
 - $\forall a \in \Sigma, \forall x'_1 \in X$: if $x_1 \xrightarrow{a} x'_1$, then there exists a state $x'_2 \in X$ such that $x_2 \xrightarrow{a} x'_2$ and $x'_1 \equiv_{k-1} x'_2$,
 - $\forall a \in \Sigma, \forall x'_2 \in X$: if $x_2 \xrightarrow{a} x'_2$, then there exists a state $x'_1 \in X$ such that $x_1 \xrightarrow{a} x'_1$ and $x'_1 \equiv_{k-1} x'_2$.

Let \mathcal{L}_1 and \mathcal{L}_2 be two regular languages given by their canonical automaton $\mathcal{A}_{\mathcal{L}_1}$ and $\mathcal{A}_{\mathcal{L}_2}$, the widening operator ∇ applied to \mathcal{L}_1 and \mathcal{L}_2 (denoted by $\mathcal{L}_1 \nabla \mathcal{L}_2$) is defined as follows (as a reminder, in the abstract lattice, the computations are performed on the automata representing the considered regular languages):

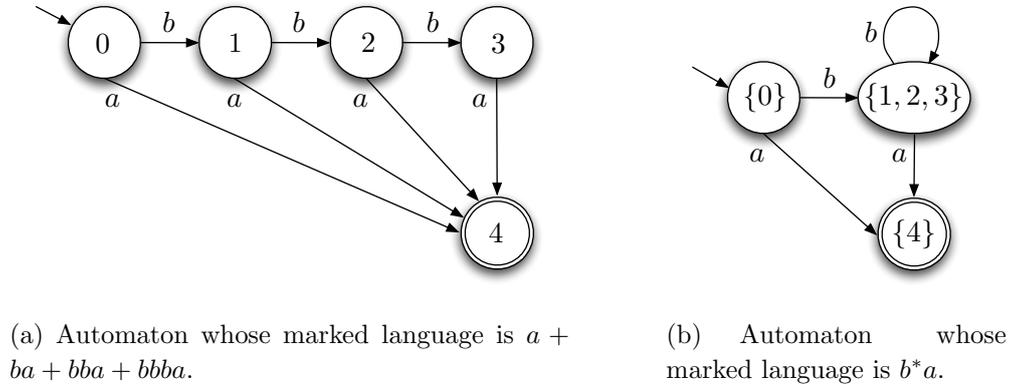


Figure 6.5 - Illustration of the k -bounded bisimulation relation \equiv_k .

- we compute the automaton $\mathcal{A}_{\mathcal{L}}$ that generates the language $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$.
- we compute the quotient automaton $\mathcal{A}_{\mathcal{L}/\equiv_k}$. This operation consists roughly in merging the states of $\mathcal{A}_{\mathcal{L}}$ that are equivalent w.r.t. \equiv_k .
- we minimize $\mathcal{A}_{\mathcal{L}/\equiv_k}$ to obtain a canonical representation.

In [LGJJ06], it is ensured that ∇ is a widening operator. In fact, this widening operator consists in merging the states of $\mathcal{A}_{\mathcal{L}}$ that are equivalent w.r.t. \equiv_k to obtain a new automaton $\mathcal{A}_{\mathcal{L}/\equiv_k}$ recognizing a regular language, which is greater than \mathcal{L} .

Example 6.5

We consider the automaton \mathcal{A} depicted in Figure 6.5(a), whose marked language is $a + ba + bba + bbba$. The equivalence classes of \equiv_0 are $\{0\}$ (initial state), $\{4\}$ (marked state) and $\{1, 2, 3\}$ (other states). The computation of the quotient $\mathcal{A}_{/\equiv_0}$ gives an automaton whose marked language is b^*a (see Figure 6.5(b)). The relation \equiv_1 , which is a refinement of \equiv_0 , has four equivalence classes $\{0\}$, $\{4\}$, $\{1, 2\}$ and $\{3\}$, because there is no outgoing transition labeled by b in the state 3.

We still need to define the concretization and the abstraction functions. The concretization function γ , that maps an element of the abstract lattice into an element of the concrete lattice, is defined by the identity function. The abstraction function α , that maps an element of the concrete lattice into an element of the abstract lattice, is also defined by the identity function. This definition of α is sufficient, because all operations used to compute the **Reachable**

and **Coreach** operators (i.e., \cup , \cap , **Pre**, **Post**, ...) are stable for the domain $L \rightarrow \text{Reg}(M)$ and hence we do not need to apply this abstraction function to languages that are not regular¹⁰ (we assume, however, that *Bad* is given by a regular language).

Effective Algorithm for the Distributed Problem. The effective algorithm is quite similar to the semi-algorithm defined in section 6.5.1: the main difference is that we compute an overapproximation of the functions **Reachable** and **Coreach** to ensure the termination of our algorithm. To compute this overapproximation, we use Theorem 1.4 with the instantiation of the representation framework defined above. This theorem requires to transpose the functions **Reachable** and **Coreach** into the abstract lattice. Since the operations used to compute the functions **Reachable** and **Coreach** are stable for the domain $L \rightarrow \text{Reg}(M)$ and since γ and α are defined by the identity function, these transpositions are immediate and the abstract function **Reachable**[#] (resp. **Coreach**[#]) is identical to the concrete function **Reachable** (resp. **Coreach**). Finally, we use the fixpoint computation strategy, defined in Theorem 1.4, with the abstract functions **Reachable**[#] (resp. **Coreach**[#]) and with the widening operator ∇ to obtain an overapproximation of the functions **Reachable** (resp. **Coreach**) in finite number of steps. One can note that the approximations are only due to the widening operator.

Remark 6.2

When the set N of \mathcal{T} contains one queue, the content of this queue is represented by a word and a set of contents is represented by a regular language (in fact, by the canonical automaton which represents this language). When $N = \{Q_1, \dots, Q_{|N|}\}$ contains at least two queues, we may represent the content of these queues by a concatenated word $w_1\# \dots \#w_{|N|}$ where w_i represents the content of the queue Q_i ($\forall i \in [1, |N|]$) and $\#$ is a special letter separating the content of the different queues [LGJJ06]. A QDD [BGWW97] is a finite automaton that recognizes sets of contents of a finite collection of unbounded FIFO queues and it can then be used to represent a set of concatenated words $w_1\# \dots \#w_{|N|}$. This encoding is interesting, because a set of contents of the queues $Q_1, \dots, Q_{|N|}$ is represented by a finite automaton and thus we can proceed as in the case, where there is one queue, to formalize the abstractions.

¹⁰Remember that initially the queues are empty and hence they can be represented by a regular language.

We must only use a slightly different widening operator not to merge the different queue contents [LGJJ06].

6.6 Proofs of Propositions 6.4 and 6.5

In this section, we prove Propositions 6.4 and 6.5. But, before that, we present two lemmas that we will use in these proofs.

6.6.1 Lemmas

First, we prove the following lemma.

Lemma 6.2

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, a CFMSM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, and a sequence $se_1 = \vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_{i-1}} \vec{v}_{i-1} \xrightarrow{e_i} \vec{v}_i \xrightarrow{e_{i+1}} \vec{v}_{i+1} \xrightarrow{e_{i+2}} \dots \xrightarrow{e_m} \vec{v}_m$ executed by \mathcal{T} , if $e_i \not\prec_c e_{i+1}$, then the sequence $se_2 = \vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_{i-1}} \vec{v}_{i-1} \xrightarrow{e_{i+1}} \vec{v}'_i \xrightarrow{e_i} \vec{v}_{i+1} \xrightarrow{e_{i+2}} \dots \xrightarrow{e_m} \vec{v}_m$ can also occur in the system \mathcal{T} .

Proof: The transition corresponding to the event e_k ($\forall k \in [1, m]$) is denoted by δ_{e_k} . We suppose that $\delta_{e_i} = \langle \ell_{e_i}, \sigma_{e_i}, \ell'_{e_i} \rangle \in \Delta_i$ and $\delta_{e_{i+1}} = \langle \ell_{e_j}, \sigma_{e_j}, \ell'_{e_j} \rangle \in \Delta_j$. Note that $i \neq j$; otherwise, we would have $e_i \prec_c e_{i+1}$ (by Definition 6.9).

Since the system is deterministic, we can prove this property by showing that $\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\vec{v}_{i-1})) = \text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\vec{v}_{i-1}))$. For that, we consider two cases:

- δ_{e_i} and $\delta_{e_{i+1}}$ act on different queues: we suppose that δ_{e_i} and $\delta_{e_{i+1}}$ respectively act on the queues Q_{k_i} and Q_{k_j} . We also suppose that $\vec{v}_{i-1} = \langle \ell_1, \dots, \ell_{e_i}, \dots, \ell_{e_j}, \dots, \ell_n, w_1, \dots, w_{k_i}, \dots, w_{k_j}, \dots, w_{|N|} \rangle$ (where w_{k_i} and w_{k_j} respectively denote the content of the queues Q_{k_i} and Q_{k_j}), and that the action σ_{e_i} (resp. σ_{e_j}), which acts on the content w_{k_i} (resp. w_{k_j}), modifies it to give w'_{k_i} (resp. w'_{k_j}). In consequence, $\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\vec{v}_{i-1}) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell_{e_j}, \dots, \ell_n, w_1, \dots, w'_{k_i}, \dots, w_{k_j}, \dots, w_{|N|} \rangle$ and $\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\vec{v}_{i-1})) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w'_{k_i}, \dots, w'_{k_j}, \dots, w_{|N|} \rangle$. Since $e_i \not\prec_c e_{i+1}$, we have that $\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\vec{v}_{i-1}) = \langle \ell_1, \dots, \ell_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w_{k_i}, \dots, w'_{k_j}, \dots, w_{|N|} \rangle$ and $\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\vec{v}_{i-1})) =$

$\langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w'_{k_i}, \dots, w'_{k_j}, \dots, w_{|N|} \rangle$, which implies that $\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\vec{v}_{i-1})) = \text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\vec{v}_{i-1}))$.

- δ_{e_i} and $\delta_{e_{i+1}}$ act on the same queue Q_k : we consider two cases:
 - $\sigma_i = Q_k!m_i$ is an output and $\sigma_{i+1} = Q_k?m_j$ is an input: the message written by δ_{e_i} cannot be read by the transition $\delta_{e_{i+1}}$, because, in this case, we would have $e_i \prec_c e_{i+1}$. Thus, $\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\vec{v}_{i-1})) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w.m_i, \dots, w_{|N|} \rangle$ where $w.m_i$ is the content of the queue Q_k . Therefore, the state $\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\vec{v}_{i-1}) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell_{e_j}, \dots, \ell_n, w_1, \dots, m_j.w.m_i, \dots, w_{|N|} \rangle$ and the state $\vec{v}_{i-1} = \langle \ell_1, \dots, \ell_{e_i}, \dots, \ell_{e_j}, \dots, \ell_n, w_1, \dots, m_j.w, \dots, w_{|N|} \rangle$. Next, we compute the state $\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\vec{v}_{i-1}) = \langle \ell_1, \dots, \ell_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w, \dots, w_{|N|} \rangle$ and the state $\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\vec{v}_{i-1})) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w.m_i, \dots, w_{|N|} \rangle$. In consequence, $\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\vec{v}_{i-1})) = \text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\vec{v}_{i-1}))$.
 - $\sigma_i = Q_k?m_i$ is an input and $\sigma_{i+1} = Q_k!m_j$ is an output: the state $\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\vec{v}_{i-1})) = \langle \ell_1, \dots, \ell'_{e_i}, \dots, \ell'_{e_j}, \dots, \ell_n, w_1, \dots, w.m_j, \dots, w_{|N|} \rangle$ where $w.m_j$ is the content of the queue Q_k . Next, similarly to the previous case, we can prove that $\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\vec{v}_{i-1})) = \text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Post}_{\delta_{e_{i+1}}}^{\mathcal{T}}(\vec{v}_{i-1}))$.

The cases, where δ_{e_i} and $\delta_{e_{i+1}}$ are both an input or an output, are not possible, because these transitions would then be executed by the same process and hence we would have $e_i \prec_c e_{i+1}$. \square

This property means that if two consecutive events e_i and e_{i+1} are such that $e_i \not\prec_c e_{i+1}$, then these events can be swapped without modifying the reachability of \vec{v}_m .

Now, we prove the second lemma.

Lemma 6.3

Given n subsystems $\mathcal{T}_i = \langle L_i, \ell_{0,i}, N_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) communicating according to the architecture defined in section 6.2, a CFMSM $\mathcal{T} = \mathcal{T}_1 || \dots || \mathcal{T}_n = \langle L, \ell_0, N, M, \Sigma, \Delta \rangle$, a transition $\delta_i = \langle \ell_i, Q_{t,i}?m_i, \ell'_i \rangle \in \Delta_i$ (with $t \neq i$), and a set of states $B \subseteq L \times (M^*)^{|N|}$, then $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(B))) = \text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(B))$.

Proof: The inequality $\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(B)) \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(B)))$ holds trivially.

To prove the other inclusion, we have to show that if a state $\vec{v}_m \in \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(B)))$, then $\vec{v}_m \in \text{Post}_{\delta_{e_i}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(B))$. We actually prove a more general result. We show that each sequence $\vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_{k-1}} \vec{v}_{k-1} \xrightarrow{e_k} \vec{v}_k \xrightarrow{e_{k+1}} \vec{v}_{k+1} \xrightarrow{e_{k+2}} \dots \xrightarrow{e_m} \vec{v}_m$ (where (i) $\vec{v}_0 \in B$, (ii) the event e_k corresponds to the transition $\delta_{e_k} = \delta_i \in \Delta_i$, and (iii) the event e_b , for each $b \neq k \in [1, m]$, corresponds to a transition $\delta_{e_b} \in \Delta \setminus \Delta_i$) can be reordered to execute e_k at the end of the sequence without modifying the reachability of \vec{v}_m i.e., the following sequence can occur: $\vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_{k-1}} \vec{v}_{k-1} \xrightarrow{e_{k+1}} \vec{v}'_{k+1} \xrightarrow{e_{k+2}} \dots \xrightarrow{e_m} \vec{v}'_m \xrightarrow{e_k} \vec{v}_m$. This reordered sequence can be obtained thanks to Lemma 6.2, but to use this lemma, we must prove that $e_k \not\prec_c e_b$ ($\forall b \in [k+1, m]$). The proof is by induction on the length of the sequence of events that begins from \vec{v}_k :

- **Base case:** we must prove that $e_k \not\prec_c e_{k+1}$. By Definition 6.9, since e_k and e_{k+1} occur in different subsystems and are consecutive events, there is one possibility to have $e_k \prec_c e_{k+1}$: it is when e_k is an output and e_{k+1} is the corresponding input. But e_k is an input and hence $e_k \not\prec_c e_{k+1}$.
- **Induction step:** we suppose that $e_k \not\prec_c e_{k+r}$ ($\forall r \in [1, j]$) and we prove that $e_k \not\prec_c e_{k+j+1}$. By Definition 6.9, since e_k and e_{k+1} occur in different subsystems, there are two possibilities to have $e_k \prec_c e_{k+j+1}$:
 - e_k is an output and e_{k+j+1} is the corresponding input. However, e_k is an input and hence $e_k \not\prec_c e_{k+j+1}$.
 - $e_k \prec_c e_{k+r}$ (with $r \in [1, j]$) and $e_{k+r} \prec_c e_{k+j+1}$. But by induction hypothesis, $e_k \not\prec_c e_{k+r}$ ($\forall r \in [1, j]$) and hence $e_k \not\prec_c e_{k+j+1}$.

Thus, in both cases $e_k \not\prec_c e_{k+j+1}$. □

6.6.2 Proof of Proposition 6.4

To show that this proposition holds, we prove by induction on the length m of an execution $\vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{v}_m$ of the system \mathcal{T} that $\forall i \in [1, n] : \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_m) \subseteq CS_i^m$. Since $\vec{v}_m \in \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_m)$, we have then that $\vec{v}_m \in CS_i^m$. In this proof, the transition corresponding to an event e_k is denoted by δ_{e_k} .

- **Base case ($m = 0$):** For each $i \in [1, n]$, the set $CS_i^0 = \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle)$ (see Algorithm 6). Therefore, we

have that $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_0) = CS_i^0$ ($\forall i \in [1, n]$), because $\vec{v}_0 = \langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle$.

