UNIVERSITÀ DEGLI STUDI DI SIENA
DIPARTIMENTO DI MATEMATICA
DOTTORATO DI RICERCA IN LOGICA MATEMATICA E INFORMATICA TEORICA

PH.D. THESIS

Formal Methods for the Analysis of Open Systems with Applications to Security Properties

Fabio Martinelli

COORDINATOR
Prof. Franco Montagna

February 1999

Dipartimento di Matematica, Via del Capitano 15, I-53100 Siena
E-mail: `martinel@di.unipi.it`

Abstract

This thesis concerns the formal verification of *open* systems. The behaviour of an open system may be not completely specified and may present some uncertainty due to the environment in which it operates.

In particular, we study the *module checking* problem for several temporal logics, as defined by Kupferman and Vardi. A module is a Kripke model refined in order to take into account a possible interaction with an environment. Every environment induces a particular behaviour of the module, which may be considered as a standard Kripke model. Hence, given a module M and a temporal logic formula ϕ, the module checking problem is the verification whether every induced behaviour of the module M satisfies the property ϕ. We show how to use compositional analysis techniques developed in concurrency theory for the solution of this problem, in particular for modal $\mu-$calculus.

Then we show how computer security properties may be defined through underspecification. It turns out that it is very natural to consider the verification problems that arise in security property analysis as module checking problems. In particular, we study information flow security properties, as proposed by Focardi and Gorrieri, together with security protocol properties. We define suitable compositional analysis techniques in order to study these kinds of properties. The resulting approach may be regarded as a new methodology for the analysis of security properties.

Finally, we study a synthesis problem that arises when we consider systems which may have an unspecified component, i.e. subsystem construction. We show how to improve algorithms for solving this problem for particular cases in the context of process algebra.

Acknowledgments

First of all, I would like to thank my advisor Andrea Masini, for his helpful suggestions and in particular because he accepted to follow me in a field that was not in the mainstream of his research interests.

Thanks are due to Andrea Maggiolo-Schettini for many stimulating discussions and encouragements over these years.

I thank my external referees Pierpaolo Degano and Roberto Gorrieri, for their careful reading of an earlier draft of this thesis, and for their helpful comments and criticisms.

I would like to thank the Department of Computer Science of Pisa for the support given me during the completion of my Ph.D. course.

At last but not least I would like to thank Maria Rosaria, without her presence this work could never have been completed.

to my parents

Contents

Part I

Formal Methods for the Analysis of Open Systems

Introduction

This thesis concerns the study and the verification of complex systems with unspecified components. We investigate many situations and problems that may be described through underspecification.

In particular we deal with so called *reactive* systems as originally defined by Pnueli in his seminal work (see [80]). This kind of systems keeps an *ongoing* interaction with the environment by receiving and emitting stimuli. Many computer systems can be described by means of this paradigm, for example operating systems, monitoring programs and network protocols. In contrast with *transformational* systems, which generally given an input produce an output, their semantics should rely on their behaviour and on their capabilities of interaction during their progress.

A typical situation where underspecification arises is during the development of complex systems. This activity involves at least three steps: *specification, implementation* and *verification*. In the first step the requirements of the system are written in some language, in the second step the implementation is provided in some executable language, and in the third step the consistency of the implementation with the specification is checked. One of the major challenges of computer science is to find suitable formalisms and methods to perform these activities correctly.

Formal methods can be considered as a set of mathematical tools and notions used to reason rigorously about the behavior and the properties of systems. The language for the description of the properties of these systems and the language for the description of systems must have a clear formal semantics. Once a formal model of a system has been given, the proof of the coherence between the specification and the implementation can be performed. While in general the specification language can be descriptive, the implementation is given in a prescriptive language.

When the specification language is different from the implementation one, the verification is called *heterogeneous*, otherwise *homogeneous*. In this thesis we will show applications of both kinds of verification.

Semantics

The semantics of languages for the description of (sequential and concurrent) systems can be defined in several ways; the following classification is widely accepted (see [107]):

Operational semantics. This assumes the notion of an abstract machine where programs are executed and shows the transitions of states of the machine, possibly together with some information about the activity that caused the changes. This approach was formalized by Plotkin in [79] in the so called *Structured Operational Semantics* (SOS, for short), where states of the machine represent terms of the language, and transitions are obtained by an inference system. Moreover the semantics of terms is generally given by means of the semantics of their subterms.

Denotational semantics. Every program has an associated mathematical object (its denotation). This way of providing semantics for programming languages has been very fruitful for the sequential programming paradigm, where the meaning of programs can be identified by the function they compute.

Axiomatic semantics. The semantics is not given directly, but a proof system for terms of the language is provided. The characteristics of the constructs of the language are expressed implicitly by the axioms and rules of the proof system.

Formal semantics for programming languages should abstract from irrelevant aspects of the language. From a formal semantics designers, implementers and programmers of the language can benefit.

Designers may give unambiguous definitions for their languages. Moreover they may establish relationships among operators of the language and also between operators of different languages (but with the same formal model).

Implementers can have a clear guide for their work, hence they can easily produce implementations of the language that are machine-independent.

Programmers benefit from formal semantics since they can get a clear "mental" model of each functionality of the language, and so they can program their applications (almost) independently of the machine in which the language is implemented.

Verification

One of the most important activities of software/hardware developers should be the certification that their programs/systems meet the requirements, as expressed by the specification. If the languages for specification and implementation have formal semantics, then the certification can be carried out in a formal setting. Moreover this verification may be computer-aided. There are two main approaches in the field of formal verification of reactive/concurrent systems (see [20]):

1. *proof* theoretic,

2. *model* theoretic.

The *proof* theoretic approach uses proof systems, which consist of axioms and rules, for inferring properties of programs. Generally, a proof system is divided into two parts: 1) a logical part 2) a specific part for program reasoning (see [27]). Theorem proving is the activity of finding a proof of a property within the proof system. In this approach it is possible to deal directly with infinite domains, for example by exploiting structural induction principles. Hand made proofs are usually error-prone and tedious, hence automated theorem provers and proof checkers are needed. The former permit one to automatically find the proof of the correctness of the system. The drawbacks of theorem proving result from the intrinsic complexity of the problem thus automated tools work well for small systems while, in general cases, require non trivial human efforts.

The *model* theoretic approach simply considers a program as a model for the interpretation of a logical formula, which represents the requirements of specification. Thus the *verification* of the consistency of the program and of its specification consists in checking whether the semantics of the program is a model of the formula (*model checking*). In particular this approach has been advocated for the analysis of concurrent systems. In a landmark paper Clarke *et al.* (see [19]) introduce the notion of *model checking* as an automated method for the analysis of protocols.

The protocols are modeled as Kripke models, i.e. triples $\langle W, R, L\rangle$, where W is a set of worlds (or states), R is a relation on W and L is a labeling function. A Kripke model may be considered as a simple computational model. States represent possible configurations of the machine. The labeling function L expresses the basic properties that hold in a state, and the accessibility relation R shows the possible next computation steps of the system.

The specification is expressed in a temporal logic, for example *Computational Tree Logic* (CTL, for short). This logic permits us to reason about the temporal relations of events. If the Kripke model is finite then the model checking problem can be solved with a complexity that is polynomial in the length of the formula and in the dimension of the model (see [19]).

Among the positive aspects of *model checking*, we recall the following ones: it is automatic (so it can be used by non experts of formal methods) and fast, it can find errors and produce evidence of errors of systems, which may subsequently be exploited for debugging. Among the drawbacks of model checking, we recall: generally the system to be checked has to be completely specified; infinite state systems are not directly manageable (instead suitable abstractions must be provided).

In a certain phase of the implementation, it is possible that decisions which have already been taken about the design of a component, make impossible the design of other components in such way that the whole system could satisfy the specification. By applying model checking techniques, this situation can be detected only at the end of the development stage. This problem has been tackled by many researchers. A possible approach is to study the system in a compositional way, i.e. by deriving properties of a system starting from the properties of its subcomponents.

Compositional analysis

Compositional analysis techniques have been developed for many concurrent languages (see [4, 40, 55, 56, 106]). Suppose that c **sat** ϕ means the system c satisfies a specification

ϕ, expressed in some logical formalism. Moreover assume that $\otimes$ is an operator of the language. A typical rule for compositional reasoning is the following:

$$\frac{c_1 \mathbf{\ sat\ } \phi_1 \qquad c_2 \mathbf{\ sat\ } \phi_2}{c_1 \otimes c_2 \mathbf{\ sat\ } \phi}.$$

The rule should be read upward i.e. in order to verify that a composed system $c_1 \otimes c_2$ satisfies a property ϕ check that components c_1 and c_2 verify the two properties ϕ_1 and ϕ_2, respectively. The problem is reduced to two subproblems, hopefully both simpler than the previous one. The choice of a right decomposition of the property ϕ in the two properties ϕ_1 and ϕ_2 is usually a difficult task.

In [55] Larsen and Xinxin tackle a problem related to the correct decomposition of properties. They study how to compute the properties that unspecified subcomponents of a system must verify so that the whole system satisfies a certain requirement. Moreover, they address the problem of finding properties of subcomponents that are as *weak* as possible, in order not to restrict unnecessarily the choice of further implementation steps.

They introduce the notion of *context*. Roughly a context $C(X_1, \ldots, X_n)$ is a system that presents some non specified components $X_1, \ldots, X_n$. Suppose we have the following problem: find a property $pe(C, \phi)$ such that $C(P)$ **sat** ϕ iff P **sat** $pe(C, \phi)$. In other words we search for a formula such that a subsystem satisfies this formula iff the complete system (obtained by the insertion of the submodule P in the context $C(X)$) satisfies the formula ϕ. It is worthwhile noticing that the requirements, expressed by the formula ϕ are changed into the ones expressed by the formula $pe(C, \phi)$, in order to correctly take into account the behaviour of the components already defined (i.e. the context $C(X)$).

In this way, after defining a partial implementation, it is possible to find the minimal properties that the unspecified components must ensure in order to build a complete system with particular requirements. A similar problem was analyzed by Andersen (see [4, 5]) but instead of using the abstract notion of context (with its operational semantics) he prefers to start his analysis from a very general *process algebra* (see [43, 44, 69]). In Andersen's terminology the formula is a partially evaluated, or rather, reduced formula.

We study a slight extension of their work. The idea is to define the function that performs the reduction of the formula (or the partial evaluation), according to the components already specified, directly from the SOS definition of the language in which the semantics of systems is given, instead of using a specific language, even though very powerful, such as context systems. The underlying motivation indicates SOS definitions as a suitable basic language for which general theories may be developed (see [3, 12, 26, 90]).

Module checking

Part of this thesis is dedicated to the study of *open systems* as defined recently by Kupferman and Vardi in [50, 51, 101]. *Open* systems heavily depend on the environment in which they operate. The environment is thus able to influence the behaviour of systems. These systems are called *open*, in contrast with *closed* systems, whose behaviour is not influenced by the environment. If we consider an *open* system then it is possible that in certain states it requires interactions with the environment in order to evolve. Hence certain potential capabilities of the system may be exploited or not depending on the interaction with the environment in which the system operates. Generally, different environments may induce different behaviours of the system.

Kupferman and Vardi define the following verification problem: check whether the induced behaviours of an open system with respect to any possible environment satisfy a certain property (generally it involves an infinite number of verifications).