- **Induction step:** We suppose that the property holds for the executions of length $k \leq m$ (i.e., $\forall 0 \leq k \leq m, \forall i \in [1, n] : \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_k) \subseteq CS_i^k$) and we prove that the property also holds for the executions of length $m + 1$. For that, we suppose that the event e_{m+1} has been executed by \mathcal{T}_i . We must consider two cases:

1. $\delta_{e_{m+1}}$ is an output on the queue $Q_{i,k}$ (with $k \neq i \in [1, n]$): We must prove that $\forall j \in [1, n] : \text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq CS_j^{m+1}$ and we consider again two cases:

- (a) $j \neq i$: By induction hypothesis, we know that $\text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_m) \subseteq CS_j^m$. Moreover, we have that:

$$\begin{aligned}
& \vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_m), \text{ by definition of Reachable} \\
\Rightarrow & \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{v}_m) \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_m)), \text{ because the} \\
& \text{function Post is monotonic} \\
\Rightarrow & \vec{v}_{m+1} \subseteq \text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_m), \text{ because } \delta_{e_{m+1}} \in \Delta \setminus \Delta_j \\
& \text{(as } \delta_{e_{m+1}} \in \Delta_i \text{) and } \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{v}_m) = \vec{v}_{m+1} \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_m)), \\
& \text{because the function Reachable is monotonic} \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_m) \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq CS_j^m, \text{ by induction hypothesis} \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq CS_j^{m+1}, \text{ because } CS_j^m = CS_j^{m+1} \\
& \text{(due to the fact that } e_{m+1} \text{ has not been executed by } \mathcal{T}_j \text{)}
\end{aligned}$$

- (b) $j = i$: By induction hypothesis, we know that $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_m) \subseteq CS_i^m$. The set $CS_i^{m+1} = \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(CS_i^m))$ (see Algorithm 7). Moreover, we have that:

$$\begin{aligned}
& \vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_m) \\
\Rightarrow & \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{v}_m) \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_m)) \\
\Rightarrow & \vec{v}_{m+1} \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_m)), \text{ as } \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{v}_m) = \vec{v}_{m+1}
\end{aligned}$$

$$\begin{aligned}
&\Rightarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq \\
&\quad \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_m))) \\
&\Rightarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(CS_i^m)), \text{ by} \\
&\quad \text{induction hypothesis} \\
&\Rightarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq CS_i^{m+1}, \text{ by definition of } CS_i^{m+1}
\end{aligned}$$

Thus, for each $j \in [1, n]$, we have that $\text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq CS_j^{m+1}$. Moreover, since we compute an overapproximation of CS_j^{m+1} ($\forall j \in [1, n]$), this inclusion remains true¹¹.

2. $\delta_{e_{m+1}}$ is an input on the queue $Q_{k,i}$ (with $k \neq i \in [1, n]$): We must prove that $\forall j \in [1, n] : \text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq CS_j^{m+1}$ and we consider again two cases:

- (a) $j \neq i$: The proof is similar to the one given in the case where $\delta_{e_{m+1}}$ in an output.
- (b) $j = i$: By induction hypothesis, we know that $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_m) \subseteq CS_i^m$. By Algorithm 8, the set $CS_i^{m+1} = \text{Temp}_1 \cap \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(CS_i^m)$ (in our algorithm, the set Temp_1 can have three possible values). To prove that $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq CS_i^{m+1}$, we first prove that $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(CS_i^m)$ and next we show that $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq \text{Temp}_1$. The first inclusion is proved as follows:

$$\begin{aligned}
&\vec{\nu}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_m) \\
&\Rightarrow \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{\nu}_m) \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_m)) \\
&\Rightarrow \vec{\nu}_{m+1} \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_m)), \text{ since } \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{\nu}_m) = \vec{\nu}_{m+1} \\
&\Rightarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq \\
&\quad \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_m))) \\
&\Rightarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_m)), \text{ by Lemma} \\
&\quad 6.3 \\
&\Rightarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{\nu}_{m+1}) \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(CS_i^m), \text{ by induction hypothesis}
\end{aligned}$$

¹¹One can note that if we compute an underapproximation of CS_j^{m+1} , the inclusion does not always hold.

To prove the second inclusion, we must consider three cases which depend on the definition of $Temp_1$. Let e_t (with $t \leq m$) be the output (executed by \mathcal{T}_k with $k \neq i \in [1, n]$) corresponding to the input e_{m+1} :

A. $Temp_1 = \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1})))$ and $V_k[i] = V_i[i]$ (as a reminder, V_k represents the vector clock of \mathcal{T}_k after the occurrence of the event e_t and V_i represents the vector clock of \mathcal{T}_i before the occurrence of the event e_{m+1}): By induction hypothesis, we know that $\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_{t-1}) \subseteq CS_k^{t-1}$. Moreover, we have that:

$$\begin{aligned}
& \vec{v}_{t-1} \subseteq \text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_{t-1}) \\
\Rightarrow & \vec{v}_{t-1} \subseteq CS_k^{t-1}, \text{ by induction hypothesis} \\
\Rightarrow & \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(\vec{v}_{t-1}) \subseteq \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}) \\
\Rightarrow & \vec{v}_t \subseteq \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}), \text{ because } \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(\vec{v}_{t-1}) = \vec{v}_t \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_t) \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1})) \quad (\beta)
\end{aligned}$$

However, since $V_k[i] = V_i[i]$, we know that, between the moment where e_t has been executed and the moment where e_m has been executed, the vector clock $V_i[i]$ has not been modified. Thus, during this period no transition of \mathcal{T}_i has been executed. In consequence, we have that $\vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_t)$ and hence $\vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}))$ by (β) . From this inclusion, we deduce that:

$$\begin{aligned}
& \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{v}_m) \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}))) \\
\Rightarrow & \vec{v}_{m+1} \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}))), \text{ because} \\
& \vec{v}_{m+1} = \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{v}_m) \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \\
& \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1})))) \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \\
& \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}))), \text{ by Lemma 6.3} \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq Temp_1, \text{ by definition of } Temp_1
\end{aligned}$$

B. $Temp_1 = \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}))))$ and $V_k[k] > V_i[k]$ (as a reminder, V_k represents the vector clock of

\mathcal{T}_k after the occurrence of the event e_t and V_i represents the vector clock of \mathcal{T}_i before the occurrence of the event e_{m+1}): By induction hypothesis, we know that $\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_{t-1}) \subseteq CS_k^{t-1}$. Moreover, we have that:

$$\begin{aligned}
& \vec{v}_{t-1} \subseteq \text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_{t-1}) \\
\Rightarrow & \vec{v}_{t-1} \subseteq CS_k^{t-1}, \text{ by induction hypothesis} \\
\Rightarrow & \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(\vec{v}_{t-1}) \subseteq \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}) \\
\Rightarrow & \vec{v}_t \subseteq \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}), \text{ because } \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(\vec{v}_{t-1}) = \vec{v}_t \quad (\gamma)
\end{aligned}$$

This inclusion is used further in the proof. Now, we prove that $\vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t))$. For that, let us consider the subsequence $se = \vec{v}_t \xrightarrow{e_{t+1}} \vec{v}_{t+1} \xrightarrow{e_{t+2}} \dots \xrightarrow{e_m} \vec{v}_m$ of the execution $\vec{v}_0 \xrightarrow{e_1} \vec{v}_1 \xrightarrow{e_2} \dots \xrightarrow{e_m} \vec{v}_m$. Let e_{K_1} be the first event of the sequence se executed¹² by \mathcal{T}_k and $s_I = e_{I_1}, \dots, e_{I_\ell}$ (with $I_1 < \dots < I_\ell$) be the events of the sequence se executed¹³ by \mathcal{T}_i . If $I_\ell < K_1$ (i.e., e_{I_ℓ} has been executed before e_{K_1}), then $\vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t))$, because all events of the sequence se executed by \mathcal{T}_i have been executed before the first event e_{K_1} of \mathcal{T}_k . Otherwise, let $s_{I'} = e_{I_d}, \dots, e_{I_\ell}$ be the events of s_I executed after e_{K_1} . We must reorder the sequence se to obtain a new sequence where all actions of \mathcal{T}_i are executed before the ones of \mathcal{T}_k and \vec{v}_m remains reachable. Lemma 6.2 allows us to swap two consecutive events without modifying the reachability when these events are not causally dependent. To use this lemma, we must prove that the events $e_{I_d}, \dots, e_{I_\ell}$ do not causally depend on e_{K_1} . For that, we first prove that $e_{K_1} \not\prec_c e_{I_\ell}$. By assumption, we know that $V_k[k] > V_i[k]$. V_k represents the vector clock of \mathcal{T}_k after the execution of e_t and V_i represents the vector clock of \mathcal{T}_i before the execution of e_{m+1} , which gives $V_k(e_t)[k] > V_i(e_m)[k]$. Moreover, $V_i(e_m)[k] \geq V_i(e_{I_\ell})[k]$ (because e_{I_ℓ} has been executed before¹⁴ e_m) and $V_k(e_{K_1})[k] \geq V_k(e_t)[k] + 1$ (because e_{K_1} is the

¹²If this element does not exist, then the transitions executed in this sequence do not belong to Δ_k ; thus, $\vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t)$ and hence $\vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t))$

¹³If the sequence s_I is empty, then the transitions executed in the sequence se do not belong to Δ_i ; thus, $\vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_t)$ and hence $\vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t))$, because $\vec{v}_t \subseteq \text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t)$

¹⁴One can note that e_{I_ℓ} may be equal to e_m .

event which follows e_t in the execution of the subsystem \mathcal{T}_k). Thus, $V_k(e_{K_1})[k] > V_i(e_{I_\ell})[k]$, and hence $e_{K_1} \not\prec_c e_{I_\ell}$. Next, since $e_{I_c} \prec_c e_{I_\ell}$ ($\forall e_{I_c} \neq e_{I_\ell} \in s_I$) and since $e_{K_1} \not\prec_c e_{I_\ell}$, we have by Proposition 6.3 that $e_{K_1} \not\prec_c e_{I_c}$. Now, in the sequence se , we will move the events $e_{I_d}, \dots, e_{I_\ell}$ to execute them before e_{K_1} without modifying the reachability of \vec{v}_m . We start by moving the element e_{I_d} . To obtain a sequence where e_{I_d} precedes e_{K_1} , we swap e_{I_d} with the events which precede it and we repeat this operation until the event e_{K_1} . Lemma 6.2 ensures that \vec{v}_m remains reachable if e_{I_d} is swapped with an element e' such that $e' \not\prec_c e_{I_d}$. However, between e_{K_1} and e_{I_d} there can be some events, that *happened-before* e_{I_d} . We must thus move these events before moving e_{I_d} . More precisely, let $s_b = e_{b_1}, \dots, e_{b_p}$ (with $b_1 < \dots < b_p$) be the greatest sequence of events such that (i) these events are executed between the occurrence of e_{K_1} and the occurrence of e_{I_d} and (ii) $\forall e_{b_c} \in s_b : e_{b_c} \prec_c e_{I_d}$ (note that the events of the sequence s_b are not executed by \mathcal{T}_k ; otherwise, we would have $e_{K_1} \prec_c e_{I_d}$). The sequence of events $s = e_{K_1}, e_{K_1+1}, e_{K_1+2}, \dots, e_{b_1-1}$ executed between e_{K_1} and e_{b_1} is such that $\forall e_{t'} \in s : e_{t'} \not\prec_c e_{b_1}$. Indeed, if $e_{t'} \prec_c e_{b_1}$, then by transitivity we would have $e_{t'} \prec_c e_{I_d}$, but this is not possible, because $e_{t'} \notin s$. Thus, by Lemma 6.2, in the sequence $\vec{v}_t \xrightarrow{e_{t+1}} \dots \xrightarrow{e_{K_1}} \vec{v}_{K_1} \xrightarrow{e_{K_1+1}} \vec{v}_{K_1+1} \xrightarrow{e_{K_1+2}} \dots \xrightarrow{e_{b_1-1}} \vec{v}_{b_1-1} \xrightarrow{e_{b_1}} \vec{v}_{b_1} \xrightarrow{e_{b_1+1}} \dots \xrightarrow{e_m} \vec{v}_m$, we can *safely* swap the events e_{b_1-1} and e_{b_1} . We then obtain a reordered sequence where \vec{v}_m remains reachable i.e., we obtain $\vec{v}_t \xrightarrow{e_{t+1}} \dots \xrightarrow{e_{K_1}} \vec{v}_{K_1} \xrightarrow{e_{K_1+1}} \vec{v}_{K_1+1} \xrightarrow{e_{K_1+2}} \dots \xrightarrow{e_{b_1-2}} \vec{v}_{b_1-2} \xrightarrow{e_{b_1}} \vec{v}'_{b_1} \xrightarrow{e_{b_1-1}} \vec{v}_{b_1} \xrightarrow{e_{b_1+1}} \dots \xrightarrow{e_m} \vec{v}_m$. By repeating this swap with the events $e_{b_1-2}, e_{b_1-3}, \dots, e_{K_1+1}, e_{K_1}$, we obtain a reordered sequence where (i) e_{b_1} is executed before e_{K_1} and (ii) \vec{v}_m remains reachable (by Lemma 6.2). We repeat the operations performed for e_{b_1} with the events e_{b_2}, \dots, e_{b_p} and e_{I_d} to obtain a reordered sequence where (i) e_{I_d} is executed before e_{K_1} and (ii) \vec{v}_m is reachable. Finally, we repeat the operations performed for e_{I_d} with the other elements of the sequence s_I to obtain a reordered sequence where (i) \vec{v}_m is reachable from \vec{v}_t and (ii) the events of \mathcal{T}_i are executed before the ones of \mathcal{T}_k , which implies that $\vec{v}_m \subseteq \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t))$.

Next, from this inclusion, we deduce that:

$$\begin{aligned}
& \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{v}_m) \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t))) \\
\Rightarrow & \vec{v}_{m+1} \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t))), \text{ because} \\
& \vec{v}_{m+1} = \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{v}_m) \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \\
& \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t)))) \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \\
& \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_t))), \text{ by Lemma 6.3} \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \\
& \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1})))), \\
& \text{ by } (\gamma) \\
\Rightarrow & \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \text{Temp}_1, \text{ by definition of } \text{Temp}_1
\end{aligned}$$

C. $\text{Temp}_1 = \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1})))$: By induction hypothesis, we know that $\text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_{t-1}) \subseteq CS_k^{t-1}$. Moreover, we have that:

$$\begin{aligned}
& \vec{v}_{t-1} \subseteq \text{Reachable}_{\Delta \setminus \Delta_k}^{\mathcal{T}}(\vec{v}_{t-1}) \\
\Rightarrow & \vec{v}_{t-1} \subseteq CS_k^{t-1}, \text{ by induction hypothesis} \\
\Rightarrow & \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(\vec{v}_{t-1}) \subseteq \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}) \\
\Rightarrow & \vec{v}_t \subseteq \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}), \text{ because } \text{Post}_{\delta_{e_t}}^{\mathcal{T}}(\vec{v}_{t-1}) = \vec{v}_t \\
\Rightarrow & \text{Reachable}_{\Delta}^{\mathcal{T}}(\vec{v}_t) \subseteq \text{Reachable}_{\Delta}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1})) \quad (\alpha)
\end{aligned}$$

However, the events e_{t+1}, \dots, e_m which lead to the state \vec{v}_m from the state \vec{v}_t correspond to transitions which belong to Δ . Thus, $\vec{v}_m \subseteq \text{Reachable}_{\Delta}^{\mathcal{T}}(\vec{v}_t)$ and hence $\vec{v}_m \subseteq \text{Reachable}_{\Delta}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}))$ by (α) . From this inclusion, we deduce that:

$$\begin{aligned}
& \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{v}_m) \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}))) \\
\Rightarrow & \vec{v}_{m+1} \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}))), \text{ because } \vec{v}_{m+1} \\
& = \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\vec{v}_m) \\
\Rightarrow & \vec{v}_{m+1} \subseteq \text{Reachable}_{\Delta}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1})), \text{ because } \delta_{e_{m+1}} \in \Delta
\end{aligned}$$

$$\begin{aligned}
&\Rightarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \\
&\quad \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\text{Reachable}_{\Delta}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}))) \\
&\Rightarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \text{Reachable}_{\Delta}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1})), \text{ because} \\
&\quad \Delta \setminus \Delta_i \subseteq \Delta \\
&\Rightarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \text{Post}_{\delta_{e_{m+1}}}^{\mathcal{T}}(\text{Reachable}_{\Delta}^{\mathcal{T}}(\text{Post}_{\delta_{e_t}}^{\mathcal{T}}(CS_k^{t-1}))), \\
&\quad \text{because } \delta_{m+1} \in \Delta \\
&\Rightarrow \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \text{Temp}_1, \text{ by definition of } \text{Temp}_1
\end{aligned}$$

In conclusion, we have proven, for each definition of Temp_1 , that $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq \text{Temp}_1$ and hence we have proven that $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq CS_i^{m+1}$.