Assume we have a machine that has to gather food for two kinds of pretty animals, for example goldfish and green parrots. It could work as follows. First it prompts the environment with a request for food for the goldfish or for the green parrots. Next, it sends food to the chosen group of animals and returns to ask the environment for food. For example, we may ask if wherever we place the machine, it is always possible to give food to the goldfish. Clearly, the answer depends on the environment in which the machine is inserted. Imagine an environment which loves animals but dislikes goldfish, then it would only give food to the second group of animals! Kupferman and Vardi argue that model

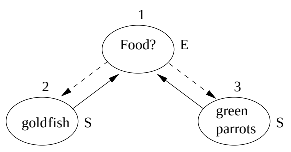


Figure 1: Module for the example.

checking approach cannot correctly tackle the above verification problem, since model checking assumes the so called *possible* world semantics of Kripke Models, namely that every world, which is *accessible* from another, may actually be accessed. The underlying assumption of model checking is that the system is checked w.r.t. an environment that enables every behaviour. Hence, model checking algorithms defined for reasoning about Kripke models do not work correctly for the analysis of open systems. There is a need to define different model checking problems for open systems.

Kupferman and Vardi define the structure of *module*. A module is a refined Kripke model, such that the set of worlds is partitioned into two sets: the *environment* worlds where an interaction of the system with the environment is necessary in order to proceed, and the *system* worlds, where the system can proceed autonomously. Every environment in composition with the *module* induces a particular computation tree, which depends on the choices made by the environment in the *environment* worlds. Figure 1 shows a possible definition for the module of our food gathering machine. It is worthwhile noticing that if we consider the module as a Kripke model then the answer to our question on goldfish is YES. In fact in this case from every world a path exists that reaches the world representing the delivery of food to the goldfish.

Given an open system, modeled by a finite module, and a temporal logic formula expressing the desired requirements, *module checking* is the verification whether, for any environment, the induced system satisfies the formula. Kupferman and Vardi propose an algorithmic approach to the solution of module checking for many temporal logics, such as PLT, CTL, CTL^*, by applying constructions on infinite tree automata (see [98]).

In this thesis, we study the module checking problem as a situation of underspecification. We give a uniform alternative solution to the module checking problem for several temporal logics. In particular, we solve this problem for modal $\mu-$calculus (see [49, 92]). The underlying idea relies upon compositional analysis concepts.

Moreover, while module checking is an interesting theoretical problem in itself, we believe that its paradigm is well suited for modeling interesting real problems, such as security property analysis.

Synthesis problems

When considering systems with an unspecified component, one may wonder if there exists an implementation that can be plugged into the system replacing the unspecified one, by satisfying some properties of the whole system. Following an analogy with top-down design methodology, at a certain point in the development it is possible to automatically derive the implementation of subcomponents, that have still to be designed. In this way, it is possible to avoid the final verification step since the system is correct by construction.

This problem was first analyzed by Merlin and Bochman in [68]. Their specifications are given as transition systems and the subcomponents are considered correct if the combined system has the same execution sequences (traces) as the specification. In [86], Shields reformulated the problem in the CCS *process algebra* ([70]) by calling it interface equation. This name depends on the fact that the problem can be restated as follows (see chapter 4 for a complete overview) :

$$\exists \text{ a process } X \text{ such that } (p\|X)\backslash L \approx q$$

where p, q and X are processes, $\|$ is the parallel composition operator and $\approx$ is an equivalence relation. The term *interface* follows from the fact that the component (process) p can actually be seen as being composed by two components that have to communicate by exploiting the communication features of X. Since programs can be represented by models of some temporal logic, it appears reasonable to use classical satisfiability procedures for the automatic synthesis of programs. By using partial evaluation techniques, we can use satisfiability procedures to solve also interface equations (obviously when the two systems p and q are finite-state).

On the other hand, in this thesis we prefer to perform our analysis by using more specific methods. We study the theory proposed by Shields and we find a property that we exploit to improve existing algorithms. Under certain conditions, we can achieve a better computational complexity and we can synthesize better solutions than through temporal logic satisfiability procedures.

Security analysis

In the second part of this thesis we apply the ideas developed in the first one for the analysis of security problems. In the last few years, research on the definition of formal methods for the analysis and the verification of security properties of systems has increased greatly. This is mainly due to the practical relevance of these systems and moreover to preliminary encouraging results achieved by the application of formal methods to security property analysis.

There are many definitions of security properties, among which information flow properties. Generally, information flow in complex systems must respect a particular policy. Many notions of information flow security properties exist, in particular Focardi and Gorrieri have defined the so called *Non Deducibility on Composition* (NDC, for short) in the CCS *process algebra*. The problem consists of modeling a system where there are several levels of confidentiality, and information flow should be directed from lower to higher level processes, only.

The intuition behind NDC is that for every high level process that the system is composed with, the resulting system always has the same behaviour with respect to low level processes. This means that no high user can downgrade information to low level processes, since if a system is NDC then low level processes always perceive the same behaviour of the system. Formally it can be stated as follows (see chapter 5 for a complete overview):

$$\forall \text{ high processes } X \text{ we have } (E \| X) \backslash H \approx E \backslash H.$$

After the above discussion, the relationship between this problem and the module checking one should be clear. The intuition of the relevance of properties that must be specified with respect to many environments was identified in the area of information flow analysis by Focardi and Gorrieri in ([33]).

We have followed the same ideas we used for solving *module checking* for the analysis of NDC like properties. So we have defined a decision procedure for NDC like properties (when restricted to finite-state systems). Then we have pushed the analogy further and we have tried to model other security properties by following the module checking schema, in particular security protocol properties. Among these protocols we recall the authentication ones. These are used in distributed systems, where each process must identify the others during a communication session.

Authentication is a mandatory task also for accessing remote services by users, in particular for services managing security-sensitive information. These protocols involve at a conceptual level only a few steps of interaction among parties nevertheless they are very difficult to be designed and moreover to be proved correct. This is testified by the great number of erroneous protocols found in the literature (see [17, 18, 59, 60, 61]). Concepts and tools from logic and concurrency theory have recently been exploited for the analysis of these protocols. As in the classical verification problems we have two possible approaches:

A *proof* theoretic approach relies on authentication logics, i.e. knowledge or belief logics with axioms and rules for specifically reasoning about authentication protocols (see [2, 48, 78, 96]). As usual, the initial assumptions of the protocols and an "idealization" (namely, a reduction of the protocols to a set of formulas) are used to infer properties that must hold at the end of a run of the protocol. Unfortunately, as observed by many authors, the "idealization" step is very difficult (and not formally defined) in authentication logics, hence it sometimes causes unintentional modifications to the protocol semantics, that may lead to inferring the correctness of protocols that actually present some errors. A notable example is the so called Needham-Schroeder Public Key protocol (see [73]). It was proven correct in [17], but later in [59] a non trivial error in the protocol was found and fixed (about twenty years after the publication). The problem was in the "idealization" step.

A *model* theoretic approach is based on the use of the process algebra theory for the analysis of authentication protocols. One advantage is the possibility to reason on a representation of the protocols which is near to an actual implementation. Moreover the concepts and the tools already developed in this area can be fruitfully exploited. In [59], Lowe shows how to use a verification tool for checking properties among processes for findings flaws in protocols. He formally describes a model of verification of these protocols, which assumes the presence of an agent that does not correctly take part in the communications, by trying to leak secrets or to impersonate other agents in order to access information or services that he is not legitimate for accessing. This agent acts as an intruder that can listen, fake and intercept messages exchanged during the communications.

The presence of an environment (i.e. the intruder) in which the systems operate must be taken into account in the definition of correctness of protocols. This is one of the main differences with standard protocols, and the source of many misunderstandings about the proof of correctness of protocols.

Our proposal is to consider the intruder as an unknown, or rather unspecified, component of the system and to reason about the correctness of the protocol with respect to each unspecified component. So we feel natural to consider the verification problem as an example of module checking (obviously in a different theoretical framework). Naturally, the next step is to apply the compositional analysis concepts also for the solution of this problem in this setting. We have successfully followed the previous idea, by producing a new approach for the analysis of cryptographic protocols.

An implementation of this approach has been developed. Preliminary experiments show that these ideas may be fruitfully exploited in practice.

We believe that the analysis methods given in the second part of this thesis may be regarded as a unified framework for the analysis of computer security properties.

Furthermore, security analysis seems to be an appropriate example for the necessity of a study of practical applications of the *module checking* paradigm.

Organization of the thesis

In the first two chapters we provide the reader with a background tailored for treatment of the remainder of the thesis. The original contributions are above all in chapters 3,4,5 and 6.

The thesis consists of the following chapters:

- **Chapter 1.** A brief introduction to temporal logics and fixpoint logics ($\mu-$calculi) is given. A slight variant of Walukiewicz's axiomatization (see [103, 104]) for $\mu-$calculus is shown, where formulas interpreted over deterministic Labeled Transition Systems. Some other variants of the $\mu-$calculus are described.

- **Chapter 2.** An introduction to SOS formats is presented, together with CCS *process algebra*. We recast the theory of [4, 40, 55] for defining partial evaluation functions for languages whose semantics is given by means of a GSOS definition.

- **Chapter 3.** A uniform approach is proposed for the solution of the *module checking* problem for modal $\mu-$calculus, and other temporal logics. The approach relies on compositional analysis techniques.

- **Chapter 4.** A synthesis problem is studied, namely finding the solution of the interface equations. We show how to improve algorithms for solving this problem. In particular, we improve an algorithm that produces "good" solutions.

- **Chapter 5.** An application of compositional analysis techniques for establishing information flow security properties is proposed. Moreover, an extension of NDC theory of Focardi and Gorrieri is given for treating with real-time systems. A prototype tool is provided to the reader for computer aided analysis.

- **Chapter 6.** A new methodology is proposed for analyzing security properties of cryptographic protocols. It is based on compositional analysis concepts. A tool which implements the theory is presented.

Some results of this thesis have been already published (see [62, 65, 66, 67]) or are under consideration for publication (see [64]).

Chapter 1

Temporal logics

In this chapter, we briefly recall some basic definitions about (propositional) temporal logics. In particular we follow the treatment of Emerson in [27]. Then we describe in more detail an expressive branching time logic, called modal $\mu-$calculus, that subsumes all the others presented in this thesis. Walukiewicz's proof system (see [103]) that completely characterizes the logic is presented. We slight modify this system in order to deal with deterministic Labeled Transition Systems. A particular attention is given to variants of $\mu-$calculus, that have received a great deal of interest for their technical convenience w.r.t. standard $\mu-$calculus (see [5, 10, 55, 92]). These concepts will be used later in this thesis for program verification purposes.

1.1 Introduction

Temporal reasoning is tied to the necessity to deal with assertions whose truth value may change during time. It is reasonable to observe the behaviour of complex reactive systems during their activities. Hence, we can imagine to express properties like "after performing the activity a the system eventually performs an activity b", or "the system is able in any possible future moment to perform some activity". These assertions express a temporal relation among events. In his landmark work (see [80]) Pnueli proposed temporal logics as a well-suited formalism to reason about reactive system behaviour. His intuition has been followed by many researchers. Since non termination is, usually, one of the main characteristics of concurrent systems, it is reasonable to search for a formalism whose operators can express properties about possibly non terminating executions of systems. Temporal logic modalities permit to reason about executions (computations) of systems.

Temporal logics may differ about the underlying nature of time which is assumed: if time has only a single possible future moment we call them *linear* time logics, otherwise if there may be many possible future moments, we call them *branching* time logics. Temporal operators of the logic typically reflect the underlying nature of time, so *linear* time temporal logics have operators for expressing properties about a single time line, while *branching* time logics have also operators that permit to quantify along possible time lines

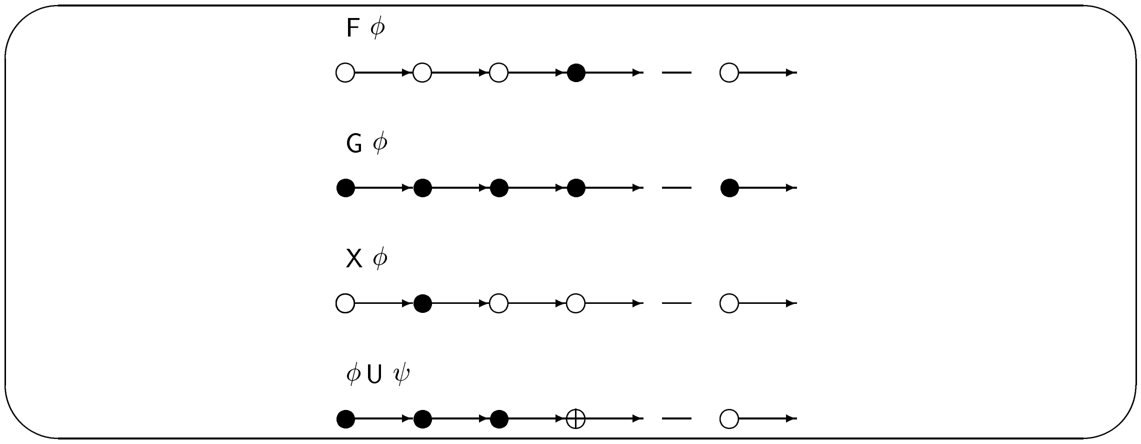


Figure 1.1: Intuition for linear-time operators. In the figure, the black states satisfy formula ϕ and the crossed state satisfies ψ.

(also referred to as computation tree) that may start from a time point.

1.2 PLTL

In this section, we recall the linear time structures that formalize the notion of time line, and we present a simple (propositional) linear time logic ($PLTL$, for short). Let AP be an underlying set of atomic propositions.

Definition 1.1 *A linear time structure is a triple $M = \langle S, x, L\rangle$, where S is a set of states, $x : \mathbb{N} \longrightarrow S$ is an infinite sequence of states, and $L : S \longrightarrow 2^{AP}$ is a labelling of each state with a set of atomic propositions in AP true at the state.*

For the infinite sequences we sometimes use the more convenient representation $x = (s_0, s_1, \ldots) = (x(0), x(1), \ldots)$. The state $x(0)$ is referred to as initial state of the time line x.

The basic temporal operators of this logic are $\mathsf{F}\ \phi$ ("sometimes ϕ"), $\mathsf{G}\ \phi$ ("always ϕ"), $\mathsf{X}\ \phi$ ("next time ϕ") and $\phi\,\mathsf{U}\ \psi$ ("ϕ until ψ"). A graphical explanation of these operators is shown in figure 1.1. In the sequel we informally say that a formula $PLTL$ holds in a state of a time line by intending that it holds in the time line that starts from this state. The formula $\mathsf{F}\ \phi$ is true in a time line if and only if there is a reachable state (future) where ϕ holds. $\mathsf{G}\ \phi$ is true in a time line if and only if at every future time ϕ holds. $\mathsf{X}\ \phi$ is true in a time line if the formula ϕ holds in the next time. Finally $\phi\,\mathsf{U}\ \psi$ holds in a time line if a state is reachable s.t. ψ holds in this state, and the crossed states satisfy the property ϕ (excepted, possibly, the last one).

The formulas of $PLTL$ logics are built using the following grammar:

$$\phi = p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\ \phi \mid \phi_1\,\mathsf{U}\ \phi_2$$

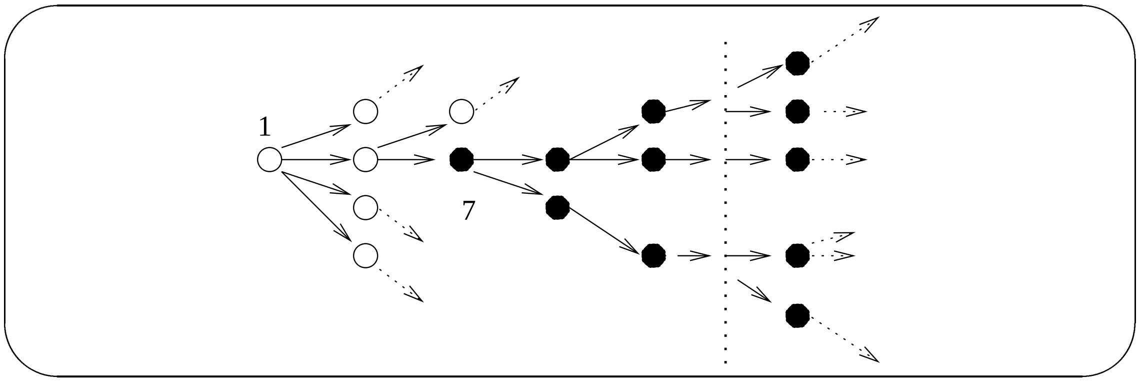


Figure 1.2: A branching structure, where the black states are labeled by the proposition p. Hence, the CTL^* formula $\exists\,\mathsf{F}\;\forall\,\mathsf{G}\;p$ is true in the state 1, since in the state 7 the formula $\forall\,\mathsf{G}\;p$ holds and 7 is reachable from 1.

where $p \in AP$ and AP is a set of propositional symbols. We can then define the other temporal operators as derived ones:

$$\begin{array}{lcl}
\mathbf{F} & \doteq & p \wedge \neg p \\
\mathbf{T} & \doteq & \neg \mathbf{F} \\
\phi_1 \vee \phi_2 & \doteq & \neg(\neg\phi_1 \wedge \neg\phi_2) \\
\phi \Longrightarrow \psi & \doteq & \neg\phi \vee \psi \\
\phi \Longleftrightarrow \psi & \doteq & (\phi \Longrightarrow \psi) \wedge (\psi \Longrightarrow \phi) \\
\mathsf{F}\;\phi & \doteq & \mathbf{T}\;\mathsf{U}\;\phi \\
\mathsf{G}\;\phi & \doteq & \neg\;\mathsf{F}\;\neg\phi
\end{array}$$

Among the derived temporal operators there is also $\psi\;\mathsf{B}\;\phi$ ("ψ before ϕ"), which expresses the fact that during a time line the formula ψ is true before the formula ϕ, this abbreviates the formula $\neg(\neg\psi\;\mathsf{U}\;\phi)$.

The formulas of $PLTL$ are interpreted over linear time structures $M = (S, x, L)$. The notation $M, x \models \phi$ means that the formula ϕ is true in the time line x of the structure M ($M, x \not\models \phi$ on the contrary). When M is clear from the context we usually write $x \models \phi$. For notational convenience we write x^i for the suffix $s_i, s_{i+1}, \ldots$ of the time line x. The truth relation ($\models$) is defined inductively in the structure of the formula ϕ:

$$\begin{array}{lll}
x \models p & \text{iff} & p \in L(s_0) \\
x \models \phi_1 \wedge \phi_2 & \text{iff} & x \models \phi_1 \text{ and } x \models \phi_2 \\
x \models \neg\phi & \text{iff} & \text{not } x \models \phi \\
x \models \mathsf{X}\;\phi & \text{iff} & x^1 \models \phi \\
x \models \psi\;\mathsf{U}\;\phi & \text{iff} & \exists j(x^j \models \phi \text{ and } \forall k < j(x^k \models \psi))
\end{array}$$

We say that a $PLTL$ formula ϕ is *satisfiable* iff there exists a linear-time structure $M = (S, x, L)$ s.t. $M, x \models \phi$. We say that any such structure is a *model* for ϕ. We say ϕ is valid, and write $\models \phi$, iff for all linear structures $M = (S, x, L)$ we have $M, x \models \phi$.

Below we give some simple equivalences among $PLTL$ formulas:

$$\begin{array}{ll}
(1) & \models \mathsf{G}\ \mathsf{G}\ \phi \Longleftrightarrow \mathsf{G}\ \phi \\
(2) & \models \mathsf{F}\ \mathsf{F}\ \phi \Longleftrightarrow \mathsf{F}\ \phi \\
(3) & \models \phi \Longrightarrow \mathsf{F}\ \phi \\
(4) & \models \mathsf{G}\ \phi \Longrightarrow \phi \\
(5) & \models \mathsf{F}\ \phi \Longleftrightarrow \phi \vee \mathsf{X}\ \mathsf{F}\ \phi \\
(6) & \models \mathsf{G}\ \phi \Longleftrightarrow \phi \wedge \mathsf{X}\ \mathsf{G}\ \phi \\
(7) & \models \psi\ \mathsf{U}\ \phi \Longleftrightarrow \phi \vee (\psi \wedge \mathsf{X}\ (\psi\ \mathsf{U}\ \phi))
\end{array}$$

The last three equivalences are called fixpoint characterization of temporal operators in terms of “nexttime” operator and “until”. For example the equivalence 5 can be explained by noticing that if $\mathsf{F}\ \phi$ holds in a time line then ϕ holds at the first instant of time or ϕ holds at a future moment and hence $\mathsf{F}\ \phi$ holds in the time line that starts from the next time. On the other hand, if ϕ holds at the first instant of the time line or $\mathsf{F}\ \phi$ holds in the time line that starts from the next instant then obviously $\mathsf{F}\ \phi$ holds in the time line.

An analogous reasoning is possible for G and U operators. This characterization is useful to study the relative expressiveness among temporal logics, in particular with respect to fixpoint calculi. It is possible to express several interesting properties in this logic, in particular that along a time line a property ϕ holds infinitely often ($\mathsf{G}\ \mathsf{F}\ \phi$), which is useful for expressing a typical fairness condition.

1.3 CTL*

In this section we show a branching time logic, namely CTL^*. The syntax of CTL^* allows quantification on paths and states.

We distinguish between state formulas, i.e. formulas that are evaluated on states of the structure, and path formulas, i.e. formulas that are evaluated over paths in the structure.