Thus, for each $j \in [1, n]$, we have that $\text{Reachable}_{\Delta \setminus \Delta_j}^{\mathcal{T}}(\vec{v}_{m+1}) \subseteq CS_j^{m+1}$. Moreover, since we compute an overapproximation of CS_j^{m+1} ($\forall j \in [1, n]$), this inclusion remains true. \square

6.6.3 Proof of Proposition 6.5

To show that this proposition holds, we prove by induction on the length m of the sequences of events e_1, \dots, e_m (let $\langle \ell_{e_k}, \sigma_{e_k}, \ell'_{e_k} \rangle$ be the transition corresponding to e_k , for each $k \in [1, m]$) executed by the system that $\forall i \in [1, n] : CS_i^m \subseteq \bigcup_{\vec{\sigma} \in P_i^{-1}(P_i(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m}))} \Delta(\vec{v}_0, \vec{\sigma})$

- **Base case ($m = 0$):** The initial state $\vec{v}_0 = \langle \ell_{0,1}, \dots, \ell_{0,n}, \epsilon, \dots, \epsilon \rangle$ and we must prove that $\forall i \in [1, n] : CS_i^0 \subseteq \bigcup_{\vec{\sigma} \in P_i^{-1}(P_i(\epsilon))} \Delta(\vec{v}_0, \vec{\sigma})$. The set $CS_i^0 = \text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_0)$ (see Algorithm 6) and $\text{Reachable}_{\Delta \setminus \Delta_i}^{\mathcal{T}}(\vec{v}_0) = \bigcup_{\vec{\sigma} \in P_i^{-1}(P_i(\epsilon))} \Delta(\vec{v}_0, \vec{\sigma})$, which implies that $CS_i^0 = \bigcup_{\vec{\sigma} \in P_i^{-1}(P_i(\epsilon))} \Delta(\vec{v}_0, \vec{\sigma})$. Moreover, since we compute an underapproximation of CS_i^0 ($\forall j \in [1, n]$), this inclusion remains true¹⁵.
- **Induction step:** We suppose that the property holds for the sequences of events of length $k \leq m$ and we prove that the property remains true for the sequences of length $m+1$. We suppose that e_{m+1} has been executed by \mathcal{T}_i and that $\delta_{e_{m+1}} = \langle \ell_{e_{m+1}}, \sigma_{e_{m+1}}, \ell'_{e_{m+1}} \rangle$ is the transition corresponding to e_{m+1} . We consider two cases:

¹⁵One can note that if we compute an overapproximation of the reachable states, the inclusion does not always hold.

1. $\delta_{e_{m+1}}$ is an output: We must prove that $\forall j \in [1, n] : CS_j^{m+1} \subseteq \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$ and we consider again two cases:
 - (a) $i \neq j$: By induction hypothesis, we know that $CS_j^m \subseteq \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m}))} \Delta(\vec{\nu}_0, \bar{\sigma})$. Since $CS_j^{m+1} = CS_j^m$ (by definition), we have that $CS_j^{m+1} \subseteq \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m}))} \Delta(\vec{\nu}_0, \bar{\sigma})$. Moreover, $\bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m}))} \Delta(\vec{\nu}_0, \bar{\sigma}) = \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$, because $P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m}) = P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}})$ (this equality is due to the fact that $\sigma_{e_{m+1}} \notin \Sigma_j$). Therefore, we have that $CS_j^{m+1} \subseteq \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$. Moreover, since we compute an underapproximation of CS_j^{m+1} , this inclusion remains true.
 - (b) $i = j$: The set $CS_j^{m+1} = \text{Reachable}_{\Delta \setminus \Delta_j}^T(\text{Post}_{\delta_{e_{m+1}}}^T(CS_j^m))$ and $P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}) = P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m}) \cdot \sigma_{e_{m+1}}$, because $\sigma_{e_{m+1}} \in \Sigma_j$. We prove that if $\vec{\nu} \in CS_j^{m+1}$, then $\vec{\nu} \in \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$. If $\vec{\nu} \in CS_j^{m+1}$, then there exists a state $\vec{\nu}' \in CS_j^m$ such that $\vec{\nu} \in \text{Reachable}_{\Delta \setminus \Delta_j}^T(\text{Post}_{\delta_{e_{m+1}}}^T(\vec{\nu}'))$. Let $\langle \ell_{e_{m+1}}, \sigma_{e_{m+1}}, \ell'_{e_{m+1}} \rangle, \langle \ell_{t_1}, \sigma_{t_1}, \ell'_{t_1} \rangle, \dots, \langle \ell_{t_k}, \sigma_{t_k}, \ell'_{t_k} \rangle$ be the sequence of transitions which leads to $\vec{\nu}$ from the state $\vec{\nu}'$ i.e., $\vec{\nu} = \Delta(\vec{\nu}', \sigma_{e_{m+1}} \cdot \sigma_{t_1} \dots \sigma_{t_k})$. The transition $\langle \ell_{t_b}, \sigma_{t_b}, \ell'_{t_b} \rangle \in \Delta \setminus \Delta_j$ (for each $b \in [1, k]$), which implies that $\sigma_{e_{m+1}} \cdot \sigma_{t_1} \dots \sigma_{t_k} \in P_j^{-1}(\sigma_{e_{m+1}})$. Moreover, by induction hypothesis, the state $\vec{\nu}' \in \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m}))} \Delta(\vec{\nu}_0, \bar{\sigma})$, which implies that $\exists \bar{\sigma}' \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m})) : \vec{\nu}' = \Delta(\vec{\nu}_0, \bar{\sigma}')$. Since $P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}})) = [P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m})) \cdot P_j^{-1}(\sigma_{e_{m+1}})]$, the sequence $\bar{\sigma}'' = \bar{\sigma}' \cdot \sigma_{e_{m+1}} \cdot \sigma_{t_1} \dots \sigma_{t_k}$ belongs to $P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))$. Moreover, $\vec{\nu} = \Delta(\vec{\nu}_0, \bar{\sigma}'')$ (because $\vec{\nu}' = \Delta(\vec{\nu}_0, \bar{\sigma}')$ and $\vec{\nu} = \Delta(\vec{\nu}', \sigma_{e_{m+1}} \cdot \sigma_{t_1} \dots \sigma_{t_k})$) which implies that $\vec{\nu} \in \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$. Therefore, we have that $CS_j^{m+1} \subseteq \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$. Again, since we compute an underapproximation of CS_j^{m+1} , this inclusion remains true.
2. $\delta_{e_{m+1}}$ is an input: We must prove that $\forall i \in [1, n] : CS_j^{m+1} \subseteq \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$ and we consider again two cases:
 - (a) $i \neq j$: The proof is similar the one given in the case where $\delta_{e_{m+1}}$ is an output.
 - (b) $i = j$: The set $CS_j^{m+1} = \text{Post}_{\delta_{e_{m+1}}}^T(CS_j^m) \cap \text{Temp}_1$ (see Algorithm 8). Thus, we have that $CS_j^{m+1} \subseteq \text{Post}_{\delta_{e_{m+1}}}^T(CS_j^m)$ and it then suf-

fices to prove that $\text{Post}_{\delta_{e_{m+1}}}^T(CS_j^m) \subseteq \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$. For that, we show that if $\vec{\nu} \in \text{Post}_{\delta_{e_{m+1}}}^T(CS_j^m)$, then $\vec{\nu} \in \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$. If $\vec{\nu} \in \text{Post}_{\delta_{e_{m+1}}}^T(CS_j^m)$, then there exists a state $\vec{\nu}' \in CS_j^m$ such that $\vec{\nu} = \text{Post}_{\delta_{e_{m+1}}}^T(\vec{\nu}')$. By induction hypothesis, the state $\vec{\nu}' \in \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m}))} \Delta(\vec{\nu}_0, \bar{\sigma})$, which implies that $\exists \bar{\sigma}' \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m})) : \vec{\nu}' = \Delta(\vec{\nu}_0, \bar{\sigma}')$. Since $P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}})) = [P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_m})) \cdot P_j^{-1}(\sigma_{e_{m+1}})]$, the sequence $\bar{\sigma}'' = \bar{\sigma}' \cdot \sigma_{e_{m+1}}$ belongs to $P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))$. Moreover, $\vec{\nu} = \Delta(\vec{\nu}_0, \bar{\sigma}'')$ (because $\vec{\nu}' = \Delta(\vec{\nu}_0, \bar{\sigma}')$ and $\vec{\nu} = \text{Post}_{\delta_{e_{m+1}}}^T(\vec{\nu}')$) which implies that $\vec{\nu} \in \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$. Therefore, we have that $CS_j^{m+1} \subseteq \bigcup_{\bar{\sigma} \in P_j^{-1}(P_j(\sigma_{e_1} \cdot \sigma_{e_2} \dots \sigma_{e_{m+1}}))} \Delta(\vec{\nu}_0, \bar{\sigma})$. Again, since we compute an underapproximation of CS_j^{m+1} , this inclusion remains true.

□

Part III

Control of Finite State Systems

Chapter 7

Computational Complexity of the Synthesis of Memoryless Controllers with Partial Observation for Finite State Systems

UNTIL now, we were interested in the control of infinite state systems. We have shown that computing a maximal memoryless controller for the state avoidance control problem is *undecidable*. To overcome this obstacle, we have defined a method where we characterize the set of states to be avoided by a fixpoint equation and use abstract interpretation techniques to ensure the termination of the computation of the least fixpoint of this equation. However, when the system to be controlled has a finite state space, abstract interpretation is not necessary to obtain an effective algorithm. Indeed, since the computation of the least fixpoint is performed in a finite lattice, it always terminates, and a controller can then be synthesized (when a solution exists). However, we may wonder whether the solutions can *quickly* be computed.

In this chapter, we are thus interested in the computational complexity of some problems related to the control of finite state systems with partial observation. In particular, we want to determine whether a maximal memoryless controller with partial observation can efficiently be computed for the state avoid-

ance control problem.

The remainder of this chapter is structured as follows. In section 7.1, we present the results, existing in the literature, regarding the computational complexity of the synthesis of memoryless controllers. In section 7.2, we recall some elements of complexity theory. Next, in section 7.3, we define the used framework, and we motivate and formalize several control problems one may want to solve or answer. Finally, in section 7.4, we study the complexity of these problems and show that no polynomial algorithm can solve them unless $P = NP$.

7.1 Related Works

We recall in this section the main results¹ regarding the computational complexity of the synthesis of memoryless controllers for the state avoidance control problem (see Table 7.1). A detailed description of these works is given in section 2.1.3.

In [Ush89], Ushio defines a polynomial time algorithm that synthesizes (when a solution exists) a controller with full observation, which allows the system to exactly reach the set Bad^c (as a reminder, Bad^c is the complement of the set Bad of forbidden states). One can note that this algorithm can be used when the deadlock free (resp. non-blocking) property must be ensured. Indeed, the controller, that this algorithm synthesizes, is the less restrictive w.r.t. the actions that are forbidden in each state of the system. Therefore, if this controller does not ensure the deadlock free (resp. non-blocking) property², then no controller, that allows the system to exactly reach Bad^c , can ensure this property. Moreover, when no controller allows the system to exactly reach Bad^c , Ushio [Ush89] provides a controller \mathcal{C} that allows the system to reach the largest possible subset of Bad^c (i.e., \mathcal{C} is optimal). As explained in section 2.1.3, this result can be extended to the cases where the deadlock free or non-blocking property must be ensured.

Under partial observation, Takai *et al.* provide, in [TUK95], a necessary and sufficient condition (called M -controllability) for the existence of a supervisor that allows the system to exactly reach the set of states Bad^c . This condition can be tested in polynomial time and when it is satisfied they provide a controller (that can be computed in polynomial time) solving this problem. One can note

¹We only focus on the state-based approach.

²These properties can be tested in polynomial time.

	Basic	Non-blocking	Deadlock free
Optimum with full observation	Polynomial	Polynomial	Polynomial
Exactly Bad^c with full observation	Polynomial	Polynomial	Polynomial
Maximum with partial observation	?	?	?
Exactly Bad^c with partial observation	Polynomial	Polynomial	Polynomial

Table 7.1 - Computational complexity of the synthesis of memoryless controllers for the state avoidance control problem.

that this result can be extended to the cases where the deadlock free or non-blocking property must be ensured. When the M -controllability condition is not satisfied, we would like to compute a controller that allows the system to reach the *largest* possible subset of Bad^c . However, as shown in Proposition 3.3, an optimal controller does not always exist and an approach to overcome this negative result is to compute a maximal controller. In [TK97, TK98], Takai *et al.* define an algorithm which synthesizes a controller that allows the system to reach a subset of Bad^c . However, this controller is not always maximal. To our knowledge, the computation of a maximal controller with partial observation for the state avoidance control problem has not been solved in the literature and our aim, in this chapter, is then to determine the complexity of this problem.

7.2 Review of Complexity Theory

In this section, we briefly recall some elements of complexity theory (see [GJ90, Pap94] for further details).

In complexity theory, we generally consider *decision problems*, because this class is sufficient to illustrate the basic concepts of this theory. A decision problem is a question where the only possible answers are either *yes* or *no*. For example, deciding if a state \vec{v} is reachable in an STS \mathcal{T} is a decision problem.

The class P is the set of decision problems that can be solved in polynomial time by a deterministic Turing machine. The class NP is the set of decision problems where a *yes* instance can be recognized in polynomial time by a non-

deterministic Turing machine. A problem π is NP-complete if it belongs to NP and if it is at least as hard as any other problem in NP (i.e., if there exists a polynomial time algorithm for π , then all problems in NP can be solved in polynomial time). Generally, we prove that π is NP-complete as follows:

1. we prove that $p \in \text{NP}$
2. we give a *polynomial transformation* from an NP-complete problem π' to π . Roughly, this reduction consists in defining an algorithm for π' which uses π as subalgorithm. Moreover, all operations of this algorithm, except the resolution of π , must have a polynomial time complexity.

The class co-NP is the set of decision problems where a *no* instance can be recognized in polynomial time by a non-deterministic Turing machine. Thus, a problem is in co-NP if its complement is in NP. Similarly, a problem is co-NP-complete if its complement is NP-complete.

One of the most known NP-complete problem is 3SAT [GJ90] which is formally defined as follows:

Problem 7.1 (3SAT)

Let ϕ be a CNF (Conjunctive Normal Form) formula over a set of variables $\mathcal{P} = \{p_0, \dots, p_k\}$: $\phi = \bigwedge_{m=1}^{n_c} c_m$, where $n_c > 0$ and the clauses $c_m = \bigvee_{j=1}^3 \ell_{m,j}$ ($\ell_{m,j}$ is a variable in \mathcal{P} or the negation of a variable in \mathcal{P}). 3SAT consists in deciding if ϕ is satisfiable. In other words, this problem consists in deciding if there exists a truth assignment for \mathcal{P} satisfying ϕ .

This problem will be used in this chapter to prove complexity results.

These concepts can be used to characterize the time complexity of an optimization problem π . Indeed, we can prove that this optimization problem cannot be solved in polynomial time (unless $P = NP$) by defining a polynomial transformation from an NP-complete problem to π .

7.3 Framework and Control Problems

In this chapter, we restrict our attention to finite state systems and we present the formalisms used in this context. The system to be controlled is modeled by a *finite* deterministic automaton $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$ and the partial observation is defined by an observer $\langle \mathcal{D}_{Obs}, M \rangle$, where (i) \mathcal{D}_{Obs} is a *finite* observation space and (ii) $M : X \rightarrow \mathcal{D}_{Obs}$ is a mask corresponding to a partition of the finite state

space. One can note that the complexity results, proved in this chapter, also hold when the system to be controlled is non-deterministic or when the mask is a covering of the state space.

As explained above, our aim is to determine whether a \preceq_p -maximal controller (i.e., a maximal controller w.r.t. the permissiveness criterion \preceq_p which is given in Definition 3.12) can efficiently be computed for the state avoidance control problem. In other words, we want to determine the time complexity of the following problem.

Problem 7.2 (\preceq_p -Maximal Basic State Avoidance Control Problem)

Given a deterministic automaton $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$, an observer $\langle \mathcal{D}_{Obs}, M \rangle$, and a predicate Bad , which represents a set of forbidden states, the \preceq_p -maximal basic state avoidance control problem (\preceq_p -maximal basic problem for short) consists in computing a memoryless controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ such that (i) $\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{C}}((x_0)_{/\mathcal{C}}) \neq \emptyset$, (ii) $\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{C}}((x_0)_{/\mathcal{C}}) \cap Bad = \emptyset$, and (iii) \mathcal{C} is \preceq_p -maximal.

Thus, this problem consists in computing a memoryless controller which is (i) non-trivial, (ii) valid, and (iii) \preceq_p -maximal. In section 7.4, we prove that this problem cannot be solved in polynomial time unless³ $P = NP$. An alternative to overcome this negative result is to define an algorithm that computes an approximate solution; for example, we can use our algorithm defined in chapter 3 to compute such a solution. We may then wonder whether a controller \mathcal{C} synthesized by such an algorithm is \preceq_p -maximal in order to evaluate the quality of this solution. Unfortunately, we can show (see section 7.4) that this *simpler* problem, which is formally defined as follows, is NP-complete:

Problem 7.3 (\preceq_p -Maximal Basic Decision Problem)

Given a deterministic automaton $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$, an observer $\langle \mathcal{D}_{Obs}, M \rangle$, a predicate Bad , which represents a set of forbidden states, and a memoryless controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$, the \preceq_p -maximal basic decision problem consists in deciding if \mathcal{C} is valid, non-trivial and \preceq_p -maximal.

An approach to overcome these negative results would be to measure the quality of the controllers with a comparison criterion which seems to be weaker than the permissiveness criterion \preceq_p : the criterion \preceq_c which is based on the

³In the sequel, we will omit to mention that P must be different from NP .

number of states that remain reachable after control⁴. Of course this criterion gives limited information since two solutions with the same number of reachable states may be completely different. However, one can hope to *efficiently* compute a maximal controller with this *simpler* criterion. Unfortunately, this problem cannot be solved in polynomial time, because we prove (see section 7.4) that the following decision problem is NP-complete:

Problem 7.4 (\preceq_c -Maximal Basic Decision Problem)

Given a deterministic automaton $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$, an observer $\langle \mathcal{D}_{Obs}, M \rangle$ and a predicate Bad , which represents a set of forbidden states, the \preceq_c -maximal basic decision problem consists in deciding if there exists a memoryless controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ such that (i) $\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{C}}((x_0)_{/\mathcal{C}}) \neq \emptyset$, (ii) $\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{C}}((x_0)_{/\mathcal{C}}) \cap Bad = \emptyset$, and (iii) $|\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{C}}((x_0)_{/\mathcal{C}})| \geq N$ (where $N \in \mathbb{N}$).