We define the set of *state* formulas ($\phi, \phi_1, \ldots$) and the set of *path* formulas ($\psi, \psi_1, \ldots$) as the languages generated by the following grammar:

$$\begin{array}{lll}
\phi & ::= & p \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists\psi \mid \forall\psi \\
\psi & ::= & \phi \mid \psi_1 \wedge \psi_2 \mid \neg\psi \mid \mathsf{X}\ \psi \mid \psi_1\,\mathsf{U}\ \psi_2.
\end{array}$$

where $p \in AP$ and AP is a set of propositional symbols. The formal semantics of CTL^* formulas is given w.r.t. a branching time structure $M = (S, R, L)$, where S is a set of states, R is a relation on S and $L : S \longrightarrow 2^{AP}$. A *full path* of a structure M is an infinite sequence $s_0, s_1, \ldots$ of states of M s.t. $\forall i(s_i, s_{i+1}) \in R$. For technical convenience it is assumed in the treatment of this section that the relation R of the structure M is total (this ensures that a full path can always be found). We write $M, s \models \phi$ (respectively $M, x \models \psi$) to mean that the state formula ϕ (respectively the path formula ψ) is true at the state s (respectively in the full path x) of the structure M. The truth relations are inductively defined as follows:

$$
\begin{array}{ll}
(1) & M, s_0 \models p \text{ iff } p \in L(s_0) \\
(2) & M, s_0 \models \phi_1 \wedge \phi_2 \text{ iff } M, s_0 \models \phi_1 \text{ and } M, s_0 \models \phi_2 \\
(3) & M, s_0 \models \neg\phi \text{ iff not } M, s_0 \models \phi \\
(4) & M, s_0 \models \exists\psi \text{ iff } \exists\text{fullpath } x = (s_0, s_1 \ldots) \text{ in } M \text{ s.t. } M, x \models \psi \\
(5) & M, s_0 \models \forall\psi \text{ iff } \forall\text{fullpath } x = (s_0, s_1 \ldots) \text{ in } M \text{ s.t. } M, x \models \psi \\
(6) & M, x \models p \text{ iff } M, s_0 \models p \\
(7) & M, x \models \psi_1 \wedge \psi_2 \text{ iff } M, x \models \psi_1 \text{ and } M, x \models \psi_2 \\
(8) & M, x \models \mathsf{X}\ \psi \text{ iff } M, x^1 \models \psi \\
(9) & M, x \models \psi_1\, \mathsf{U}\, \psi_2 \text{ iff } \exists i (M, x^i \models \psi_2 \text{ and } \forall j (j < i \text{ implies } x^j \models \psi_1))
\end{array}
$$

Following [27] we say that a state formula ϕ (respectively a path formula ψ) is *valid* provided that for every structure M and every state s (respectively fullpath x) we have $M, s \models \phi$ (respectively $M, x \models \psi$). A state formula ϕ (respectively a path formula ψ) is *satisfiable* provided that for some structure M and for some state s (respectively fullpath x) in M we have $M, s \models \phi$ (respectively $M, x \models \psi$).

In figure 1.2 is shown a simple use of the quantification over paths for expressing properties of systems. In particular the formula $\exists\, \mathsf{F}\, \forall\, \mathsf{G}\, p$ expresses the fact that there exists a path that leads the system to a state where in every path which starts in that state the proposition p always holds.

1.3.1 CTL

In this subsection we present a simple branching time logic, namely *Computational Tree Logic* (CTL for short). It is a sublogic of CTL^*. The basic temporal operators of this logic permit to check if a path (or all paths) satisfies a property expressed by a linear time operator whose subformulas can be CTL formulas (so the nesting of temporal operators is forbidden). This restriction and the alternation between branching modalities and linear time make model checking problem (i.e. establishing whether $M, s \models \phi$) for this logic solvable in polynomial time in the dimension of the structure and in the dimension of the formula (see [19]). CTL formulas are generated by the following grammar:

$$
\begin{array}{lcl}
\phi & ::= & p \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \psi \\
\psi & ::= & \phi \mid \exists\mathsf{X}\ \phi \mid \exists\phi_1\, \mathsf{U}\, \phi_2 \mid \forall\mathsf{X}\ \phi \mid \forall\phi_1\, \mathsf{U}\, \phi_2.
\end{array}
$$

Every quantification on path must be tied to a linear time operator. Since the nesting of linear time operators is not allowed it is not possible to express properties like “along some path ϕ holds infinitely often” (which can be expressed by the CTL^* formula ($\exists\, \mathsf{G}\, \mathsf{F}\, \phi$). This limits the expressiveness of this logic also w.r.t. $PLTL$, when $PLTL$ formulas are interpreted over linear-time structures derived from branching time structures.

1.3.2 ECTL*

$ECTL^*$ is an extension of CTL^*, which exploits the expressive power of automata on infinite string (or words, see [98]). Here we follow the treatment of Dam (see [22]). First of all we recall the definition of automata on infinite words.

Definition 1.2 *A $Büchi$ Automaton $\mathcal{A}$ is a $5-$tuple $\langle Q, \Sigma, \longrightarrow, q_0, F\rangle$ where Q is the set of states, Σ is the alphabet, $\longrightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation, q_0 is the initial state and $F \subseteq Q$ is the set of final states.*

A run of $\mathcal{A}$ on an infinite word ($\omega-$word) $\alpha : \mathbb{N} \longrightarrow \Sigma$, is a $\omega-$word $r : \mathbb{N} \longrightarrow Q$ s.t. $r(0) = q_0$ and $r(i) \times \alpha(i) \times r(i+1) \in \longrightarrow$ for every $i \in \mathbb{N}$. Then a $Büchi$ automaton $\mathcal{A}$ *accepts* an $\omega-$word α, if there is a run r of $\mathcal{A}$ and a state $q \in F$ s.t. $r(i) = q$ for infinitely many i and the language recognized by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{\alpha \mid \alpha \text{ is an } \omega - \text{word and } \mathcal{A} \text{ accepts } \alpha\}$.

The syntax of $ECTL^*$ formulas is defined by the following grammar:

$$\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid E(\mathcal{A})$$

where $p \in AP$ and AP is a set of propositional symbols, $\mathcal{A}$ is a $Büchi$ automaton over an alphabet $\Sigma = 2^{\{\phi_1,\ldots,\phi_n\}}$ and ϕ_i is a $ECTL^*$ formula, for $i \in \{1,\ldots,n\}$. It is worthwhile noticing that all formulas are state formulas and the linear-time dependencies are accounted by the automata. The semantics clauses for propositional constants and boolean connectives are the usual ones:

$$s \models p \text{ iff } p \in L(s)$$
$$s \models \phi_1 \wedge \phi_2 \text{ iff } s \models \phi \text{ and } s \models \phi_2$$
$$s \models \neg\phi \text{ iff } \text{ not } s \models \phi.$$

To explain the interpretation of a formula $E(\mathcal{A})$ we need some auxiliary technical definitions. Given a full path $x = x_0, x_1 \ldots$ on a branching time structure M and an automaton $\mathcal{A}$ with alphabet $2^{\{\phi_1,\ldots,\phi_n\}}$ we have the $\omega-$word $trans(x)(i) = \{\phi \in \{\phi_1,\ldots,\phi_n\} \mid x_i \models \phi\}$. The intuition is that the formula $E(\mathcal{A})$ is true in a state s if there is an $\omega-$word in M $x = x_0, x_1 \ldots$ starting from s such that the $\omega-$word over $2^{\{\phi_1,\ldots,\phi_n\}}$ that encodes the satisfaction of $\{\phi_1,\ldots,\phi_n\}$ along x, is accepted from $\mathcal{A}$. More formally:

$$s \models E(\mathcal{A}) \text{ iff } \exists \text{ a fullpath } x \text{ s.t. } (trans(x) \in \mathcal{L}(\mathcal{A})).$$

1.4 HML

In this section we present HML (Hennessy-Milner Logic, see [70]) which is a temporal logic well suited for specification and verification of systems whose behaviour is naturally described by state changes through actions. The syntax of HML formulas is the following:

$$\phi ::= \mathbf{T} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \langle a\rangle\phi$$

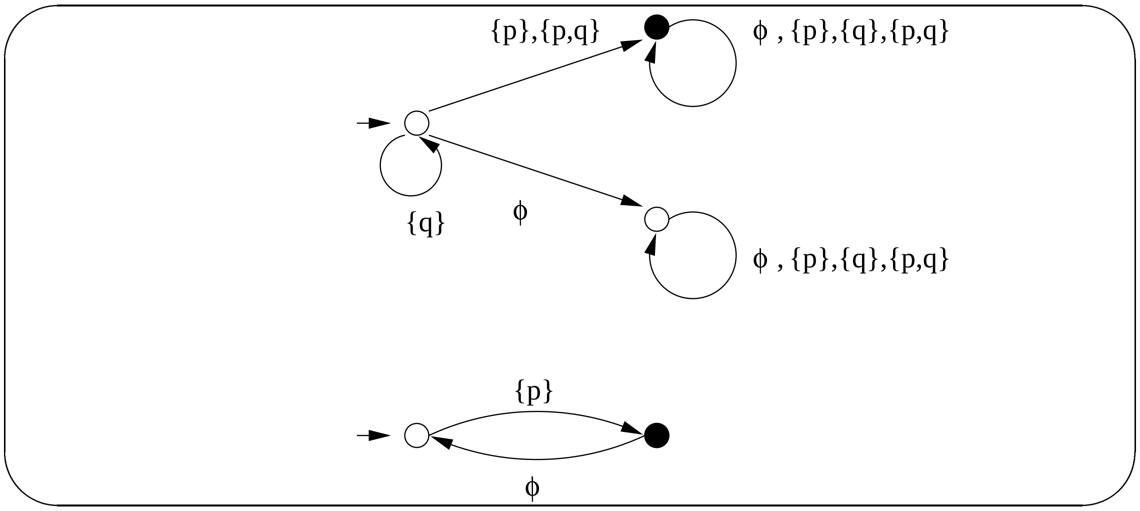


Figure 1.3: The automaton in the upper part accepts infinite paths that satisfy $q \cup p$. The automaton in the lower part accepts infinite paths where p holds only at the even instants of time. Final states are black colored.

where $a \in Act$ and Act is a set of actions (or labels). The basic notion of information is no more an atomic proposition that represents a fact that is true or not in a world, but the notion of action or transition between worlds. The semantics of Hennessy-Milner formulas is given w.r.t. Labeled Transition Systems.

Definition 1.3 *A triple $\langle S, A, \{\xrightarrow{a}\}_{a \in A}\rangle$ is called Labeled Transition System (LTS), where S is a set of states, A is a set of actions and $\{\xrightarrow{a}\}_{a\in A}$ is a set of relations on S.*

LTSs are mathematical objects used to give formal (operational) semantics to concurrent programming languages (see section 2.2 for a discussion).

Informally a formula $\langle a\rangle A$ holds in a state if there exists an a–labeled transition from this state to another one where A holds. Hence it expresses a *possibility*. The formula $[a]A$ holds in a state if A holds in each state that is reachable through an a–labeled transition. Hence it expresses a *necessity*.

The satisfaction for a formula w.r.t. a state of an LTS is defined inductively as follows:

$$\begin{array}{lll} s \models \mathbf{T} & & \text{for every } s \\ s \models \neg\phi & \text{iff} & \text{not } s \models \phi \\ s \models \phi_1 \wedge \phi_2 & \text{iff} & s \models \phi_1 \text{ and } s \models \phi_2 \\ s \models \langle a\rangle\phi & \text{iff} & \exists s'(s \xrightarrow{a} s' \text{ and } s' \models \phi) \end{array}$$

We give a simple example that shows how HML formulas may be used to distinguish between two $LTSs$, whose initial states have a different branching structure. Consider the two $LTSs$ in figure 1.4. In particular let ϕ_1 be $(\langle a\rangle\langle b\rangle\mathbf{T}) \wedge (\langle a\rangle\langle c\rangle\mathbf{T})$ and let ϕ_2 be

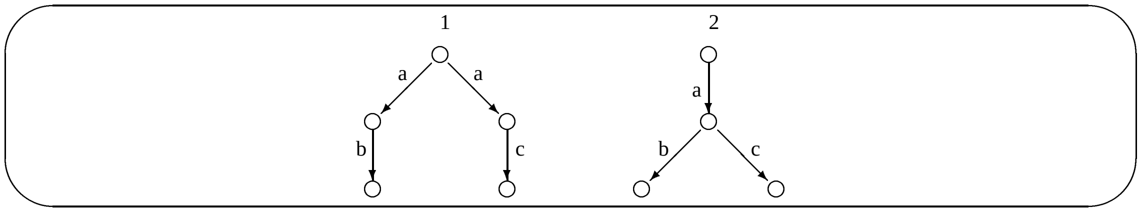


Figure 1.4: The state 2 satisfies ϕ_2, while the state 1 does not satisfy ϕ_2.

$(\langle a\rangle(\langle b\rangle\mathbf{T}\wedge\langle c\rangle\mathbf{T})$. Then the state 2 of the rightmost LTS satisfies ϕ_1 and ϕ_2, while the state 1 of the leftmost LTS satisfies ϕ_1 but not ϕ_2.

1.5 Modal $\mu-$calculus

Modal $\mu-$calculus is a process logic which extends HML with fixpoint operators in order to reason directly about recursive definitions of properties. It permits us to analyze non terminating behaviour of systems.

Let a be in Act, X be a variable ranging over a set of variables $Vars$. Formulas are generated by the following grammar:

$$A ::= X \mid T \mid F \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \langle a\rangle A \mid [a]A \mid \mu X.A \mid \nu X.A$$

We consider the usual definitions of bound and free variables. There is a syntactic restriction on the formulas $\mu X.\alpha(X)$ and $\nu X.\alpha(X)$, namely the variable X must be under the scope of an even number of negations ($\neg$). This restriction is due to the fact that the interpretation of a closed formula A w.r.t. an LTS L is the set of states where A is true. In particular the interpretation of a formula $\alpha(X)$ with a free variable X is a function from set of states to set of states. Hence the interpretation of $\mu X.\alpha(X)$ ($\nu X.\alpha(X)$) is the least (greatest) fixpoint of this function. The syntactic restriction ensures that the interpretation of a formula with free variables, is indeed a monotonic function and so a least (greatest) fixpoint exists.

Formally, given an LTS $\mathcal{M} = \langle S, A, \{\xrightarrow{a}\}_{a\in A}\rangle$ the semantics of a formula A is a subset $[\![A]\!]_\rho$ of the states of $\mathcal{M}$, defined as below, where ρ is a function (called environment) from free variables of A to subsets of the states of $\mathcal{M}$. The environment $\rho[S'/X](Y)$ is equal to $\rho(Y)$ if $Y \neq X$, otherwise $\rho[S'/X](X) = S'$.

$$\begin{array}{rcl}
[\![T]\!]_\rho & = & S \\
[\![F]\!]_\rho & = & \emptyset \\
[\![X]\!]_\rho & = & \rho(X) \\
[\![\neg A]\!]_\rho & = & S \backslash [\![A]\!]_\rho \\
[\![A_1 \wedge A_2]\!]_\rho & = & [\![A_1]\!]_\rho \cap [\![A_2]\!]_\rho \\
[\![A_1 \vee A_2]\!]_\rho & = & [\![A_1]\!]_\rho \cup [\![A_2]\!]_\rho \\
[\![\langle a \rangle A]\!]_\rho & = & \{s | \exists s' : s \xrightarrow{a} s' \text{ and } s' \in [\![A]\!]_\rho\} \\
[\![[a]A]\!]_\rho & = & \{s | \forall s' : s \xrightarrow{a} s' \text{ implies } s' \in [\![A]\!]_\rho\} \\
[\![\mu X.A]\!]_\rho & = & \bigcap\{S' | [\![A]\!]_{\rho[S'/X]} \subseteq S'\} \\
[\![\nu X.A]\!]_\rho & = & \bigcup\{S' | S' \subseteq [\![A]\!]_{\rho[S'/X]}\}
\end{array}$$

Given a model (LTS) $\mathcal{M} = \langle S, A, \{\xrightarrow{a}\}_{a \in A}\rangle$, we write $\mathcal{M}, s \models A$ as notation for $s \in [\![A]\!]_\rho$ when the environment ρ is evident from the context or A is a closed formula (i.e. without free variables). Actually we have presented a slight variant of the propositional $\mu-$calculus as described by Kozen ([49]) or Walukiewicz ([104]). In our treatment, we omit propositional symbols. As usual we consider $\alpha \Longrightarrow \beta$ as an abbreviation for $\neg\alpha \vee \beta$.

1.5.1 Examples and facts

Modal $\mu-$calculus allows to express a lot of interesting properties like *safety* properties (i.e., nothing bad happens) as well as *progress* properties (i.e., something good happens). Moreover equivalence conditions over LTSs may be expressed through this logic (see [4, 91]).

Safety properties are usually defined by means of greatest fixpoint formulas, while *progress* properties by least fixpoint formulas.

An example of *safety* property is the absence of deadlocks in the system, i.e. in any state reachable from the initial one there is always the possibility to perform an action. A $\mu-$calculus formula that expresses this property is $\nu X.[Act]X \wedge \langle Act \rangle T$[1]. Another interesting formula is $\nu X.[K]X \wedge [Act \backslash K]F$ which expresses the fact that only actions in K can be performed by a process in any reachable state.

A *progress* property like "there exists a path for a state which satisfies ϕ" is expressed by $\mu X.\langle Act \rangle X \vee \phi$ (see figure 1.5 for a graphical interpretation of a similar property). Figure 1.6 represents graphically an interpretation of the formula $\mu X.[a]X \vee \langle b \rangle \mathbf{T}$, that expresses that a state where b can be performed is always reached by traversing arcs labeled by a.

Modal $\mu-$calculus can express cyclic properties, e.g. "there exists an infinite path such that the formula ψ is true at all even instants". The $\mu-$calculus formula which expresses the aforementioned property is the following:

$$\nu X.(\langle Act \rangle \langle Act \rangle X \wedge \psi).$$

[1]We use an extended notation, with $K \subseteq Act$ let $[K]A$ be $\bigwedge_{a \in K}[a]A$, and $\langle K \rangle A$ be $\bigvee_{a \in K}\langle a \rangle A$. Since Act is finite the indexed disjunctions (conjunctions) can be expressed by means of disjunction (conjunction).

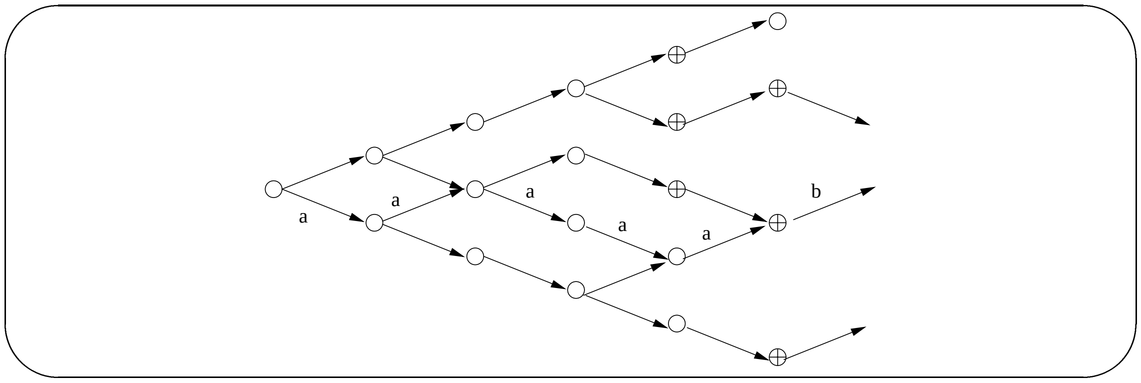


Figure 1.5: The crossed states satisfy $\langle b \rangle \mathbf{T}$. The root of the computation tree satisfies $\mu X. \langle a \rangle X \vee \langle b \rangle \mathbf{T}$.

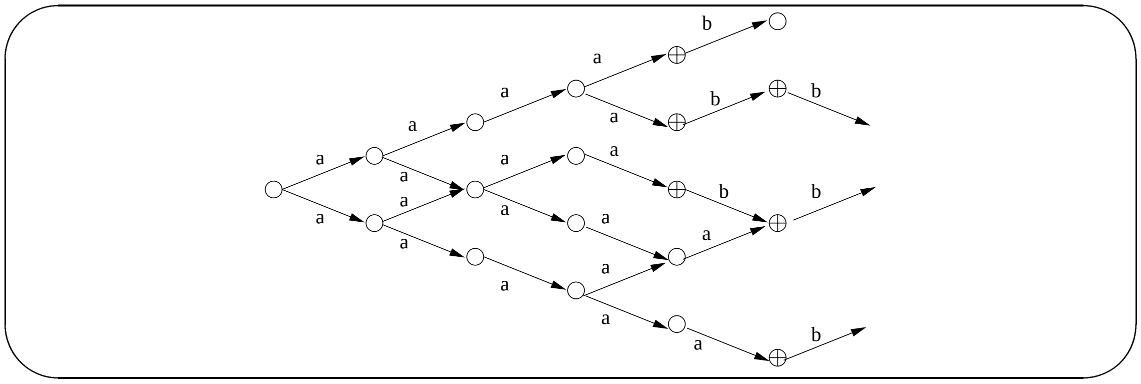


Figure 1.6: The crossed states satisfy $\langle b \rangle \mathbf{T}$. The root of the computation tree satisfies $\mu X. [a] X \vee \langle b \rangle \mathbf{T}$.

The following lemma states some well known facts about $\mu-$calculus (see [21]) that will be used later in the chapter.

Lemma 1.1 *Let ϕ be a $\mu-$calculus formula, ρ an environment and $\sigma \in \{\mu, \nu\}$.*

1. *if X is not free in ϕ then $[\![\sigma X.\phi]\!]_\rho = [\![\phi]\!]_\rho$.*
2. *Let Y be a variable that does not appear free in $\sigma X.\phi$.*
   Then $[\![\sigma X.\phi]\!]_\rho = [\![\sigma Y.\phi[Y/X]]\!]_\rho$.
3. *Let ψ be a formula, then $[\![\phi[\psi/X]]\!]_\rho = [\![\phi]\!]_{\rho[[\![\psi]\!]_\rho/X]}$.*
4. *$[\![\sigma X.\phi]\!]_\rho = [\![\phi[\sigma X.\phi/X]]\!]_\rho$.*

Modal $\mu-$calculus subsumes several temporal logics such as PDL, CTL and CTL^* (see [10, 22]). But, despite its expressiveness, the satisfiability problem (namely finding a structure and a state where the formula holds) still remains EXPTIME-complete (see [94]).

Moreover $\mu-$calculus enjoys the *finite model* property, i.e. if a closed formula is satisfiable then there exists a finite model (a finite state process) for that formula (see [94]).

A finitary axiom system has been proposed by Walukiewicz in [103, 105].

1.6 Deterministic $\mu-$calculus

In the remainder of this thesis we are interested in studying validity problems with respect to specific classes of LTS. In particular, we consider the class of $A-$deterministic LTS (A–det, for short), where A is a set of actions. Roughly, for $A-$deterministic LTS, the relations $\xrightarrow{a}_{a\in A}$ are actually functions. We parameterize this class by means of a set of actions, since this permits us to have a more flexible theory. For example, in chapter 5 we are interested in the investigations of LTS that are deterministic only with respect to a specific action. Let us see formally the definition of these classes.