So, the partial observation makes *more difficult* the computation of a maximal controller. Therefore, we relax the constraint on the quality of the controllers that we want to compute: we do not ask to compute a maximal controller, but only a controller that allows the system to reach at least a set Min of states (Min can be seen as the minimum admissible behavior). Even with this simpler constraint of quality, a controller with partial observation cannot be computed in polynomial time, because we prove (see section 7.4) that the following decision problem is NP-complete:

Problem 7.5 (Interval Basic Problem)

Given a deterministic automaton $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$, an observer $\langle \mathcal{D}_{Obs}, M \rangle$ and two non-empty subsets $Min \subseteq X$ and $Max \subseteq X$ (with $Min \subseteq Max$), which give the minimal and the maximal set of allowable states, the interval basic problem consists in deciding if there exists a memoryless controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ such that $Min \subseteq \text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{C}}((x_0)_{/\mathcal{C}}) \subseteq Max$.

Min can be seen as the minimum admissible behavior and Max as the maximum admissible behavior (i.e., the controller must prevent the system from reaching Max^c).

⁴We can formally define this criterion \preceq_c as follows. Given an automaton $\mathcal{T} = \langle X, X_0, X_m, \Sigma, \Delta \rangle$ and an observer $\langle \mathcal{D}_{Obs}, M \rangle$, a controller $\mathcal{C}_1 = \langle \mathcal{S}_1, E_1 \rangle$ is better than a controller $\mathcal{C}_2 = \langle \mathcal{S}_2, E_2 \rangle$ w.r.t the criterion \preceq_c if and only if $|\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{C}_2}((x_0)_{/\mathcal{C}_2})| \leq |\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{C}_1}((x_0)_{/\mathcal{C}_1})|$.

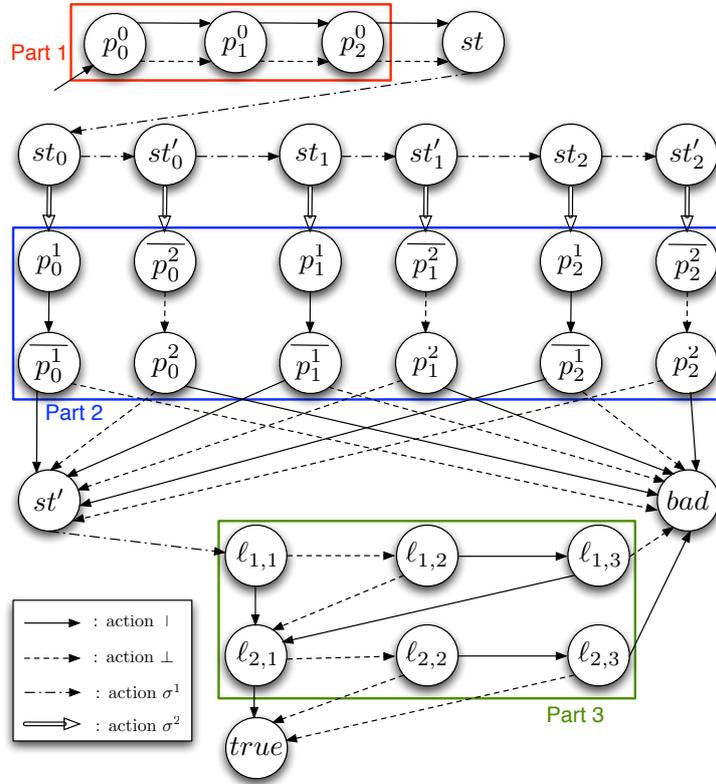


Figure 7.1 - Construction of \mathcal{T}_ϕ .

In the sequel, we also consider, for all problems defined above, the cases where the non-blocking or deadlock free property must be ensured.

7.4 Time Complexity of the Control Problems

In this section, we first give the time complexity of the problems defined in section 7.3. Next, we extend these results to the cases where the deadlock free or non-blocking property must be ensured.

7.4.1 The Basic Case

We use a reduction from 3SAT to the \preceq_p -maximal basic problem to prove that this problem cannot be solved in polynomial time. Let us first illustrate the principle of the reduction.

Example 7.1

Let ϕ be a boolean formula in Conjunctive Normal Form (CNF) defined by:

$$\phi = (p_0 \vee \neg p_1 \vee p_2) \wedge (p_0 \vee \neg p_1 \vee \neg p_1).$$

We would like to construct, from ϕ , an instance of the \preceq_p -maximal basic problem, and in particular a deterministic automaton $\mathcal{T}_\phi = \langle X, x_0, X_m, \Sigma, \Delta \rangle$, in such a way that a particular state (named *true*) of \mathcal{T}_ϕ will be reachable in the computed controlled system if and only if ϕ is satisfiable. In our example, ϕ is composed of 2 clauses and 3 variables. The construction is the following:

- we create (i) for each variable p_i ($\forall i \in [0, 2]$), the states $p_i^0, p_i^1, \overline{p_i^1}, p_i^2, \overline{p_i^2}, st_i, st'_i$, (ii) for each clause c_i ($\forall i \in [1, 2]$), the states $\ell_{i,1}, \ell_{i,2}, \ell_{i,3}$ (i.e., one state for each literal of the clause), and (iii) four additional states $st, st', bad, true$ (see Figure 7.1).
- the observation space $\mathcal{D}_{Obs} = \{op_0, op_1, op_2, ost\}$. There is thus an observation op_i for each variable p_i of ϕ .
- the states $bad, true, st, st', st_i, st'_i$ ($\forall i \in [0, 2]$) are undistinguishable (i.e., $M^{-1}(ost) = \{st, st', st_0, st'_0, st_1, st'_1, st_2, st'_2, bad, true\}$) and, for each other state, the observation of its corresponding variable is observed (for example, $M^{-1}(op_0) = \{p_0^0, p_0^1, \overline{p_0^1}, p_0^2, \overline{p_0^2}, \ell_{1,1}, \ell_{2,1}\}$).
- there are four controllable actions $\top, \perp, \sigma^1, \sigma^2$ (see Figure 7.1).

The construction is mainly composed of three parts (see Figure 7.1):

- With the part 1, a solution of the \preceq_p -maximal basic problem must allow \top or \perp in op_i ($\forall i \in [0, 2]$). Indeed, otherwise the controller, which forbids no action in op_i ($\forall i \in [0, 2]$) and forbids σ^2 in ost , would be more permissive.
- With the part 2, a solution of the \preceq_p -maximal basic problem, which allows σ^1 and σ^2 in ost , must forbid \top or \perp in op_i ($\forall i \in [0, 2]$). Indeed, if for example \top and \perp are allowed in op_0 , then *bad* will be reachable from p_0^1 and $\overline{p_0^2}$ (through $\overline{p_0^1}$ and p_0^2) in the controlled system.

- Thus, a solution $\mathcal{C} = \langle \mathcal{S}, E \rangle$ of the \preceq_p -maximal basic problem, which enables σ^1 and σ^2 in ost , must forbid either \top or \perp in op_i ($\forall i \in [0, 2]$). We can then create a one-to-one mapping between the choices of the action to be forbidden between \top and \perp in each observation of $\{op_0, op_1, op_2\}$ (the function \mathcal{S}) and the valuations $val : \{p_0, p_1, p_2\} \rightarrow \mathbb{B}$ (which assign a value to the variables of ϕ). We define this relation as follows: for each $i \in [0, 2]$, \perp (resp. \top) $\in \mathcal{S}(op_i)$ and \top (resp. \perp) $\notin \mathcal{S}(op_i)$ if and only if $val(p_i) = \text{tt}$ (resp. ff). It means that \top is allowed in op_i if and only if the value of p_i is tt . So, the part 3 is constructed in such a way that if \mathcal{S} corresponds to a valuation $val \models \phi$ (resp. $val \not\models \phi$), then $true$ is reachable (resp. bad is reachable) from the state $\ell_{1,1}$ in the controlled system. Indeed, if the literal $\ell_{i,j}$ of a clause c_i is valuated to true, then from the state $\ell_{i,j}$ we can reach $\ell_{i+1,1}$ (or $true$ if c_i is the last clause) and if the literal $\ell_{i,j}$ is valuated to false, then from the state $\ell_{i,j}$ we can reach $\ell_{i,j+1}$ (or bad if $\ell_{i,j}$ is the last literal of the clause c_i). For example, (i) if val is such that $val(p_0) = val(p_1) = \text{ff}$ and $val(p_2) = \text{tt}$ (i.e., $val \models \phi$), then we traverse the states $\ell_{1,1}, \ell_{1,2}, \ell_{2,1}, \ell_{2,2}$ and $true$, and (ii) if val is such that $val(p_0) = \text{ff}$ and $val(p_1) = val(p_2) = \text{tt}$ (i.e., $val \not\models \phi$), then we traverse the states $\ell_{1,1}, \ell_{1,2}, \ell_{1,3}, \ell_{2,1}, \ell_{2,2}, \ell_{2,3}$ and bad . Note that, since bad cannot be reached in the \preceq_p -maximal basic problem, the part 3 (hence $true$) will not be reachable in the computed solution, if the formula is unsatisfiable.

Proposition 7.1

The \preceq_p -maximal basic problem cannot be solved in polynomial time.

Proof

The proof consists in a reduction from 3SAT to the \preceq_p -maximal basic problem. More precisely, we define an algorithm for 3SAT which uses the \preceq_p -maximal basic problem as subalgorithm; this proves the proposition, because 3SAT is NP-complete. This algorithm to decide 3SAT is defined as follows. From the formula ϕ , we build the instance of the \preceq_p -maximal basic problem (and in particular the system to be controlled \mathcal{T}_ϕ) defined in the polynomial transformation given below. Then, we get a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ from an algorithm solving the \preceq_p -maximal basic problem and we decide that ϕ is satisfiable if and only if a particular state $true$ of \mathcal{T}_ϕ is reachable in the controlled system $(\mathcal{T}_\phi)_{\mathcal{C}}$.

Polynomial Transformation from 3SAT to the \preceq_p -maximal Basic Problem. We consider a CNF formula ϕ over a set of variables $\mathcal{P} = \{p_0, \dots, p_k\}$: $\phi = \bigwedge_{m=1}^{n_c} c_m$, where $n_c > 0$ and the clauses $c_m = \bigvee_{j=1}^3 \ell_{m,j}$. Each $\ell_{m,j}$ is a variable in \mathcal{P} (positive literal) or the negation of a variable in \mathcal{P} (negative literal). The set of clauses of ϕ is denoted by Cl .

An instance of the \preceq_p -maximal basic problem is built from 3SAT as follows. First, we define the state space X of the system to be controlled, the observation space \mathcal{D}_{Obs} and the set of forbidden states Bad :

- The domain $X = \{bad, true, st, st'\} \cup \{st_i, st'_i \mid \forall i \in [0, k]\} \cup \{p_i^0, p_i^1, p_i^2, \overline{p_i^1}, \overline{p_i^2} \mid \forall p_i \in \mathcal{P}\} \cup \{\ell_{m,1}, \ell_{m,2}, \ell_{m,3} \mid \forall c_m \in \text{Cl}\}$.
- $\mathcal{D}_{Obs} = \{op_i \mid \forall p_i \in \mathcal{P}\} \cup \{ost\}$.
- $Bad = \{bad\}$.

In the sequel, we denote by $\mathcal{O}_{\mathcal{P}} = \{op_i \mid \forall i \in [0, k]\}$ the set of observations, ost excluded.

The partial observation is modeled by the observer $\langle \mathcal{D}_{Obs}, M \rangle$ where the mask M is defined as follows. The states $bad, true, st, st', st_i, st'_i$ ($\forall i \in [0, k]$) are undistinguishable and, for each other state, the observation of its corresponding variable is observed. Formally,

- $\forall op_i \in \mathcal{O}_{\mathcal{P}}, M^{-1}(op_i) = \{p_i^0, p_i^1, p_i^2, \overline{p_i^1}, \overline{p_i^2}\} \cup \{\ell_{m,j} \mid (m \in [1, n_c]) \wedge (j \in [1, 3]) \wedge ((\ell_{m,j} = p_i) \vee (\ell_{m,j} = \neg p_i))\}$
- $M^{-1}(ost) = \{st, st', bad, true\} \cup \{st_i, st'_i \mid \forall i \in [0, k]\}$

Finally, we define the deterministic automaton $\mathcal{T}_{\phi} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$ to be controlled. As explained above, the construction of \mathcal{T}_{ϕ} must be such that $true$ will be reachable in the controlled system obtained by solving the \preceq_p -maximal basic problem if and only if ϕ is satisfiable. Formally, \mathcal{T}_{ϕ} is defined by (i) the set X defined above, (ii) the initial state $x_0 = p_0^0$, (iii) the set of marked states $X_m = X$ (in fact, X_m can have any value), (iv) the set of actions $\Sigma = \Sigma_c \uplus \Sigma_{uc}$ with $\Sigma_c = \{\top, \perp, \sigma^1, \sigma^2\}$ and $\Sigma_{uc} = \emptyset$ (intuitively, a transition $\langle x_1, \top, x_2 \rangle$ represents a valuation $x_1 = \text{tt}$ and a transition $\langle x_1, \perp, x_2 \rangle$ represents a valuation $x_1 = \text{ff}$), and (v) the transition relation Δ defined as follows:

- $\forall i \in [0, k - 1] : \langle p_i^0, \top, p_{i+1}^0 \rangle, \langle p_i^0, \perp, p_{i+1}^0 \rangle \in \Delta$. Moreover, $\langle p_k^0, \top, st \rangle, \langle p_k^0, \perp, st \rangle \in \Delta$ (see part 1 of Figure 7.1).
- $\langle st, \sigma^1, st_0 \rangle \in \Delta$ and $\forall i \in [0, k] : \langle st_i, \sigma^1, st'_i \rangle, \langle st_i, \sigma^2, p_i^1 \rangle \in \Delta$. Moreover, $\forall i \in [0, k - 1] : \langle st'_i, \sigma^1, st_{i+1} \rangle, \langle st'_i, \sigma^2, p_i^2 \rangle \in \Delta$. We have also that $\langle st'_k, \sigma^2, p_k^2 \rangle \in \Delta$.
- $\forall i \in [0, k] : \langle p_i^1, \top, \overline{p_i^1} \rangle, \langle \overline{p_i^1}, \perp, bad \rangle, \langle \overline{p_i^1}, \top, st' \rangle, \langle \overline{p_i^2}, \perp, p_i^2 \rangle, \langle p_i^2, \top, bad \rangle, \langle p_i^2, \perp, st' \rangle \in \Delta$ (see part 2 of Figure 7.1).
- $\langle st', \sigma^1, \ell_{1,1} \rangle \in \Delta$.
- for the states of the clause c_m ($\forall m \in [1, n_c - 1]$), we define the following transitions:
 - if $\ell_{m,j}$ (for $j = 1, 2$) is a positive literal, $\langle \ell_{m,j}, \perp, \ell_{m,j+1} \rangle, \langle \ell_{m,j}, \top, \ell_{m+1,1} \rangle \in \Delta$. Otherwise, $\langle \ell_{m,j}, \top, \ell_{m,j+1} \rangle, \langle \ell_{m,j}, \perp, \ell_{m+1,1} \rangle \in \Delta$.
 - if $\ell_{m,3}$ is a positive literal, $\langle \ell_{m,3}, \perp, bad \rangle, \langle \ell_{m,3}, \top, \ell_{m+1,1} \rangle \in \Delta$. Otherwise, $\langle \ell_{m,3}, \top, bad \rangle, \langle \ell_{m,3}, \perp, \ell_{m+1,1} \rangle \in \Delta$.
- for the states of the clause c_{n_c} , we define the following transitions:
 - if $\ell_{n_c,j}$ (for $j = 1, 2$) is a positive literal, $\langle \ell_{n_c,j}, \perp, \ell_{n_c,j+1} \rangle, \langle \ell_{n_c,j}, \top, true \rangle \in \Delta$. Otherwise, $\langle \ell_{n_c,j}, \top, \ell_{n_c,j+1} \rangle, \langle \ell_{n_c,j}, \perp, true \rangle \in \Delta$.
 - if $\ell_{n_c,3}$ is a positive literal, $\langle \ell_{n_c,3}, \perp, bad \rangle, \langle \ell_{n_c,3}, \top, true \rangle \in \Delta$. Otherwise, $\langle \ell_{n_c,3}, \top, bad \rangle, \langle \ell_{n_c,3}, \perp, true \rangle \in \Delta$.

Correctness of the Reduction. We prove that ϕ is satisfiable if and only if the state *true* is reachable in the controlled system $(\mathcal{T}_\phi)_{/C}$, where C is a solution of the \preceq_p -maximal basic problem.

We first prove that a solution $C = \langle S, E \rangle$ of the \preceq_p -maximal basic problem is such that $p_0^0 \notin E$.

Lemma 7.1

The controllers $C = \langle S, E \rangle$, with $p_0^0 \in E$, are not a solution of the \preceq_p -maximal basic problem.