Definition 1.4 *An LTS L=$\langle S, Act, \{\xrightarrow{a}\}_{a\in Act}\rangle$ is called A–deterministic ($A \subseteq Act$) iff $\forall a \in A \quad \forall s \in S$ if $s \xrightarrow{a} s'$ and $s \xrightarrow{a} s''$ then $s' = s''$.*

The formulas of deterministic $\mu-$calculus are interpreted over A–deterministic LTS. In the remainder of this section we present a slight modification of the deductive system for the $\mu-$calculus proposed by Walukiewicz in [104], for dealing with formulas interpreted over A–det structures.

Walukiewicz's proof system for modal $\mu-$calculus is based on a sequent calculus. A sequent is a couple of sets of formulas, written $\Gamma \vdash \Delta$, that should be logically intended as $\bigwedge_{\gamma\in\Gamma} \gamma \implies \bigvee_{\delta\in\Delta} \delta$ (i.e. the conjunction of premises implies the disjunction of consequences). Let us suppose to have $P \subseteq Act$, hence $\langle P^* \rangle \alpha$ is an abbreviation for $\mu X.\langle P \rangle X \vee \alpha$, and $\langle P^* \rangle \Delta = \{\langle P^* \rangle \delta : \delta \in \Delta\}$.

The proof system $\mathcal{D}_\mu^A$ consists of the following set of axioms and rules:
Axioms:

$$\neg\langle a \rangle \neg\alpha \vdash [a]\alpha \qquad \alpha, \Gamma \vdash \alpha, \Delta \qquad \neg[a]\neg\alpha \vdash \langle a \rangle \alpha$$

$$\nu X.\alpha(X) \vdash \neg\mu X.\neg\alpha(\neg X) \qquad \mu X.\alpha(X) \vdash \neg\nu X.\neg\alpha(\neg X)$$

Rules:

$$(\neg) \qquad \frac{\Gamma \vdash \alpha, \Delta}{\Gamma, \neg\alpha \vdash \Delta} \qquad \frac{\Gamma, \alpha \vdash \Delta}{\Gamma \vdash \neg\alpha, \Delta}$$

$$(\wedge) \qquad \frac{\alpha, \beta, \Gamma \vdash \Delta}{\alpha \wedge \beta, \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \alpha, \Delta \quad \Gamma \vdash \beta, \Delta}{\Gamma \vdash \alpha \wedge \beta, \Delta}$$

$$(\vee) \qquad \frac{\alpha, \Gamma \vdash \Delta \quad \beta, \Gamma \vdash \Delta}{\alpha \vee \beta, \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \alpha, \beta, \Delta}{\Gamma \vdash \alpha \vee \beta, \Delta}$$

$$(\langle a \rangle_{\in A}) \qquad \frac{\alpha, \{\alpha : \langle a \rangle \alpha \in \Gamma\} \cup \{\beta : [a]\beta \in \Gamma\} \vdash \{\gamma : \langle a \rangle \gamma \in \Delta\}}{\langle a \rangle \alpha, \Gamma \vdash \Delta}$$

$$(\langle a \rangle_{\notin A}) \qquad \frac{\alpha, \{\beta : [a]\beta \in \Gamma\} \vdash \{\gamma : \langle a \rangle \gamma \in \Delta\}}{\langle a \rangle \alpha, \Gamma \vdash \Delta}$$

$$(cut) \qquad \frac{\Gamma \vdash \Delta, \alpha \quad \Sigma, \alpha \vdash \Omega}{\Gamma, \Sigma \vdash \Delta, \Omega}$$

$$(\mu) \qquad \frac{\Gamma \vdash \alpha(\mu X.\alpha(X)), \Delta}{\Gamma \vdash \mu X.\alpha(X), \Delta}$$

$$(ind) \qquad \frac{\phi(F) \vdash \Delta \quad \phi(\alpha(\mu X.Z \wedge \alpha(X))) \vdash \Delta, \langle P \rangle \phi(\mu X.Z \wedge \alpha(X))}{\phi(\mu X.\alpha(X)) \vdash \langle P^* \rangle \Delta}$$

where $Z \notin FV(\phi(\mu X.\alpha(X)), \Delta)$

The only difference with the system proposed in [105] is the rule $\langle a \rangle \in A$. We have inserted this rule for reflecting the fact that if $a \in A$ then $\langle a \rangle \phi \wedge \langle a \rangle \phi' \Longrightarrow \langle a \rangle(\phi \wedge \phi')$, is valid over $A-$det structures.

A *proof* for a sequent $\Gamma \vdash \Delta$ is a finite tree, whose root is labeled by $\Gamma \vdash \Delta$, constructed by using the above rules, where leaves are labeled with axioms.

The induction rule (*ind*) is quite cumbersome. Walukiewicz needs this rule for showing the completeness of his finitary axiomatization. Alternatively, one could use the more intuitive rule suggested by Kozen:

$$\frac{\alpha(\phi) \vdash \phi}{\mu X.\alpha(X) \vdash \phi}$$

The next proposition states the soundness of the proposed axiomatization for deterministic $\mu-$calculus.

Proposition 1.1 *All the rules of the proof system $\mathcal{D}_\mu^A$ are sound and the axioms are valid, w.r.t. A–deterministic Labeled Transition Systems.*

Proof: For rules and axioms, excepted $(\langle a\rangle \in A)$, it follows from theorem 3.1.3 of [104] over general structures. Let us study the rule $(\langle a\rangle \in A)$.

By contradiction. We assume that the premise of the rule $(\langle a\rangle \in A)$ is valid and we suppose that the conclusion is not valid. Thus there exists a structure $\mathcal{M} = \langle S, Act, \{\xrightarrow{a}\}_{a\in Act}\rangle$ and a state $s \in S$ such that for all $\gamma \in \Gamma \cup \{\langle a\rangle\alpha\}$ $\quad \mathcal{M}, s \models \gamma$, and for no $\delta \in \Delta$ we have $\mathcal{M}, s \models \delta$.

First we note that $\mathcal{M}, s \models \langle a\rangle\alpha$. Hence it follows that there exists an $a-$successor of s, say s' such that $\mathcal{M}, s' \models \alpha$ and for every formula γ' s.t. $[a]\gamma' \in \Gamma$ we have $\mathcal{M}, s' \models \gamma'$. Since the structure $\mathcal{M}$ is $\{a\}$–deterministic then for every $\langle a\rangle\gamma' \in \Gamma$ if $\mathcal{M}, s \models \langle a\rangle\gamma'$ then $\mathcal{M}, s' \models \gamma'$. By the hypothesis on the premise of the rule $(\langle a\rangle \in A)$, it follows that there exists δ' s.t. $\langle a\rangle\delta' \in \Delta$ and $\mathcal{M}, s' \models \delta'$ (see the rule $(\langle a\rangle \in A)$). This leads to a contradiction since in this case $\mathcal{M}, s \models \langle a\rangle\delta'$ with $\langle a\rangle\delta' \in \Delta$. $\square$

We have proposed a slight variant of Walukiewicz's axiomatization. With this modification all the results in [75] may be restated also for deterministic $\mu-$calculus. Since these results represent steps forward to the proof of the completeness of Walukiewicz's axiomatization, we hope that the variant of the axiomatization we propose is also complete for deterministic $\mu-$calculus. To our knowledge this problem is not yet solved. However we leave the proof of the completeness of our axiomatization as a future work. In fact in this thesis we use deterministic $\mu-$calculus as a technical tool and indeed we are more involved with decidability results of the satisfiability (validity) problem for deterministic $\mu-$calculus, that fortunately can be obtained by means of standard arguments.

We recall the definition of a particular set of formulas, namely the positive guarded ones.

Definition 1.5 *A formula α is called positive iff the only negations which occur in α are applied to $\mathbf{T}$ of $\mathbf{F}$. A variable X in $\mu X.A(X)$ (or $\nu X.A(X)$) is* guarded *iff every occurrence of X in $\alpha(X)$ is in the scope of some modality operator $\langle\rangle, []$. We say that a formula is guarded iff every bound variable in a formula is* guarded.

The next proposition states that we do not loose in generality by restricting our attention to positive guarded formulas.

Proposition 1.2 *(Kozen) Every formula α is equivalent of a positive guarded formula.*

The following result can be proved by putting together standard results for decision procedures for mu-calculi (see [29, 93, 94, 95]).

Theorem 1.1 *Given a positive guarded formula γ it is possible to decide in deterministic exponential time in the length of γ if there exists an A–det structure which is a model of γ.*

1.7 Variants of the modal $\mu-$calculus

In this section we show slight different versions of the modal $\mu-$calculus. These variants have been exploited by many authors (see [5, 10, 55, 92]) because of their technical convenience. In particular these variants allow the sharing of subexpressions, hence the formula which specifies the property can be more concise. For example, in [102] it is given a linear translation from PDL to simultaneous $\mu-$calculus, while the translation from PDL to modal $\mu-$calculus is exponential on the length of the formula. Usually, these variants are used as specification languages in model checking problems.

Bhat and Cleaveland have given a number of translations for several temporal logics (CTL, CTL^* and $ECTL^*$) into an equational variant of $\mu-$calculus (see [10]).

In particular we will use these logics in chapters 2,3 and 5.

1.7.1 Simultaneous fixpoint $\mu-$calculus

The first variant is the simultaneous fixpoint $\mu-$calculus (see [102]) that permits us to reason directly about mutual recursive definitions. Semantically the two formulations are equivalent, namely for every formula of the first language there is a semantically equivalent formula of the second and *vice versa*. The key point is that the translation from simultaneous to modal $\mu-$calculus is exponential in the length of the formula while there is a linear translation from modal to simultaneous $\mu-$calculus (see [4]).

The set of M_μ of simultaneous fixpoint $\mu-$calculus formulas is defined by the following grammar:

$$A ::= X \mid T \mid F \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \langle a \rangle A \mid [a]A \mid (\mu \vec{X}.\vec{A})^i \mid (\nu \vec{X}.\vec{A})^i$$

where $\vec{X}$ is a vector of variables, and $\vec{A}$ is a vector of formulas. There is always the syntactic condition on the nesting of negations of variables.

The only difference w.r.t. modal $\mu-$calculus is the interpretation of fixpoint formulas. A vector $\vec{A}$ of formulas of dimension k with free variables $\vec{X}$ can be seen as a monotonic function from $\mathcal{P}(S)^k$ to $\mathcal{P}(S)^k$ and the $(\mu\vec{X}.\vec{A})^i$ represents the $i-th$ component of the minimal fix point of this function that must exist by Tarski-Knaster theorem (see [97]).

In [94] the authors claim that their solution to the decidability problem for modal $\mu-$calculus leads to the same results for the simultaneous one, by pushing forward the results of [102] where Vardi presents a decidability result for a limited class of formulas, similar to the *disjunctive* ones of Walukiewicz (see [46]).

1.7.2 Equational $\mu-$calculus

Another variant of the modal $\mu-$calculus that permits the sharing of subterms is the equational one, is given by to Andersen (see [4, 5]). Let a be in Act and X be a variable ranging over a finite set of variables $Vars$. Equational $\mu-$calculus is based on fixpoint equations that substitute recursion operators. A minimal (maximal) fixpoint equation is

$X =_\mu A$ ($X =_\nu A$), where A is an assertion, i.e. a simple modal formula without recursion operators. The syntax of the assertions (A) and of the lists of (D) equations is given by the following grammar:

$$\begin{array}{rl} A ::= & X \mid T \mid F \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \langle a \rangle A \mid [a]A \\ D ::= & X =_\nu A\ D \mid X =_\mu A\ D \mid \epsilon. \end{array}$$

It is assumed that variables appear only once on the left-hand sides of the equations of the list, the set of these variables will be denoted as $Def(D)$. Since every equation may be seen as a definition of a variable, the lists of equations are also called definition lists. Let $\langle S, A, \{\xrightarrow{a}\}_{a\in A}\rangle$ be an LTS, ρ be an environment that assigns subsets of S to the variables that appear in the assertions of D, but which are not in $Def(D)$.

The semantics of the assertions is equal to the semantics of corresponding modal $\mu-$calculus formulas. The semantics of a list of equations D, $[\![D]\!]_\rho$[2] is an environment that assigns subsets of S to variables in $Def(D)$. A list of equations is closed if every variable that appears in the assertions of the list is in $Def(D)$. We use $\sqcup$ to represent union of disjoint environments. Let σ be in $\{\mu, \nu\}$, then $\sigma U.f(U)$ represents the σ fixpoint of the function f in one variable U. With the notation $D \downarrow X$, where X is the first variable of the closed list D, we means $[\![D]\!](X)$.

$$\begin{array}{rcl} [\![\epsilon]\!]_\rho & = & [] \\ [\![X =_\sigma AD']\!]_\rho & = & [\![D']\!]_{\rho\sqcup[U'\setminus X]} \sqcup [U'/X] \\ & & \text{where} \\ & & \quad U' \;=\; \sigma U.[\![A]\!]_{(\rho\sqcup[U/X]\sqcup\rho'(U))} \\ & & \quad \rho'(U) \;=\; [\![D']\!]_{(\rho\sqcup[U/X])} \end{array}$$

It informally says that *the solution to $(X =_\sigma A)D$ is the σ fixpoint solution U' of $[\![A]\!]$ where the solution to the rest of the list of equations D is used as environment*.

1.7.3 Another semantics for the equational $\mu-$calculus

Sometimes it is convenient to assume a slight different semantics for definition lists, based on the notion of *block*, i.e. a list of equations with the same fixpoint operator. A definition list D can be partitioned in a list $B_1 \ldots B_n$ of blocks. Now the semantics $[\![B]\!]'_\rho$ of a block $B = (X_1 =_\sigma A_1, \ldots, X_n =_\sigma A_n)$ can be seen as the minimal ($\sigma = \mu$) or maximal ($\sigma = \nu$) fixpoint of a function Φ, i.e.:

$$[\![B]\!]'_\rho(X_i) = \pi_i(\sigma\Phi) \quad i \in \{1, \ldots, n\}$$

where

$$\Phi(U_1, \ldots, U_n) = ([\![A_1]\!]_{\rho\sqcup[U_1/X_1,\ldots,U_n/X_n]}, \ldots, [\![A_n]\!]_{\rho\sqcup[U_1/X_1,\ldots,U_n/X_n]})$$

[2]There is an overloading of the symbol $[\![\ ]\!]$ used as semantic interpretation function for both formulas and lists of equations.

and π_i is a projection on the $i - th$ component of a vector. By the semantics of the assertions, if follows that Φ is monotonic and hence a σ fixpoint exists. Moreover Φ may be seen as a function that given an environment which defines the variables in $Def(D)$ returns an environment with defines the same variables, hence the semantics of a block may be considered as the σ fixpoint of a function from environments to environments. Given a block $B = (X_1 =_\sigma A_1, \ldots, X_n =_\sigma A_n)$ we call $\mathsf{lhs}(\vec{B})$ the vector of variables $[X_1, \ldots, X_n]$ and $\mathsf{rhs}(\vec{B})$ the vector of assertions $[A_1, \ldots, A_n]$. Now we can define the semantics of a vector of assertions in the following way $[\![\vec{A}]\!]'_\rho = ([\![A_1]\!]_\rho, \ldots, [\![A_n]\!]_\rho)$. With abuse of notation we write $X \in \mathsf{lhs}(\vec{B})$ if the variable X appears among the variables in $\mathsf{lhs}(\vec{B})$. For a list of blocks the semantics is defined similarly as for list of equations[3]:

$$\begin{array}{rcl} [\![\epsilon]\!]'_\rho & = & [] \\ [\![B_1, B]\!]'_\rho & = & [\vec{U}'/\mathsf{lhs}(\vec{B}_1)] \sqcup [\![B]\!]'_{\rho \sqcup [\vec{U}'/\mathsf{lhs}(\vec{B}_1)]} \\ & & \text{where} \\ & & \begin{array}{rcl} \vec{U}' & = & \sigma \vec{U}.[\![\mathsf{rhs}(\vec{B}_1)]\!]'_{(\rho \sqcup [\vec{U}/\mathsf{lhs}(\vec{B}_1)] \sqcup \rho'(\vec{U}))} \\ \rho'(\vec{U}) & = & [\![B]\!]'_{(\rho \sqcup [\vec{U}/\mathsf{lhs}(\vec{B}_1)])} \end{array} \end{array}$$

The following theorem can be used to show that the simultaneous fixed point operators do not increase the expressive power of the $\mu-$calculus (see [4, 7] for a full account of the theorem).

Theorem 1.2 *(Bekič's theorem) Let D and E be two complete lattices, and $f : D \times E \longrightarrow D$ and $g : D \times E \longrightarrow E$ monotonic functions, then:*

$$\mu(x, y).(f(x, y), g(x, y)) = (\mu x.f(x, \mu y.g(x, y)), \mu y.g(\mu x.f(x, y), y)).$$

Given a definition list $E = (X_1 =_{\sigma_1} A_1, \ldots, X_i =_{\sigma_i} A_i, \ldots, X_j =_{\sigma_j} A_j, \ldots, X_n =_{\sigma_n} A_n)$, with $E_{i,j}$ we indicate the list $(X_i =_{\sigma_i} A_i, \ldots, X_j =_{\sigma_j} A_j)$.

The two semantics agree for a definition list which can be seen as a single block (see [21]).

Lemma 1.2 *Given a definition list $E = (X_1 =_\sigma A_1, \ldots, X_n =_\sigma A_n)$ and an environment ρ then:*

$$[\![E]\!]_\rho = [\![E]\!]'_\rho$$

Furthermore, it is possible to prove:

[3]There is an overloading of the symbol $[\![\ ]\!]'$ used as semantic interpretation function for both vectors of assertions and lists of blocks.

Lemma 1.3 *Given a definition list $E = (B_1, \dots, B_l)$, where $B_i \quad i \in \{1, \dots, l\}$ are blocks, then for every environment ρ we have:*

$$[\![E]\!]_\rho = [\![E]\!]'_\rho.$$

Proof: By induction on the length of the list E.

- $\mid E \mid = 1$. Trivial,
- $\mid E \mid = n$. By induction on $i \subseteq \{1, \dots, n\}$.
  - $i = 1$. Suppose that B_1 is equal to $(X_1 =_{\sigma_1} A_1, B_1')$, with $B_1' = (X_2 =_{\sigma_1} A_2, \dots X_{b_1} =_{\sigma_1} A_{b_1})$. The case where $|B_1| = 1$ is immediate. Now we have $[\![E]\!]_\rho(X_1) = \sigma_1 U_1.[\![A_1]\!]_{\rho \sqcup [U_1/X_1] \sqcup \rho'(U_1)}$, where $\rho'(U_1) = [\![E_{2,n}]\!]_{\rho \sqcup [U_1/X_1]}$. Thus we can consider the list E as a list of blocks, and X_1 belongs to $\mathsf{lhs}(B_1)$, hence we have:

    $$[\![E]\!]'_\rho(X_1) = \pi_1(\sigma_1 \vec{U}.[\![\mathsf{rhs}(\vec{B_1})]\!]'_{\rho_2})$$

    where $\rho_2 = \rho \sqcup [\vec{U}/\vec{X}] \sqcup \rho_1(\vec{U}),\ \rho_1(\vec{U}) = [\![B_2, \dots, B_l]\!]'_{\rho \sqcup [\vec{U}/\vec{X}]}$, and $\mathsf{lhs}(\vec{B_1}) = \vec{X}$. By applying the $Beki\check{c}$'s theorem for binary fixpoints to $\sigma_1 \vec{U}.[\![\mathsf{rhs}(\vec{B_1})]\!]'_{\rho_2}$ we get that:

    $$[\![E]\!]'_\rho(X_1) = \sigma_1 U_1.[\![A_1]\!]_{\rho \sqcup [\vec{e}/\vec{X}] \sqcup \rho_1(\vec{e})}$$

    where:

    $$e_i = \begin{cases} \pi_i(\sigma_1 U_2 \dots U_{b_1}.[\![\mathsf{rhs}(\vec{B_1'})]\!]'_{\rho_2}) & i \in \{2, \dots, b_1\} \\ U_1 & i = 1 \end{cases}$$

    Now we prove the following equality for every U_1, from which the thesis follows:

    $$\rho \sqcup [U_1/X_1] \sqcup \rho'(U_1) = \rho \sqcup [\vec{e}/\vec{X}] \sqcup \rho_1(\vec{e}) \tag{1.1}$$

    For the variables defined in ρ and for X_1 the above equality is true. Note that for each variable X_i with $i \in \{2, \dots b_1\}$, we have $e_i = [\![B_1', \dots, B_l]\!]'_{\rho \sqcup [U_1/X_1]}(X_i)$ by the definition of the semantics of the list of blocks. Since by inductive hypothesis on n we know:

    $$[\![B_1', \dots, B_l]\!]'_{\rho \sqcup [U_1/X_1]} = [\![E_{2,n}]\!]_{\rho \sqcup [U_1/X_1]} \tag{1.2}$$

    it follows that for every variable X_i with $i \in \{2, \dots b_1\}$ the equality 1.1 holds. Now consider a variable Y in $Def(B_2, \dots, B_l)$. Then we have $\rho_1(\vec{e})(Y) = [\![B_2, \dots, B_l]\!]'_{\rho \sqcup [\vec{e}/X]}(Y)$. Now we have:

    $$\begin{array}{lcl} [\![B_2, \dots, B_l]\!]'_{\rho \sqcup [\vec{e}/\vec{X}]}(Y) & = & \text{by induction on } n \\ [\![B_2, \dots, B_l]\!]_{\rho \sqcup [\vec{e}/\vec{X}]}(Y) & = & \text{from 1.2 for variables } X_i, i \in \{2, \dots, b_1\} \\ [\![B_1', \dots, B_l]\!]_{\rho \sqcup [U_1/X_1]}(Y) & & \end{array}$$

from which follows that the equality 1.1 is true.