Proof

Reachable $_{\Sigma}^{(\mathcal{T}_\phi)_{/C}}((x_0)_{/C}) = \emptyset$, because the set of initial states of the controlled system $(\mathcal{T}_\phi)_{/C}$ is empty i.e., $(x_0)_{/C} = \emptyset$. Thus, C is a trivial controller.

In the sequel, we use the notation E_g to denote a subset of X that does not contain p_0^0 .

Then, we prove that a solution of the \preceq_p -maximal basic problem must allow \top or \perp in each observation of $\mathcal{O}_{\mathcal{P}}$.

Lemma 7.2

If $\mathcal{C} = \langle \mathcal{S}, E_g \rangle$ is a solution of the \preceq_p -maximal basic problem, then $\forall op_i \in \mathcal{O}_{\mathcal{P}} : (\top \notin \mathcal{S}(op_i)) \vee (\perp \notin \mathcal{S}(op_i))$.

Proof

Suppose there exists an observation $op_i \in \mathcal{O}_{\mathcal{P}}$ such that $\top, \perp \in \mathcal{S}(op_i)$. Then, $\text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}}) \subset \{p_0^0, p_1^0, \dots, p_k^0, st\}$. The controller, which only forbids σ^2 in ost , is non-trivial, valid and strictly more permissive than \mathcal{C} . But, it is a contradiction with the fact that \mathcal{C} is a solution of the \preceq_p -maximal basic problem.

This property implies that st is reachable in the system $(\mathcal{T}_{\phi})_{/\mathcal{C}}$ under the control of a solution of the \preceq_p -maximal basic problem.

Now, we prove that a solution of the \preceq_p -maximal basic problem must forbid \top or \perp in each observation of $\mathcal{O}_{\mathcal{P}}$ (if σ^1 and σ^2 are allowed in ost).

Lemma 7.3

If $\mathcal{C} = \langle \mathcal{S}, E_g \rangle$ is a solution of the \preceq_p -maximal basic problem and if $\sigma^1, \sigma^2 \notin \mathcal{S}(ost)$, then $\forall op_i \in \mathcal{O}_{\mathcal{P}} : (\top \in \mathcal{S}(op_i)) \vee (\perp \in \mathcal{S}(op_i))$.

Proof

Suppose that the observation $op_j \in \mathcal{O}_{\mathcal{P}}$ is such that $\top, \perp \notin \mathcal{S}(op_j)$. The states in the set $\{p_i^1, \overline{p_i^2} \mid \forall i \in [0, k]\}$ are reachable in $(\mathcal{T}_{\phi})_{/\mathcal{C}}$, because $\sigma^1, \sigma^2 \notin \mathcal{S}(ost)$ and by Lemma 7.2. In particular, p_j^1 and $\overline{p_j^2}$ are reachable. Since no action is forbidden in these states, the state bad is reachable from these states (through $\overline{p_j^1}$ and p_j^2). But, it is a contradiction with the fact that \mathcal{C} is valid.

Thus, a solution $\mathcal{C} = \langle \mathcal{S}, E \rangle$ of the \preceq_p -maximal basic problem, which enables σ^1 and σ^2 in ost , must forbid either \top or \perp in op_i ($\forall i \in [0, k]$). We can then create one-to-one mapping between the choices of the action to be forbidden among \top and \perp in each observation $op_i \in \mathcal{O}_{\mathcal{P}}$ (the supervisory function \mathcal{S}) and the valuations $val : \mathcal{P} \rightarrow \mathbb{B}$ (which assign a value to each variable of \mathcal{P}). We define this mapping as follows: for each $i \in [0, k]$, \perp (resp. \top) $\in \mathcal{S}(op_i)$ and \top (resp. \perp) $\notin \mathcal{S}(op_i)$ if and only if $val(p_i) = \text{tt}$ (resp. ff).

Lemma 7.4

Given a clause c_j of ϕ , a valuation $val : \mathcal{P} \rightarrow \mathbb{B}$ and a controller $\mathcal{C} = \langle \mathcal{S}, E_g \rangle$ such that its control policy regarding the actions \top and \perp in the observations of $\mathcal{O}_{\mathcal{P}}$ corresponds to the valuation val , if $\ell_{j,1}$ is reachable in $(\mathcal{T}_{\phi})_{/\mathcal{C}}$ and if $val \models c_j$, then $\ell_{j+1,1}$ (or *true* if $j = n_c$) is reachable in $(\mathcal{T}_{\phi})_{/\mathcal{C}}$.

Proof

This lemma is a direct consequence of the \mathcal{T}_{ϕ} 's construction and the control policy of \mathcal{C} .

Lemma 7.5

Given a clause c_j of ϕ , a valuation $val : \mathcal{P} \rightarrow \mathbb{B}$ and a controller $\mathcal{C} = \langle \mathcal{S}, E_g \rangle$ such that its control policy regarding the actions \top and \perp in the observations of $\mathcal{O}_{\mathcal{P}}$ corresponds to the valuation val , if $\ell_{j,1}$ is reachable in $(\mathcal{T}_{\phi})_{/\mathcal{C}}$ and if $val \not\models c_j$, then *bad* is reachable in $(\mathcal{T}_{\phi})_{/\mathcal{C}}$.

Proof

This lemma is a direct consequence of the \mathcal{T}_{ϕ} 's construction and the control policy of \mathcal{C} .

Lemma 7.6

Let $\mathcal{C} = \langle \mathcal{S}, E_g \rangle$ be a solution of the \preceq_p -maximal basic problem. If ϕ is not satisfiable, then $\sigma^1 \notin \mathcal{S}(ost)$ and $\sigma^2 \in \mathcal{S}(ost)$. Hence, *true* is not reachable in $(\mathcal{T}_{\phi})_{/\mathcal{C}}$.

Proof

Suppose that $\sigma^1 \in \mathcal{S}(ost)$, then \mathcal{C} is not maximal, because the controller, which only forbids σ^2 in *ost*, is strictly more permissive than \mathcal{C} .
 Now, suppose that $\sigma^2 \notin \mathcal{S}(ost)$. Then, by Lemmas 7.2 and 7.3, a solution of the \preceq_p -maximal basic problem must forbid either \top or \perp in op_i ($\forall op_i \in \mathcal{O}_{\mathcal{P}}$) and $\ell_{1,1}$ is reachable in $(\mathcal{T}_{\phi})_{/\mathcal{C}}$. Let $val : \mathcal{P} \rightarrow \mathbb{B}$ be the valuation which corresponds to the control policy of \mathcal{C} regarding the actions \top and \perp in the observations of $\mathcal{O}_{\mathcal{P}}$. Since, ϕ is not satisfiable, there exists at least one clause $c \in \text{Cl}$ such that $val \not\models c$. Let c_j be this first clause (then c_1, \dots, c_{j-1} are satisfied by val). Then, for each $1 \leq m < j$, $val \models c_m$ and, by Lemma 7.4, there exists a path between $\ell_{m,1}$ and $\ell_{m+1,1}$ in the controlled system $(\mathcal{T}_{\phi})_{/\mathcal{C}}$. But, since $val \not\models c_j$, the state *bad* is reachable from $\ell_{j,1}$ (by Lemma 7.5); it is a contradiction.

Lemma 7.7

Let $\mathcal{C} = \langle \mathcal{S}, E_g \rangle$ be a solution of the \preceq_p -maximal basic problem. If ϕ is satisfiable, then *true* is reachable in $(\mathcal{T}_\phi)_{/\mathcal{C}}$.

Proof

Let $\mathcal{C}_1 = \langle \mathcal{S}_1, E_g \rangle$ be a controller such that $\sigma^1, \sigma^2 \notin \mathcal{S}_1(ost)$ and its control policy regarding the actions \top and \perp in the observations of $\mathcal{O}_{\mathcal{P}}$ corresponds to a valuation $val' : \mathcal{P} \rightarrow \mathbb{B}$ satisfying ϕ . Such a valuation exists, since ϕ is satisfiable. Now, we prove that *true* is reachable in $(\mathcal{T}_\phi)_{/\mathcal{C}_1}$. The state $\ell_{1,1}$ is reachable, because $\sigma^1, \sigma^2 \notin \mathcal{S}_1(ost)$. Since $val' \models c_m$ ($1 \leq m < n_c$), $\ell_{m+1,1}$ is reachable from $\ell_{m,1}$ (by Lemma 7.4) and *bad* is not reachable from $\ell_{m,1}$. Moreover, since $val' \models c_{n_c}$, *true* is reachable from $\ell_{n_c,1}$ (by Lemma 7.4), and *bad* is not reachable from $\ell_{n_c,1}$. Clearly, \mathcal{C}_1 is more permissive than any valid and non-trivial controller $\mathcal{C}_2 = \langle \mathcal{S}_2, E_g \rangle$ such that $\sigma^1 \in \mathcal{S}_2(ost)$ or $\sigma^2 \in \mathcal{S}_2(ost)$. Thus, when ϕ is satisfiable, a solution $\mathcal{C}_3 = \langle \mathcal{S}_3, E_g \rangle$ of the \preceq_p -maximal basic problem is such that $\sigma^1, \sigma^2 \notin \mathcal{S}_3(ost)$. By Lemmas 7.2 and 7.3, \mathcal{C}_3 forbids either \top or \perp in op_i ($\forall op_i \in \mathcal{O}_{\mathcal{P}}$).

By the proof of Lemma 7.6, the valid and non-trivial controllers $\mathcal{C}_4 = \langle \mathcal{S}_4, E_g \rangle$, whose control policy regarding the actions \top and \perp in the observations of $\mathcal{O}_{\mathcal{P}}$ corresponds to a valuation which does not satisfy ϕ , are such that $\sigma^1 \in \mathcal{S}_4(ost)$ or $\sigma^2 \in \mathcal{S}_4(ost)$. Indeed, otherwise *bad* would be reachable in $(\mathcal{T}_\phi)_{/\mathcal{C}_4}$. In consequence, these controllers cannot be \preceq_p -maximal.

Thus, when ϕ is satisfiable, the solutions $\mathcal{C}_5 = \langle \mathcal{S}_5, E_g \rangle$ of the \preceq_p -maximal basic problem are such that $\sigma^1, \sigma^2 \notin \mathcal{S}_5(ost)$ and their control policy regarding the actions \top and \perp in the observations of $\mathcal{O}_{\mathcal{P}}$ corresponds to a valuation satisfying ϕ . As shown above, these controllers allow $(\mathcal{T}_\phi)_{/\mathcal{C}}$ to reach *true*.

In conclusion, by Lemmas 7.6 and 7.7, ϕ is satisfiable if and only if *true* is reachable in $(\mathcal{T}_\phi)_{/\mathcal{C}}$, where \mathcal{C} is a solution of the \preceq_p -maximal basic problem.

Proposition 7.2

The \preceq_p -maximal basic decision problem is co-NP-complete.

Proof

We prove that the complementary problem of the \preceq_p -maximal basic decision problem, which consists in deciding if there exists a valid and non-trivial controller \mathcal{C}' strictly more permissive than \mathcal{C} , is NP-complete.

For that, we first prove that the complementary of the \preceq_p -maximal basic decision problem is in NP. Let \mathcal{T} , $\langle \mathcal{D}_{Obs}, M \rangle$, Bad and \mathcal{C} be an instance of this problem, we select a controller $\mathcal{C}' = \langle \mathcal{S}', E' \rangle$, and determine if \mathcal{C}' is valid, non-trivial and strictly more permissive than \mathcal{C} . These properties can be checked in polynomial time. If \mathcal{C}' satisfies these properties, then we have found a valid and non-trivial controller strictly more permissive than \mathcal{C} . Therefore, this problem is in NP.

The second part of the proof consists in a reduction from 3SAT to the complementary of the \preceq_p -maximal basic decision problem.

Polynomial Transformation from 3SAT to the Complementary of the \preceq_p -maximal Basic Decision Problem. An instance of the this problem is built from 3SAT as follows. The system to be controlled \mathcal{T}_ϕ , the observer $\langle \mathcal{D}_{Obs}, M \rangle$ and the set Bad are built as in the proof of Proposition 7.1. The controller $\mathcal{C} = \langle \mathcal{S}, \emptyset \rangle$ is built as follows:

$$S(obs) = \begin{cases} \emptyset & \text{if } obs \in \mathcal{O}_p \\ \{\sigma^2\} & \text{if } obs = ost \end{cases}$$

Correctness of the Reduction. We prove that ϕ is satisfiable if and only if there exists a valid and non-trivial controller \mathcal{C}' strictly more permissive than \mathcal{C} . This equivalence is proven as follows:

- If ϕ is not satisfiable, then there exists no valid controller \mathcal{C}' strictly more permissive than \mathcal{C} . Indeed, by Lemma 7.6, if ϕ is not satisfiable, then a \preceq_p -maximal, valid and non-trivial controller allows σ^1 in ost and forbids σ^2 in ost . In consequence, \mathcal{C} is a \preceq_p -maximal, valid and non-trivial controller.
- If ϕ is satisfiable, then there exists a valid and non-trivial controller \mathcal{C}' strictly more permissive than \mathcal{C} . Indeed, the controller \mathcal{C}_1 , defined in the proof of Lemma 7.7, is valid, non-trivial, and strictly more permissive than \mathcal{C} .

Proposition 7.3

The \preceq_c -maximal basic decision problem is NP-complete.

Proof

First, we prove that \preceq_c -maximal basic decision problem is in NP. Let \mathcal{T} , $\langle \mathcal{D}_{Obs}, M \rangle$, Bad , and N be an instance of this problem, we select a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ and determine if \mathcal{C} is a valid and non-trivial controller which satisfies the property $|\text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{C}}((x_0)_{/\mathcal{C}})| \geq N$. These properties can be checked in polynomial time. If \mathcal{C} satisfies these properties, then it is a solution of the \preceq_c -maximal basic decision problem. Therefore, this problem is in NP.

The second part of the proof consists in a reduction from 3SAT to \preceq_c -maximal basic decision problem.

Polynomial Transformation from 3SAT to the \preceq_c -maximal Basic Decision Problem. An instance of the \preceq_c -maximal basic decision problem is built from 3SAT as follows. The system to be controlled \mathcal{T}_{ϕ} , the observer $\langle \mathcal{D}_{Obs}, M \rangle$ and the set Bad are built as in the proof of Proposition 7.1 and we set N to $3.k + 5$.

Correctness of the Reduction. We prove that ϕ is satisfiable if and only if there exists a valid and non-trivial controller \mathcal{C} such that $|\text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}})| \geq 3.k + 5$. This equivalence is proven as follows:

- If ϕ is not satisfiable, then there exists no valid and non-trivial controller \mathcal{C} such that $|\text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}})| \geq 3.k + 5$. Indeed, by Lemma 7.6, if ϕ is not satisfiable, then a \preceq_p -maximal, valid and non-trivial controller $\mathcal{C}_1 = \langle \mathcal{S}_1, E_1 \rangle$ allows σ^1 in ost and forbids σ^2 in ost . The set of reachable states in $(\mathcal{T}_{\phi})_{/\mathcal{C}_1}$ is given by $\{p_i^0 \mid \forall i \in [0, k]\} \cup \{st\} \cup \{st_i, st'_i \mid \forall i \in [0, k]\}$ i.e., $3.k + 1$ states are reachable. All controllers, that allow the system $(\mathcal{T}_{\phi})_{/\mathcal{C}}$ to reach strictly more than $3.k + 1$ states, must allow σ_1 in ost , σ_2 in ost , and at least one action between \top and \perp in op_i ($\forall i \in [0, k]$). These controllers are non-trivial and strictly more permissive than \mathcal{C}_1 . But, they are not valid; otherwise, we would have a \preceq_p -maximal, valid and non-trivial controller, which allows σ^1 and σ^2 in ost , and hence a contradiction with Lemma 7.6. In consequence, we have proved that there exists no valid and non-trivial controller \mathcal{C} such that $|\text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}})| > 3.k + 1$.

- If ϕ is satisfiable, then there exists a valid and non-trivial controller \mathcal{C} such that $|\text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}})| \geq 3.k + 5$. Indeed, the controller \mathcal{C}_1 , defined in the proof of Lemma 7.7, is valid and non-trivial and satisfies the property $|\text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}})| \geq 3.k + 5$.

Proposition 7.4

The interval basic problem is NP-complete.

Proof

First, we prove that the interval basic problem is in NP. Let \mathcal{T} , $\langle \mathcal{D}_{Obs}, M \rangle$, Min and Max be an instance of this problem, we select a controller $\mathcal{C} = \langle \mathcal{S}, E \rangle$ and determine if $Min \subseteq \text{Reachable}_{\Sigma}^{\mathcal{T}/\mathcal{C}}((x_0)_{/\mathcal{C}}) \subseteq Max$. This property can be checked in polynomial time. If \mathcal{C} satisfies this property, then it is a solution of the interval basic problem. Therefore, this problem is in NP.

The second part of the proof consists in a reduction from 3SAT to the interval basic problem.

Polynomial Transformation from 3SAT to the Interval Basic Problem. An instance of the this problem is built from 3SAT as follows. The system to be controlled \mathcal{T}_{ϕ} and the observer $\langle \mathcal{D}_{Obs}, M \rangle$ are built as in the proof of Proposition 7.1. We set Min to $\{true\}$ and Max to $\overline{\{bad\}}$.

Correctness of the Reduction. We prove that ϕ is satisfiable if and only if there exists a controller \mathcal{C} such that $Min \subseteq \text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}}) \subseteq Max$. This equivalence is proven as follows:

- If ϕ is not satisfiable, then there exists no controller \mathcal{C} such that $Min \subseteq \text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}}) \subseteq Max$. Indeed, a controller \mathcal{C} solving the interval basic problem must prevent the system \mathcal{T}_{ϕ} from reaching bad ; otherwise, we would have $\text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}}) \not\subseteq Max$. By Lemma 7.6, we know that if ϕ is not satisfiable, then all \preceq_p -maximal controllers, that prevent the system \mathcal{T}_{ϕ} from reaching bad , do not allow \mathcal{T}_{ϕ} to reach the state $true$. Thus, all controllers \mathcal{C}_1 , that prevent \mathcal{T}_{ϕ} from reaching bad , do not satisfy the property $Min \subseteq \text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}})$. Therefore, there exists no controller \mathcal{C} such that $Min \subseteq \text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}}) \subseteq Max$.

- If ϕ is satisfiable, then there exists a controller \mathcal{C} such that $Min \subseteq \text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}}((x_0)_{/\mathcal{C}}) \subseteq Max$. Indeed, the controller \mathcal{C}_1 , defined in the proof of Lemma 7.7, allows the system \mathcal{T}_{ϕ} to reach *true* and prevents it from reaching *bad*. This controller thus satisfies the property $Min \subseteq \text{Reachable}_{\Sigma}^{(\mathcal{T}_{\phi})/\mathcal{C}_1}((x_0)_{/\mathcal{C}_1}) \subseteq Max$.

7.4.2 The Non-blocking and Deadlock Free Cases

Each complexity result, given in section 7.4.1, can be extended to the case where the non-blocking (resp. deadlock free) property must be ensured by using a reduction similar to the one defined in the proof of Proposition 3.7 (resp. Proposition 3.6).

However, for the \preceq_p -maximal basic problem, we differently proceed to prove a more general result. More precisely, we prove that, for any comparison criterion \preceq , the \preceq -maximal non-blocking (resp. deadlock free) problem, which consists in computing a \preceq -maximal, valid and non-trivial controller ensuring the non-blocking (resp. deadlock free) property, cannot be solved in polynomial time. For that, we prove that the non-blocking (resp. deadlock free) problem, which consists in deciding if there exists a valid and non-trivial controller ensuring the non-blocking (resp. deadlock free) property, cannot be solved in polynomial time⁵.

Proposition 7.5

Given a deterministic automaton $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$, an observer $\langle \mathcal{D}_{Obs}, M \rangle$, and a predicate *Bad*, which represents a set of forbidden states, the non-blocking problem, which consists in deciding if there exists a valid and non-trivial controller ensuring the non-blocking property, is NP-complete.

Proof

We start by showing that the non-blocking problem is in NP. For that, it is sufficient to see that if we select a controller \mathcal{C} , then we can check in polynomial time that it is valid and non-trivial, and that $\mathcal{T}_{/\mathcal{C}}$ is non-blocking.

⁵Such an approach cannot be used for the basic case, because the problem, which consists in deciding if there exists a valid and non-trivial controller, can be solved in polynomial time.

The second part of the proof consists in a reduction from 3SAT to the non-blocking problem. This reduction is very similar to the one defined in the proof of Proposition 7.1 and it consists in building an instance of the non-blocking problem from 3SAT in such a way that this instance (of the non-blocking problem) is positive (i.e., its answer is *yes*) if and only if ϕ is satisfiable.

Polynomial Transformation from 3SAT to the Non-blocking Problem. The polynomial reduction is similar to the one defined in the proof of Proposition 7.1 except that the actions σ^1 and σ^2 are uncontrollable and that $X_m = \{true\} \cup \{p_i^1, \overline{p_i^2} \mid \forall i \in [0, k]\}$.

Correctness of the Reduction. We prove that ϕ is satisfiable if and only if the built instance of the non-blocking problem is a positive instance (i.e., its answer is *yes*).

We can first remark that Lemma 7.1 remains valid for the non-blocking problem. Again, we will use the notation E_g to denote a subset of X that does not contain p_0^0 .

Then, we prove that a valid and non-trivial controller ensuring the non-blocking property must allow \top or \perp in each observation of $\mathcal{O}_{\mathcal{P}}$.

Lemma 7.8

If $\mathcal{C} = \langle \mathcal{S}, E_g \rangle$ is a valid and non-trivial controller ensuring the non-blocking property, then $\forall op_i \in \mathcal{O}_{\mathcal{P}} : (\top \notin \mathcal{S}(op_i)) \vee (\perp \notin \mathcal{S}(op_i))$.

Proof

Suppose that there exists an observation $op_i \in \mathcal{O}_{\mathcal{P}}$ such that $\top, \perp \in \mathcal{S}(op_i)$, then the controlled system $(\mathcal{T}_{\phi})_{/\mathcal{C}}$ is blocking, because the states of X_m are not reachable.

This property implies that st is reachable in the system \mathcal{T}_{ϕ} under the control of a valid and non-trivial controller ensuring the non-blocking property.

Now, we prove that a valid and non-trivial controller ensuring the non-blocking property must forbid \top or \perp in each observation of $\mathcal{O}_{\mathcal{P}}$ (we recall that σ^1 and σ^2 are allowed, because they are uncontrollable).

Lemma 7.9

If $\mathcal{C} = \langle \mathcal{S}, E_g \rangle$ is a valid and non-trivial controller ensuring the non-blocking property, then $\forall op_i \in \mathcal{O}_{\mathcal{P}} : (\top \in \mathcal{S}(op_i)) \vee (\perp \in \mathcal{S}(op_i))$.

Proof

The proof is similar to the one given in Lemma 7.3.

By Lemmas 7.8 and 7.9, any valid and non-trivial controller \mathcal{C} ensuring the non-blocking property forbids, in each op_i ($\forall i \in [0, k]$), either \top or \perp . We can therefore create a one-to-one mapping between the choices of the action to be forbidden between \top and \perp in each observation $op_i \in \mathcal{O}_{\mathcal{P}}$ (the function \mathcal{S}) and the valuations $val : \mathcal{P} \rightarrow \mathbb{B}$ (which assign a value to the variables of ϕ). We define this mapping as follows: for each $i \in [0, k]$, \perp (resp. \top) $\in \mathcal{S}(op_i)$ and \top (resp. \perp) $\notin \mathcal{S}(op_i)$ if and only if $val(p_i) = \text{tt}$ (resp. ff).

Lemmas 7.4 and 7.5 remain valid for the non-blocking problem and we can then prove the two following lemmas.

Lemma 7.10

If ϕ is not satisfiable, then the built instance of the non-blocking problem is a negative instance (i.e., its answer is *no*).

Proof

Let us suppose that $\mathcal{C} = \langle \mathcal{S}, E_g \rangle$ is a valid and non-trivial controller ensuring the non-blocking property. By Lemmas 7.8 and 7.9, the controller \mathcal{C} must forbid either \top or \perp in each $op_i \in \mathcal{O}_{\mathcal{P}}$ and $\ell_{1,1}$ is reachable in $(\mathcal{T}_{\phi})_{/\mathcal{C}}$. Let $val : \mathcal{P} \rightarrow \mathbb{B}$ be the valuation which corresponds to the control policy of \mathcal{C} regarding the actions \top and \perp in the observations of $\mathcal{O}_{\mathcal{P}}$. Since ϕ is not satisfiable, there exists at least one clause $c \in \text{Cl}$ such that $val \not\models c$. Let c_j be this first clause (then c_1, \dots, c_{j-1} are satisfied by val). Thus, by Lemmas 7.4 and 7.5, the state $\ell_{j,1}$ is reachable from $\ell_{1,1}$ and the state *bad* is reachable from $\ell_{j,1}$; hence, it is a contradiction. In consequence, the built instance of the non-blocking problem is a negative instance.

Lemma 7.11

If ϕ is satisfiable, then the built instance of the non-blocking problem is a positive instance (i.e., its answer is *yes*).

Proof

Let $\mathcal{C} = \langle \mathcal{S}, E_g \rangle$ be a controller such that its control policy regarding the actions \top and \perp in the observations of $\mathcal{O}_{\mathcal{P}}$ corresponds to a valuation $val : \mathcal{P} \rightarrow \mathbb{B}$ satisfying ϕ . The state $\ell_{1,1}$ is reachable from the initial state p_0^0 in $(\mathcal{T}_{\phi})_{/\mathcal{C}}$, because \mathcal{C} allows either \top or \perp in each $op_i \in \mathcal{O}_{\mathcal{P}}$. Since each clause c of ϕ is satisfied by val , the state *true* is reachable from $\ell_{1,1}$ and the state *bad* is not reachable (by Lemmas 7.4 and 7.5). Moreover, the system $(\mathcal{T}_{\phi})_{/\mathcal{C}}$ is non-blocking and hence \mathcal{C} is a valid and non-trivial controller ensuring the non-blocking property.

This proposition implies that, for any comparison criterion \preceq , the \preceq -maximal non-blocking problem, which consists in computing a \preceq -maximal, valid and non-trivial controller ensuring the non-blocking property, cannot be solved in polynomial time.

Corollary 7.1

The \preceq -maximal non-blocking problem cannot be solved in polynomial time.

Proposition 7.6

Given a deterministic automaton $\mathcal{T} = \langle X, x_0, X_m, \Sigma, \Delta \rangle$, an observer $\langle \mathcal{D}_{Obs}, M \rangle$, and a predicate Bad , which represents a set of forbidden states, the deadlock free problem, which consists in deciding if there exists a valid and non-trivial controller ensuring the deadlock free property, is NP-complete.

Proof (Sketch)

We start by showing that the deadlock free problem is in NP. For that, it is sufficient to see that if we select a controller \mathcal{C} , then we can check in polynomial time that it is valid and non-trivial, and that \mathcal{T}/\mathcal{C} is deadlock free.

The second part of the proof consists in a reduction from 3SAT to the deadlock free problem. This reduction is similar to the one given in the proof of Proposition 7.5, except that we add an uncontrollable self-loop to the states that were marked in this reduction. In this way, whenever ϕ is satisfiable, a deadlock free controlled system can be computed.

This proposition implies that, for any comparison criterion \preceq , the \preceq -maximal deadlock free problem, which consists in computing a \preceq -maximal, valid and non-trivial controller ensuring the deadlock free property, cannot be solved in polynomial time.

Corollary 7.2

The \preceq -maximal deadlock free problem cannot be solved in polynomial time.

Remark 7.1

Since $NP \subseteq PSPACE$, the space complexity of the problems considered in this chapter is PSPACE.

7.5 Discussions

In this section, we have studied the complexity of several control problems. In particular, we have shown that a \preceq_p -maximal memoryless controller with partial observation solving the state avoidance control problem cannot be computed in polynomial time. A way to overcome this negative result is to define a method computing approximate solutions. In chapter 3, we have already explored this approach by defining an algorithm that uses abstract interpretation techniques to compute an approximate solution for the state avoidance control problem.

Conclusion



IN this chapter, we briefly recall the works, presented in this thesis, on the control of infinite and finite state systems and we propose several extensions for future research works.

Summary

In the introduction of this thesis, we explained why the development of validation methods for computer systems is an important issue. This motivated our initial goal of providing *supervisory control methods*.

A lot of works in this domain assume that the system to be controlled has a finite state space. In this case, the termination of the algorithms is generally not a challenging issue, since the problems to be solved are usually decidable. In the case of infinite state systems, the control problems are generally *undecidable*, and the termination is thus an important issue. Most of the works overcome this obstacle by restricting their attention to particular cases where the problems are *decidable* (see works on Petri nets, timed automata, . . .). In this thesis, we were interested in the state avoidance control problem for infinite state systems and we followed a different approach to define effective algorithms for this undecidable problem: we used abstract interpretation techniques to ensure, at the price of some overapproximations, the termination of our control algorithm.

We started our work by developing a *centralized* control method, which synthesizes a *single memoryless* controller with partial observation, that restricts the behavior of the system to ensure the desired properties. Our algorithm uses abstract interpretation techniques to ensure the termination of the computations. Since, these techniques may overapproximate some computations, we implemented our method in our tool SMACS to evaluate it experimentally. Our experiments were conclusive, since they shown that the controllers synthesized by SMACS have a good permissiveness. The implementation of SMACS allows us to use several abstract lattices and we evaluate the performance (time and memory) of our tool by using these different lattices. A web page allows people

interested by SMACS to get information on this tool, use it online and download its source code.

Next, we studied methods to improve the quality of the synthesized controllers and we proposed two different solutions: the *k-memory* controllers and the *online* controllers. These controllers record a part or the whole of the execution of the system and use this information to define their control policy. Moreover, we theoretically compared our different controllers and these comparisons shown that the online controllers define the best control policy and that the *k-memory* controllers are better than the memoryless controllers.

However, in some cases, the nature of the system to be controlled makes unrealistic the use of a centralized framework; for example, in the case of distributed systems. We then turned our attention to the control of this kind of systems. We started our work on this topic by developing a *decentralized* control method, which can be used for the control of distributed systems with synchronous communications. This method consists in synthesizing several local controllers that are coordinated to jointly ensure the desired properties. In this framework, we assume that we have no information regarding the structure of the distributed system to be controlled while the computation of the controllers; this implies that the computations are performed on the model of the global system. Next, we developed a *modular* method for the cases where we know this structure. This method exploits the structure of the distributed system to be controlled to perform the computations locally i.e., on each subsystem which composes the global system. We have implemented these two methods and our experiments shown that the performance (time and memory) of the modular method (tool Mod-SMACS) is better than the one of the decentralized method (tool SMACS).

Unfortunately, our decentralized and modular methods cannot be used for the control of an interesting class of distributed system: the distributed systems with asynchronous communications. Indeed, in this case, the synchronization mechanism used by both methods to define the global control policy is no more valid, because of the asynchronous nature of the communications of these systems. Therefore, we tackled this problem and we defined a distributed control method. In this method, each subsystem of the global system is controlled by a local controller. These controllers can exchange information through the existing asynchronous communications. We shown how they can exploit this information to have a better knowledge of the current state of the global system and hence to define a better control policy.

Finally, we restricted our attention to the control of finite state systems with

partial observation and we studied the time complexity of several control problems related to this topic. In particular, we proved that the problem, which consists in computing a maximal controller with partial observation for the state avoidance control problem, cannot be solved in polynomial time unless $P = NP$.

Future Works

Theoretical Research. In chapter 5, we studied the decentralized control of infinite state systems. In our framework, the local controllers cannot exchange information and they only propose unconditional control decisions i.e., they propose either to forbid or to allow an action. In chapter 2, we saw that there exist, in the event-based approach, more elaborate decentralized frameworks: some frameworks allow the controllers to communicate (e.g., [BL00, Tri02]) and other ones allow the local controllers to propose conditional control decisions (e.g., [Ric03, YL04]) i.e., the local controllers can propose control decisions that depend on the decisions of the other controllers. An interesting line of research would be to try to extend our decentralized framework with these approaches and to determine whether the permissiveness of the computed solutions is improved.

We want to solve the open question, given in section 5.2.2, which consists in comparing the permissiveness of the decentralized and modular approaches.

In our distributed method (see chapter 6), the local controllers exchange information through FIFO channels in order to refine their control policy. The size of these information can be big and an interesting issue would then be to minimize these communications in order to avoid the *saturation* of the queues.

In chapter 7, we restrict our attention to finite state systems and we determine the time complexity of several control problems. In these problems, the controllers are memoryless and have a partial observation of the system. A natural extension of this research would be to determine the time complexity of these problems when we must synthesize k -memory and online controllers.

Implementation. SMACS (and Mod-SMACS) allows the user to use three different abstract lattices: the lattice of intervals, the lattice of octagons and the lattice of convex polyhedra. With these lattices a set of states is represented either by an interval, or by an octagon, or by a convex polyhedron. To improve the quality of the abstractions, we could represent a set of states by a finite union of convex polyhedra (resp. of octagons or of intervals). The main difficulty to implement this approach is to define a *good* widening operator. Indeed, our

attempts of implementing our algorithms with the lattice of finite unions of convex polyhedra (resp. of octagons or of intervals) fail for the moment, because of the widening operator which gives rough results. The used widening operator can be found in [Bou92].

In chapter 6, we proposed an effective algorithm for the distributed control of FIFO systems. As future works, we want to implement this algorithm to experimentally evaluate it.

In this thesis, we tried to provide the most complete contribution to the control of infinite state systems based on abstract interpretation. However, much works remain to be done in this domain and will surely be investigated in the future.

Appendix

A

SMACS

 IN the previous chapters, we have presented several control algorithms which overapproximate some computations to ensure that they always terminate. To experimentally evaluate this approach based on the computation of approximations, we have implemented some of these algorithms in our tool SMACS. We describe, in this chapter, our implementation. SMACS is written in *Objective CAML* [OCa09]. In the current version of SMACS (version 1.0), we have used the *APRON libraries* [JM09]. APRON provides several libraries which implement numerical abstract domains like intervals [CC77], octagons [Min01], and convex polyhedra [CH78]. These libraries have the same interface, which makes easy the switch between them in our tool. We tried to make the interface as modular as possible. So, our tool does not specifically need the APRON libraries: any abstract domain with an Objective CAML library can replace APRON.

In the remainder of this chapter, we give a description of our tool (section A.1), and we define its input language (section A.2) and the outputs that it can generate (section A.3).

A.1 Description of SMACS

We have actually two different implementations: SMACS (Symbolic Masked Controller Synthesis) and Mod-SMACS (Modular Symbolic Masked Controller

Synthesis). The first tool implements the algorithms, defined in chapter 3 and in section 5.1, which respectively synthesize centralized and decentralized controllers. The second one implements the algorithm defined in section 5.2, which synthesizes modular controllers. These tools have a lot of similarities and we will only provide a description of SMACS.

The SMACS tool is written in Objective CAML and is available online [KL10]. [KL10] provides the source codes of SMACS and Mod-SMACS, and it also allows the user to use SMACS online.

Libraries Used by SMACS. SMACS needs several libraries:

- GMP [GMP]: a library for arbitrary precision arithmetic, which operates on signed integers, rational numbers, and floating-point numbers.
- MPFR [MPF]: a library for multiple-precision floating-point computations.
- CamlIDL [Cama]: it allows us to use C libraries from OCaml.
- MLGMPIDL [MLG]: it offers an interface to the GMP and MPFR libraries for OCaml.
- Camllib [Camb]: a library implementing various datatypes.
- Fixpoint [Fix09]: a generic fixpoint calculator.
- APRON [JM09]: it provides several libraries which implement numerical abstract domains like intervals [CC77], octagons [Min01], and convex polyhedra [CH78]. In the current version of SMACS, we can only choose one of the abstract domains of APRON as abstract lattice (we can thus use integer and floating-point variables). However, we have tried to make independent our implementation from APRON by adding an interface between SMACS and APRON. So, we only use the APRON libraries in the file *sts_manager.ml* and if we want to use the abstract domain of another library, we just have to rewrite this file.

Overview of SMACS. SMACS computes a centralized or a decentralized controller. Since the centralized control is a particular case of the decentralized control, the user can choose the centralized control method by specifying only one local controller in the input of SMACS. The input of SMACS is a textual description of the system to be controlled. The variables of this system can be

either integer or real. The guards of the symbolic transitions are boolean combinations of linear constraints and the assignments are given by linear expressions. Moreover, the set of forbidden states is specified by a combination of linear constraints. Four kinds of masks are available to model the partial observation of each local controller (a description of these different kinds of masks is given below). SMACS then computes a solution from this input and describes it. More precisely, it displays the control function \mathcal{F}_i of each local controller \mathcal{C}_i and it graphically displays the controlled system (when it can be computed). In the next sections, we detail the input and the output of SMACS.

A.2 Input Language of SMACS

Grammar of the Input Language. The input language of SMACS is a textual description of the state avoidance control problem that SMACS must solve. This language describes:

- the system to be controlled
- the set of forbidden states
- the partial observation (mask) of each local controller \mathcal{C}_i . When a local controller has a full observation of the system, the user specifies no mask for this controller. To obtain a centralized controller, the user just has to specify one local controller.

More formally, the input language is defined by the following grammar where $\langle \rangle$ denotes a non-terminal symbol, the brackets $[]$ (in italic) denote an optional element (when the brackets are not in italic, they denote a word of the input language), ϵ denotes the empty symbol, and the bold words are the key words of the language.

```

<automaton> ::= automaton <string> :
               <event_declaration>
               <variable_declaration>
               <initial_locations>
               <location_list_with_trans>
               <bad_states_declaration>
               <controllers_list>

```

The system to be controlled is identified by a string and is defined by a

list of events, a list of variables (with the initial values), a list of initial locations and a list of locations with their outgoing transitions. Next, the set of forbidden states and the partial observation of the local controllers are specified.

```

<event_declaration> ::= events: <event_list> ;
<event_list> ::= <event_list> ; <event> | <event>
<event> ::= C <integer_list> <string> | U <string>
<integer_list> ::= <integer_list> <integer> | <integer>

```

An action (or event) is identified by a string and is preceded either by the letter *U* or by the letter *C*. The symbol *U* (resp. *C*) means that the action is uncontrollable (resp. controllable). For a controllable action, the user can specify a list of integers, which gives the identifiers of the controllers that control this action; if no identifier is specified, the action can be controlled by all controllers.

Example A.1

Let us consider the following instructions:

```

events:
  U Cons;
  C 1 3 StopProd;

```

The second instruction declares the uncontrollable action *Cons* and the third instruction declares the action *StopProd*, which can be controlled by the local controllers 1 and 3.

```

<variable_declaration> ::= <variables_list> ; | ;
<variables_list> ::= <variables_list> ; <variable> | <variable>
<variable> ::= int <string> [= <integer> ]
               | float <string> [= <float> ]

```

The list of events is followed by the list of variables of the system. A variable is either integer or floating-point and the user can specify the initial value of this variable. If no value is specified, then this variable can have any value at the beginning of the execution of the system.

Example A.2

Let us consider the following instruction:

```
int x = 0;
```

It declares an integer variable x . The initial value of this variable is 0.

```
<initial_locations> ::= initial : <location_list>
<location_list> ::= <location_list> , <string> | <string>
```

The user must specify the set of initial locations. Each location is identified by a string.

```
<location_list_with_trans> ::= <location_list_with_trans>
                               <location_with_trans>
                               | <location_with_trans>
<location_with_trans> ::= state <string> : <transition_list>
```

The user must specify the list of locations of the system. Each declaration of a location is followed by the list of outgoing transitions of this location.

```
<transition_list> ::= <transition_list> <transition> ; | ε
<transition> ::= to <string> : when <bexpr> ,
                <string> [, <assign_trans>]
<bexpr> ::= true | false | ( <bexpr> )
           | <bexpr> and <bexpr>
           | <bexpr> or <bexpr> | not <bexpr>
           | <iexpr> <rel> <iexpr>
<rel> ::= == | != | < | <= | > | >=
<iexpr> ::= <float> | <integer> | <string> |
            ( <iexpr> ) | <iexpr> + <iexpr>
            | <iexpr> - <iexpr> | <iexpr> * <iexpr>
            | <iexpr> / <iexpr> | - <iexpr>
<assign_trans> ::= with <assign_list> |
<assign_list> ::= <assign_list> , <assign> | <assign>
<assign> ::= <string> = <iexpr>
```

The description of a transition consists in specifying (*i*) the location, which is reached after this transition, (*ii*) the guard of this transition, (*iii*) the action labeling this transition, and (*iv*) the assignments of the variables. The guards

are boolean combinations of linear constraints and the assignments are given by linear expressions¹.

Example A.3

Let us consider the following instructions:

```
state loc1 :
  to loc2 : when x < 2 , Choice with x = x - 1;
```

These instructions mean that the location *loc1* has an outgoing transition leading to the location *loc2*. This transition is labeled by the action *Choice*. Moreover, its guard is defined by the inequality $x < 2$ and its assignment function is defined by the assignment $x = x - 1$.

```
<bad_states_declaration> ::= bad_states : <bad_states_list>
<bad_states_list> ::= <bad_states_list> <bad_states>
                       | <bad_states>
<bad_states> ::= state <string> : <bexpr>
```

The user can define, in each location of the system, a boolean combination of linear constraints specifying the bad states.

Example A.4

Let us consider the following instructions:

```
bad_states:
  state loc1 : x > 10 and y > 0
```

A state is bad if its location is *loc1*, the value of x is greater than 10 and the value of y is greater than 0.

```
<controller_list> ::= <controller_list> <controller> | <controller>
<controller> ::= controller <integer> mask : <mask>
<mask> ::= same_localities : <same_localities_list>
           | hidden_variables : <hidden_variables_list>
           | intervals : <same_intervals_list>
           | u_variables : <u_variables_list> |  $\epsilon$ 
```

¹This choice is due to the use of lattices of intervals, octagons and convex polyhedra which work well when the guards are linear constraints and the assignments are also linear.

The user is able to define the partial observation of each local controller (a local controller is identified by an integer) and can use, for that, one of the four available kinds of masks. These kinds of masks are described below. If the user specifies no mask for a local controller, then this controller perfectly observes the system.

```

<same_localities_list> ::= <same_localities_list> , [ <localities> ]
                        | [ <localities> ]
<localities> ::= <localities> , <string> | <string>
<hidden_variables_list> ::= <hidden_variables_list> , <string>
                        | <string>
<same_intervals_list> ::= <same_intervals_list> <same_intervals>
                        | <same_intervals>
<same_intervals> ::= ( <string> , [ <plus_minus> <float> ,
                        <plus_minus> <float> ] )
<plus_minus> ::= ε | -
<u_variables_list> ::= <u_variables_list> <u_variables>
                        | <u_variables>
<u_variables> ::= ( <string> , <float> , <float> )

```

The first kind of masks allows the user to specify lists of locations that the controller cannot distinguish. The second kind of masks allows the user to specify variables that the controller does not observe. The controller cannot thus determine the value of these variables. The third kind of masks allows the user to specify some variables whose value is unknown whenever it belongs to a given interval. For that, the user specifies, for each of these variables, the identifier of this variable and the interval in which the controller cannot observe it. The fourth kind of masks allows the user to specify an imprecision regarding the value of some variables. More precisely, the user specifies a list of elements composed of:

- the identifier of a variable v_i
- two numbers c_i and c'_i .

So, when the controller observes this variable v_i , it receives a value which belongs to the interval $[\vec{v}_i - c_i, \vec{v}_i + c'_i]$, where \vec{v}_i is the current value of v_i . One can note that the first three masks correspond to a partition of the state space,

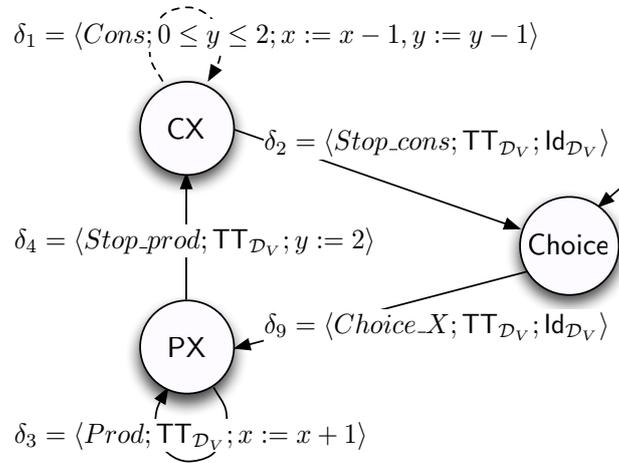


Figure A.1 - Producer and consumer example.

whereas the last one corresponds to a covering.

Example A.5

Let us consider the following instructions:

```

controller 1 mask: hidden_variables: x
controller 2 mask:
controller 3 mask: same_localities: [loc1, loc2]
controller 4 mask: intervals: (x, [5.0,10.0])
controller 5 mask: u_variables: (x, 1.0, 2.0)

```

The first instruction specifies a local controller 1 which does not observe the variable x and the second instruction declares a local controller 2 which perfectly observes the system. The local controller 3 does not distinguish the locations $loc1$ and $loc2$. The local controller 4 does not observe the variable x when its value belongs to the interval $[5.0, 10.0]$. However, when it does not belong to this interval, the controller 4 perfectly observes x . The local controller 5 does not receive the exact value of the variable x : if x_1 is the current value of x , it receives a value which belongs to $[x_1 - 1, x_1 + 2]$.

Now, we illustrate this grammar with a complete example.

Example A.6

Let us consider the system \mathcal{T} depicted in Figure A.1. This system is a simplified version of the producer and consumer example given in Figure 3.1. It has a tuple $V = \langle \ell, x, y \rangle$ of three variables, where (i) $\ell \in \{\text{CX}, \text{PX}, \text{Choice}\}$ gives the current location of the system, (ii) x gives the number of parts X and (iii) y gives the number of parts X that can be produced. The initial state is $\langle \text{Choice}, 0, 0 \rangle$ and the set $Bad = \{\langle \text{CX}, x, y \rangle \mid (x \leq 10) \wedge (y \in [0, 2])\}$. The system \mathcal{T} is controlled by the decentralized controller $\mathcal{C}_d = \langle \langle \mathcal{C}_1, \mathcal{C}_2 \rangle, \mathcal{R} \rangle$. The controller \mathcal{C}_1 has a partial observation modeled by the mask $M_1 : Loc \times \mathbb{Z}^2 \rightarrow 2^{Loc \times \mathbb{Z}^2}$ where, for each state $\vec{v} = \langle \ell, x, y \rangle \in \mathcal{D}_V$, the set of states, that are undistinguishable from \vec{v} , is given by $\{\langle \ell, x_1, y_1 \rangle \mid x_1, y_1 \in \mathbb{Z}\}$ (i.e., \mathcal{C}_1 does not observe the variables x and y). \mathcal{C}_1 does not control the action $Cons$ (i.e., $\Sigma_{1,uc} = \{Cons\}$). The controller \mathcal{C}_2 has a partial observation modeled by the mask $M_2 : Loc \times \mathbb{Z}^2 \rightarrow 2^{Loc \times \mathbb{Z}^2}$ defined as follows: for each state $\vec{v} = \langle \ell, x, y \rangle \in \mathcal{D}_V$, if $x \notin [5, 15]$, then \vec{v} is perfectly observed, otherwise the set of states, that are undistinguishable from \vec{v} , is given by $\{\langle \ell, x_1, y \rangle \mid x_1 \in [5, 15]\}$ (i.e., \mathcal{C}_2 does not observe the variable x when its value belongs to the interval $[5, 15]$). \mathcal{C}_2 does not control the actions $Cons$ and $Stop_prod$ (i.e., $\Sigma_{2,uc} = \{Cons, Stop_prod\}$). The input language corresponding to this system is the following:

```

automaton prod_cons :

events:
  U Cons;
  C 1 StopProd;
  C StopCons;
  C Prod;
  C Choice;

int x = 0;
int y = 0;

initial : Choice

state Choice :
  to PX : when true , Choice;

```

```

state PX:
  to PX : when true , Prod with x = x + 1;
  to CX : when true , StopProd with y = 2;

state CX:
  to CX : when y <= 2 and y >= 0 , Cons with x = x - 1,
          y = y - 1;
  to Choice : when true , StopCons;

bad_states:
  state CX : x <= 10 and y <= 2 and y >= 0

controller 0 mask: hidden_variables: x, y
controller 1 mask: intervals: (x, [5.0, 15.0])

```

Input Options. SMACS is called by the command *bin/smacs.byte*. The main options offered to the user are:

- *-non-blocking*: ensures that the controlled system is deadlock-free.
- *-dot-output <string>*: gives the name of the dot file in which the controlled system is saved. By default, the dot file is not generated.
- *-text-output <string>*: gives the name of the text file in which the controlled system is saved. The format used in this text file to describe the controlled system is very close to the one used in the input language to describe the system to be controlled. By default, this file is not generated.
- *-widening-start <integer>*: gives the iteration at which the widening operator is used for the first time in the fixpoint computation (by default, it is 2).
- *-widening-freq <integer>*: gives the number of iterations (in the fixpoint computation) between two successive applications of the widening operator (by default, it is 1).
- *-descending-steps <integer>*: gives the number of descending steps i.e., the number of additional iterations to be performed in the fixpoint computation to refine the computed post-fixpoint (by default, it is 2).

- *lattice* {*intervals* | *octagons* | *polyhedra*}: gives the abstract lattice used in the computations. By default, the convex polyhedra are used.

The online version of SMACS only proposes the options *-non-blocking* and *-text-output* <*string*>.

A.3 Outputs of SMACS

SMACS provides several kinds of outputs to describe the results that it obtains:

- it prints onto the standard output the control function \mathcal{F}_i of each local controller \mathcal{C}_i and also gives the global control resulting from these local control decisions.
- if asked, it generates a dot file containing a description of the controlled system. This file provides a graphical representation of this system.
- if asked, it generates a text file containing a description of the controlled system. The format used in this file to describe the controlled system is very close to the one used in the input language to describe the system to be controlled.

The last two outputs are optional and are not generated when a local controller has a partial observation modeled by a mask corresponding to a covering of the state space. Indeed, as explained in chapter 3, a controlled system cannot be computed in this case.

Example A.7

We present the outputs generated by SMACS for the input file given in Example A.6. SMACS prints the following information onto the standard output:

```
*****
*           Forbidden states           *
*****
state CX : [|-x+y+10>=0; -y+2>=0; y>=0|]
*****
* SMACS output (general case) *
*****
```

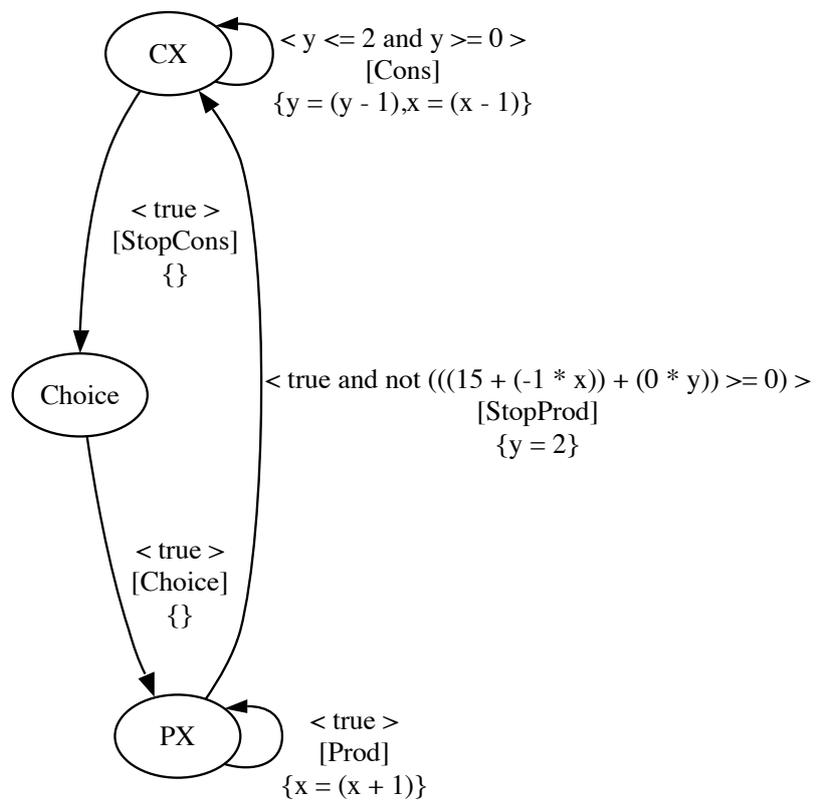


Figure A.2 - Controlled system generated by SMACS for the producer and consumer example described in Example A.6.

```

-----
Controller 1 function
-----
disable event StopProd in state PX when [|-x+15>=0|]
-----
Controller 0 function
-----

-----
Global control function
-----
disable event StopProd in state PX when [|-x+15>=0|]

```

Thus, the set of forbidden states $I(Bad) = \{\langle \ell, x, y \rangle \mid (x - y \leq 10) \wedge (0 \leq y \leq 2)\}$. The local controller 1 forbids the action *Stop_prod* in the location PX when $x \leq 15$ and the local controller 2 forbids no action. The resulting global control consists in forbidding the action *Stop_prod* in the location PX when $x \leq 15$.

The information written in the dot file gives the controlled system depicted in Figure A.2.

The description of the controlled system written in the text file is the following:

```

Automaton controlled_prod-cons :

events:
C Choice,
C Prod,
C StopCons,
C 1 StopProd,
U Cons

int y;
int x;

```

```
initial condition: Choice => x == 0 and y == 0

state CX :
  to Choice : when true, StopCons;
  to CX : when y <= 2 and y >= 0, Cons with y = (y - 1),
        x = (x - 1);

state PX :
  to CX : when not (((15 + (-1 * x)) + (0 * y)) >= 0), StopProd
        with y = 2;
  to PX : when true, Prod with x = (x + 1);

state Choice :
  to PX : when true, Choice;
```

Bibliography

- [ACH⁺95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.
- [AD90] R. Alur and D.L. Dill. Automata for modeling real-time systems. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 322–335, New York, USA, 1990. Springer-Verlag New York, Inc.
- [AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [AMP95] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. *Lecture Notes in Computer Science*, 999:1–20, 1995.
- [AMPS98] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proceedings of the 5th IFAC Cconference on System Structure and Control (SSC'98)*, pages 469–474. Elsevier Science, July 1998.
- [AP68] R. Aron and V. Pareto. *Traité de sociologie générale*. Droz, 1968.
- [ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.
- [BDMP03] P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In Warren A. Hunt, Jr and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192. Springer, July 2003.

- [BGWW97] Bernard Boigelot, Patrice Godefroid, Bernard Willems, and Pierre Wolper. The power of qdds. In *SAS '97: Proceedings of the 4th International Symposium on Static Analysis*, pages 172–186, London, UK, 1997. Springer-Verlag.
- [BL98] G. Barrett and S. Lafortune. A novel framework for decentralized supervisory control with communication. In *IEEE Conference on Systems, Man, and Cybernetics*, pages 11–14, San Diego, California, October 1998.
- [BL00] G. Barrett and S. Lafortune. Decentralized supervisory control with communicating controllers. *IEEE Transactions on Automatic Control*, 45(9):1620–1638, 2000.
- [Bou92] F. Bourdoncle. *Sémantiques des Langages Impératifs d'Ordre Supérieur et Interprétation Abstraite*. PhD thesis, Ecole Polytechnique, 1992.
- [BZ83a] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.
- [BZ83b] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
- [Cama] The CamlIDL Library. Available <http://caml.inria.fr/pub/old.caml.site/camlidl/>. [August 12, 2010].
- [Camb] The Camllib Library. Available <http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/camllib/index.html>. [August 12, 2010].
- [CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'77*, pages 238–252, 1977.
- [CDF⁺05] F. Cassez, A. David, E. Fleury, K.G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. *CONCUR 2005 - Concurrency Theory*, pages 66–80, 2005.
- [CDHR07] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games of incomplete information. *Logical Methods in Computer Science*, 3(3:4), 2007.

-
- [CE82] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag.
- [CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL '78*, pages 84–96, 1978.
- [Chu36] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.
- [CL08] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems (2nd edition)*. Springer, 2008.
- [Cou81] P. Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.
- [Dar05] P. Darondeau. Distributed implementations of Ramadge-Wonham supervisory control with petri nets. In *44th IEEE Conference on Decision and Control and European Control Conference*, pages 2107–2112, Sevilla, Spain, December 2005.
- [DDR06] M. De Wulf, L. Doyen, and J.-F. Raskin. A lattice theory for solving games of imperfect information. In J.P. Hespanha and A. Tiwari, editors, *HSCC*, volume 3927 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 2006.
- [DM02] D. D’Souza and P. Madhusudan. Timed control synthesis for external specifications. In *STACS '02: Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, pages 571–582. Springer-Verlag, 2002.
- [DRS01] N. Dor, M. Rodeh, and S. Sagiv. Cleanness checking of string manipulations in c programs via integer analysis. In *SAS '01: Proceedings of the 8th International Symposium on Static Analysis*, pages 194–212, London, UK, 2001. Springer-Verlag.

- [Dub06] J. Dubreil. Non-interference on symbolic transition system. Technical report, Uppsala University, February 2006.
- [Fix09] Fixpoint: an OCaml library implementing a generic fix-point engine, 2009. Available <http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/fixpoint/>. [August 12, 2010].
- [GD07] B. Gaudin and P.H. Deussen. Supervisory control on concurrent discrete event systems with variables. In *American Control Conference 2007*, July 2007.
- [GJ90] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [GM05] B. Gaudin and H. Marchand. Efficient computation of supervisors for loosely synchronous discrete event systems: A state-based approach. In *6th IFAC World Congress*, Prague, Czech Republic, July 2005.
- [GMP] The GMP library. Available <http://gmplib.org/>. [August 12, 2010].
- [God96] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*. Springer-Verlag New York, Inc., 1996.
- [HB91] C. Hoaxum and H. Baosheng. Distributed control of discrete event systems described by a class of controlled petri nets. In *Preprints of IFAC International Symposium on Distributed Intelligence Systems*, Virginia, 1991.
- [HHWt97] T. A. Henzinger, P-H. Ho, and H. Wong-toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463, 1997.
- [HK99] T.A. Henzinger and P.W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221(1-2):369–392, 1999.

-
- [HKG97] L.E. Holloway, B.H. Krogh, and A. Giua. A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems: Theory and Application*, 7:151–190, 1997.
- [HMR05] T.A. Henzinger, R. Majumdar, and J.-F. Raskin. A classification of symbolic transition systems. *ACM Trans. Comput. Logic*, 6(1):1–32, 2005.
- [HMU06] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
- [Hol04] G. J. Holzmann. *The SPIN Model Checker : Primer and Reference Manual*. Addison-Wesley Professional, 2004.
- [HPR97] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.
- [HTL08] R.C. Hill, D.M. Tilbury, and S. Lafortune. Covering-based supervisory control of partially observed discrete event systems for state avoidance. In *9th International Workshop on Discrete Event Systems*, May 2008.
- [Jea03] B. Jeannet. Dynamic partitioning in linear relation analysis. Application to the verification of reactive systems. *Formal Meth. in Syst. Design*, 23(1):5–37, 2003.
- [JM09] B. Jeannet and A. Miné. The APRON library, 2009. Available <http://apron.cri.enscm.fr/>. [August 12, 2010].
- [JMR06] T. Jéron, H. Marchand, and V. Rusu. Symbolic determinisation of extended automata. In *4th IFIP International Conference on Theoretical Computer Science*, Santiago, Chili, August 2006.
- [KG05] R. Kumar and V.K. Garg. On computation of state avoidance control for infinite state systems in assignment program model. *IEEE Trans. on Autom. Science and Engineering*, 2(2):87–91, 2005.

- [KGM91] R. Kumar, V. Garg, and S. I. Marcus. On controllability and normality of discrete event dynamical systems. *Systems and Control Letters*, 17:17–157, 1991.
- [KGM93] R. Kumar, V. Garg, and S.I. Marcus. Predicates and predicate transformers for supervisory control of discrete event dynamical systems. *IEEE Trans. Autom. Control*, 38(2):232–247, 1993.
- [KH90] B. H. Krogh and L. E. Holloway. Synthesis of feedback control logic for a class of controlled petri nets. *IEEE Trans. on Automatic Control*, 35(5):514–523, 1990.
- [KH91] B. H. Krogh and L. E. Holloway. Synthesis of feedback control logic for discrete manufacturing systems. *International Federation Of Automatic Control*, 27(4):641–651, 1991.
- [KH92] B. H. Krogh and L. E. Holloway. On closed-loop liveness of discrete event systems under maximally permissive control. *IEEE Trans. on Automatic Control*, 37(5):692–697, 1992.
- [KH96] R. Kumar and L.E. Holloway. Supervisory control of deterministic petri nets with regular specification languages. *IEEE Trans. on Automatic Control*, 41(2):245–249, 1996.
- [KL10] G. Kalyon and T. Le Gall. The SMACS Tool, 2010. Available <http://www.smacs.be/> [August 12, 2010].
- [KLMM10] G. Kalyon, T. Le Gall, H. Marchand, and T. Massart. Decentralized control of infinite systems. *submitted to Journal of Discrete Event Dynamical Systems*, 2010.
- [Kna28] B. Knaster. Un théorème sur les fonctions d'ensembles. *Annales de la Société Polonaise de Mathématique*, 6:194–212, 1928.
- [Kri63] S.A. Kripke. Semantical consideration on modal logic. *Acta Philosophica Fennica*, 16:83–94, 1963.
- [Kro87] B.H. Krogh. Controlled petri nets and maximally permissive feedback logic. In *Proc. of the 25th Annual Allerton Conf.*, University of Illinois, Urbana., 1987.

-
- [Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
- [LGJJ06] T. Le Gall, B. Jeannet, and T. Jéron. Verification of communication protocols using abstract interpretation of fifo queues. In Michael Johnson and Varmo Vene, editors, *11th International Conference on Algebraic Methodology and Software Technology, AMAST '06, Kuressaare, Estonia*, LNCS. Springer-Verlag, July 2006.
- [LJM05] T. Le Gall, B. Jeannet, and H. Marchand. Supervisory control of infinite symbolic systems using abstract interpretation. In *CD-C/ECC'05*, December 2005.
- [Lov78] D.W. Loveland. *Automated theorem proving: A logical basis (Fundamental studies in computer science)*. Elsevier North-Holland, 1978.
- [LW88] F. Lin and W.M. Wonham. On observability of discrete-event systems. *Inf. Sci.*, 44(3):173–198, 1988.
- [Mar97] J. C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill Higher Education, 1997.
- [Mat89] F. Mattern. Virtual time and global states of distributed systems. In *Proceedings of the Workshop on Parallel and Distributed Algorithms*, pages 215–226, North-Holland / Elsevier, 1989.
- [McM93] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers., 1993.
- [Min01] A. Miné. The octagon abstract domain. In *Proceedings of the Workshop on Analysis, Slicing, and Transformation (AST'01)*, IEEE, pages 310–319, Stuttgart, Germany, October 2001. IEEE CS Press.
- [MLG] The MLGMPIDL Library. Available <http://www.inrialpes.fr/pop-art/people/bjeannet/mlxxxidl-forge/mlgmpidl/>. [August 12, 2010].
- [MPF] The MPFR library. Available <http://www.mpfr.org/>. [August 12, 2010].

- [MPS95] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In Ernst W. Mayr and Claude Puech, editors, *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer-Verlag, March 1995.
- [Mye79] G. J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 1979.
- [OCa09] The programming language Objective CAML, 2009. Available <http://caml.inria.fr/>. [August 12, 2010].
- [Pap94] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.
- [Pet62] C.A. Petri. Kommunikation mit automaten. *Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn*, Schrift Nr 2, 1962.
- [Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977.
- [PR89] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In G. Ausiello, M. Dezani-Ciancaglini, and S.R. Della Rocca, editors, *ICALP*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989.
- [PVK01] D. Peled, A. Valmari, and I. Kokkarinen. Relaxed visibility enhances partial order reduction. *Formal Methods in System Design*, 19(3):275–289, 2001.
- [QS82] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351, London, UK, 1982. Springer-Verlag.
- [RDFV88] Cieslak R., C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete-event process with partial observation. *IEEE Trans. on Automatic Control*, 33(3):249–260, 1988.

-
- [Rei84] J.H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2):274–301, 1984.
- [Ric03] K. Ricker, S.L. Rudie. Knowledge is a terrible thing to waste: using inference in discrete-event control problems. In *ACC'03: Proceedings of the 2003 conference on American Control Conference*, pages 2246–2251, 2003.
- [RP05] S. Riedweg and S. Pinchinat. You can always compute maximally permissive controllers under partial observation when they exist. In *Proc. 2005 American Control Conference.*, volume 4, pages 2287–2292, Portland, Oregon, jun 2005.
- [RW87] P.J. Ramadge and W.M. Wonham. Modular feedback logic for discrete event systems. *SIAM J. Control Optim.*, 25(5):1202–1218, September 1987.
- [RW89] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98, 1989.
- [RW92a] K. Rudie and W. M. Wonham. An automata-theoretic approach to automatic program verification. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 3770–3777, Tucson, Arizona, 1992.
- [RW92b] K. Rudie and W. M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, November 1992.
- [RW95] K. Rudie and J. C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Transactions on Automatic Control*, 40:1313–1319, 1995.
- [SAF09] M. Sköldstam, K. Akesson, and F. Fabian. Supervisory control applied to automata extended with variables. *Technical Report R001/2008, Department of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden.*, 2009.

- [Sre93] R. S. Sreenivas. On a weaker notion of controllability of a language k with respect to to a language l . *IEEE Trans. on Automatic Control*, 38(9):1446–1447, 1993.
- [Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
- [Thi05] J. G. Thistle. Undecidability in decentralized supervision. *Systems & Control Letters*, 54(5):503–509, 2005.
- [TK97] S. Takai and S. Kodama. M-controllable subpredicates arising in state feedback control of discrete event systems. *International Journal of Control*, 67(4):553–566, 1997.
- [TK98] S. Takai and S. Kodama. Characterization of all m-controllable subpredicates of a given predicate. *International Journal of Control*, 70:541–549(9), 10 July 1998.
- [TKU94] Shigemasa Takai, Shinzo Kodama, and Toshimitsu Ushio. Decentralized state feedback control of discrete event systems. *Syst. Control Lett.*, 22(5):369–375, 1994.
- [Tri02] Stavros Tripakis. Decentralized control of discrete event systems with bounded or unbounded delay communication. In *WODES '02*. IEEE Computer Society, 2002.
- [Tsi89] J. N. Tsitsiklis. On the control of discrete event dynamical systems. *Mathematics of Control, Signals and Systems*, 2(2):95–107, 1989.
- [TUK95] S. Takai, T. Ushio, and S. Kodama. Static-state feedback control of discrete-event systems under partial observation. *IEEE transactions on automatic control*, 40:1950–1954(11), 11 November 1995.
- [Ush89] T. Ushio. On controllable predicates and languages in discrete-event systems. In *Proc. of the 28th Conference on Decision and Control*, pages 123–124, Tampa, Floride, Decembre 1989.
- [VNJ90] N. Viswanadham, Y. Narahari, and T.L. Johnson. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using petri net models. *IEEE Transactions on Robotics and Automation*, 6(6):713–723, 1990.

-
- [VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, June 1986.
- [Won05] W.M. Wonham. Lecture notes on control of discrete-event systems. Technical report, University of Toronto, 2005.
- [WR87] W. Wonham and P. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, 25(3):637–659, 1987.
- [XK09] S. Xu and R. Kumar. Distributed state estimation in discrete event systems. In *ACC'09: Proceedings of the 2009 conference on American Control Conference*, pages 4735–4740. IEEE Press, 2009.
- [YL02] T.-S. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Discrete Event Dynamic Systems*, 12(3):335–377, 2002.
- [YL04] T.-S. Yoo and S. Lafortune. Decentralized supervisory control with conditional decisions: supervisor existence. *IEEE Transactions on Automatic Control*, 49(11):1886–1904, 2004.