– $i = i'+1$. Without loss of generality we can suppose that X_i is the first variable in $\mathsf{lhs}(\vec{B_j})$ for a block B_j. Otherwise one can apply the $Beki\check{c}$'s theorem in the appropriate way. Then there is a proof analogous to the one above, in particular by noticing that by inductive hypothesis we have for every ρ':

$$[\![B_1, \ldots, B_{j-1}]\!]'_{\rho'} = [\![E_{1,i'}]\!]_{\rho'}.$$

□

A translation from equational $\mu-$calculus to modal $\mu-$calculus

In this section we give a translation from equational to modal $\mu-$calculus. Given a list of equations D with n equations, the translation is the following:

$$\begin{array}{rcl} tr(D_{1,1}) & = & \sigma_1 X_1.A_1 \\ tr(D_{1,n}) & = & tr(D_{1,(n-1)}[\sigma_n X_n.A_n/X_n]) \end{array}$$

where $D[\sigma X.A/X]$ is the list D where every occurrence of X is replaced by $\sigma X.A$ (we assume that the list of equations are well named). In the following lemma, ρ is supposed not to be defined for variables in $Def(D)$. Moreover the semantics of equational $\mu-$calculus must be consistently updated to deal with fixpoints in the assertions. A similar lemma is stated in [4].

Lemma 1.4 *For every variable $X \in Def(D_{1,(n-1)})$, and for every environment ρ we get:*

$$([\![D_{1,n}]\!]_\rho)(X) = ([\![D_{1,(n-1)}[\sigma_n X_n.A_n/X_n]]\!]_\rho)(X).$$

Given a closed definition list D, then by applying $n-1$ times the above lemma we can prove that $D \downarrow X_1 = ([\![D_{1,n}]\!])(X_1) = [\![tr(D_{1,n})]\!]$.

Chapter 2

Structural operational semantics and partial evaluation

In this chapter we introduce a formal method for giving (operational) semantics to programs, namely *Structural Operational Semantics* (SOS, for short) proposed by Plotkin (see [79]). This method has a logical flavor and permits to reason compositionally about the behaviour of programs. As example language, whose semantics is defined in a SOS style, we present the *Calculus of Communicating Systems* (CCS, for short [69, 70]). This language belongs to the family of *process algebras* ([43, 44]), i.e. formalisms for the description of concurrent communicating processes. This kind of languages and the systems they allow to describe will be used later in this thesis. Furthermore we rephrase the compositional analysis techniques proposed in [4, 5, 55] for languages described in a SOS style.

2.1 Structural Operational Semantics

The method we are going to present is based on the notion of (Labeled) Transition System. The states of the transition system are the elements of some formal language. The main notion is the transition between states, i.e. $s \xrightarrow{a} s'$, which expresses the fact that the system s has performed a step of an activity, identified by an action or label a, by reaching a state s'. In general, transitions between terms can be inferred through a set of (conditional) rules, based on the syntax of the language. The transition $s \xrightarrow{a} s'$ can be derived by inspecting the transitional behaviour of subcomponents of s. In the wide range of formats for SOS rules the general structure is the following:

$$\frac{\text{premises}}{\text{conclusions}}$$

where *premises* and *conclusions* are properties expressed on transitions of open terms of the language. The semantics of terms can be inferred by the semantics of the subterms. Many important facts about programming languages, whose operational semantics is given in terms of SOS rules, can be deduced simply by inspecting the format of these rules. For example, it is possible to ensure that an equivalence relation among closed terms is indeed a congruence w.r.t. the operators of the language (see [12]). Moreover a lot of work has been carried out in order to automatically derive from SOS specifications for (concurrent) programming languages complete equational theories (see [3]). In this chapter we show how to apply compositional reasoning techniques for a generic language defined by a system of SOS rules.

2.1.1 Preliminaries

Let V be a countable set of variables, ranged over by $x, y \ldots$, possibly subscripted. Let Act be a finite set of actions, ranged over by $a, b, c \ldots$, possibly subscripted.

Definition 2.1 *A* signature Σ *is a pair* (F, ar)*, where:*

- F *is a set of function symbols, disjoint from* V*,*
- $ar : F \mapsto \mathbb{N}$ *is a* rank function *which gives the arity of a function symbol; if* $f \in F$ *and* $ar(f) = 0$ *then* f *is called a* constant symbol.

Definition 2.2 *Let* Σ *be a signature. Let* $W \subseteq V$ *be a set of variables. The set of* $\Sigma-$*terms over* W*, with notation* $T(\Sigma, W)$*, is the least set satisfying:*

- $W \subseteq T(\Sigma, W)$*,*
- *if* $f \in F$ *and* $t_1, .., t_{ar(f)} \in T(\Sigma, W)$*, then* $f(t_1, .., t_{ar(f)}) \in T(\Sigma, W)$*.*

When $W = \emptyset$, $T(\Sigma, \emptyset)$ is abbreviated by $T(\Sigma)$ and $T(\Sigma, V)$ is abbreviated by $\mathsf{T}(\Sigma)$; elements from $T(\Sigma, \emptyset)$ are called *closed* or *ground terms*, elements from $\mathsf{T}(\Sigma)$ are called *open terms*.

Definition 2.3 *Given a signature* $\Sigma = (F, ar)$*, an assignment is a function* γ *from* V *to* $T(\Sigma)$*. An assignment* γ *induces straightforwardly a mapping from terms to* $T(\Sigma)$*:*

$$\gamma(f(t_1, \ldots, t_{ar(f)})) = f(\gamma(t_1), \ldots, \gamma(t_{ar(f)})).$$

Given a term t, let $Vars(t)$ ($Mvars(t)$) be the set (multiset) of variables in t. A term t is *closed* if $Vars(t) = \emptyset$.

De Simone's Format

Perhaps the first who has clearly identified the notion of format is De Simone in [89]. A rule in the De Simone's format is as follows:

$$\frac{\{x_i \xrightarrow{a_i} y_i\}_{i \in I}}{f(x_1, \ldots, x_k) \xrightarrow{a} t} \tag{2.1}$$

where:

- $f \in F$, $ar(f) = k$ and $I \subseteq \{1, \ldots, k\}$,
- $x_1, \ldots, x_k$ and y_i for $i \in I$ are distinct variables,
- moreover, if we define the variables $x'_i = y_i$ if $i \in I$ and $x'_i = x_i$ otherwise, then each variable x'_i occurs almost once in t and $Vars(t) \subseteq \{x'_1, \ldots, x'_k\}$.

Now we give the way of calculating whenever a closed term t can perform a transition $t \xrightarrow{a} t'$.

Definition 2.4 *Let $\mathcal{D}$ be a set of rules in De Simone's format. A* proof *for a transition $t \xrightarrow{a} t_1$ is a well founded upward branching tree, whose nodes are labeled by transitions $t' \xrightarrow{a'} t''$ s.t.:*

- *the root is labeled with $t \xrightarrow{a} t_1$,*
- *if $t' \xrightarrow{a'} t''$ is the label of a node q and χ_i for $i \in I$ is the set of labels of nodes directly above q then there is a rule in $\mathcal{D}$:*

$$\frac{\{x_i \xrightarrow{a_i} y_i\}_{i \in I}}{f(x_1, \ldots, x_k) \xrightarrow{a} t'''} \tag{2.2}$$

and an assignment γ s.t. $\gamma(x_i) \xrightarrow{a_i} \gamma(y_i) = \chi_i$ for all $i \in I$ and $\gamma(f(x_1, \ldots, x_k)) \xrightarrow{a} \gamma(t''') = t' \xrightarrow{a'} t''$.

A simple language

Let us give a simple language defined by means of rules in the above format, which permits us to describe finite labeled trees. The signature is composed by a constant $\mathbf{0}$, the choice operator $+$ with arity 2 (we use the infix notation), and a finite set of prefix operators $a.$ where $a \in Act$ and Act is a finite set of actions. For every prefix operator $a.$ there is an axiom, i.e. a rule without premises:

$$\frac{}{a.x \xrightarrow{a} x}(p_a)$$

and moreover we have two more rules for the choice operator (for each $a \in Act$):

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'}(+^l_a) \quad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}(+^r_a)$$

There are no rules for $\mathbf{0}$ since it represents a process that cannot perform any action (terminated or deadlocked process). The following is a simple proof for the transition $(a.c.\mathbf{0} + \mathbf{0}) + b.\mathbf{0} \xrightarrow{a} c.\mathbf{0}$.

$$\cfrac{\cfrac{\cfrac{}{a.c.\mathbf{0} \xrightarrow{a} c.\mathbf{0}}\, p_a}{(a.c.\mathbf{0} + \mathbf{0}) \xrightarrow{a} c.\mathbf{0}}\, +^l_a}{(a.c.\mathbf{0} + \mathbf{0}) + b.\mathbf{0} \xrightarrow{a} c.\mathbf{0}}\, +^l_a$$

GSOS Format

We recall the definition of GrandSOS or GSOS format of Bloom *et al.* in [3, 12, 13], by following the treatment proposed by Simpson in [90].

A $GSOS$ rule r has the following format:

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij}\}_{1\le j\le m_i}^{1\le i\le k} \quad \{x_i \not\xrightarrow{b_{ij}}\}_{1\le j\le n_i}^{1\le i\le k}}{f(x_1,\ldots,x_k) \xrightarrow{c} g(\vec{x},\vec{y})} \tag{2.3}$$

where all variables are distinct; $\vec{x}$ and $\vec{y}$ are the vectors of all x_i and y_{ij} variables respectively; $m_i, n_i \geq 0$ and k is the arity of f. We say that f is the *operator* of the rule ($op(r) = f$) and c is the action. A GSOS system $\mathcal{G}$ is given by a signature and a finite set of GSOS rules.

Given a signature $\Sigma = (F, ar)$, an assignment γ is *effective* for a term $f(s_1,\ldots,s_k)$ and a rule r[1] if:

1. $\gamma(x_i) = s_i$ for $1 \leq i \leq k$;
2. for all i, j with $1 \leq i \leq k$ and $1 \leq j \leq m_i$, it holds that $\gamma(x_i) \xrightarrow{a_{ij}} \gamma(y_{ij})$;
3. for all i, j with $1 \leq i \leq k$ and $1 \leq j \leq n_i$, it holds that $\gamma(x_i) \not\xrightarrow{b_{ij}}$.

The transition relation among closed terms can be defined in the following way: we have $f(s_1,\ldots,s_n) \xrightarrow{c} s$ if and only if there exists an *effective* assignment γ for a rule r with operator f and action c such that:

$$s = \gamma(g(\vec{x},\vec{y}))$$

Bloom *et al.* in [13] proved that there exists a unique transition relation induced by a GSOS system. Moreover another interesting peculiarity of this transition relation is that it is *finitely branching*, namely for every closed term s we have $\{s' : \exists a \in Act \quad s \xrightarrow{a} s'\}$ is finite.

Given a rule r in the GSOS format we call $source(r) = \{x_i \mid 1 \leq i \leq k\}$, $target(r) = g(\vec{x},\vec{y})$. For every rule r and for every variable x_i in $source(r)$ let $Pos(r, x_i)$ be $\{a_{ij} \mid 1 \leq j \leq m_i\}$ and $Neg(r, x_i)$ be $\{b_{ij} \mid 1 \leq j \leq n_i\}$. For every rule r and for every variable x_i in $source(r)$ let $Succ(r, x_i)$ be $\{y_{ij} \mid 1 \leq j \leq m_i\}$.

2.2 Process Algebras

In this section we briefly recall CCS (or *Calculus of Communicating Systems*) of Milner, a language defined in the SOS style for describing concurrent systems. This formalism is called *process algebra* since it has a mathematical and algebraic flavor.

[1]In the rest of this chapter when we refer to a generic rule r, we consider it in the format in 2.3.

Rather than actual programming languages, *process algebras* are specification formalisms for system that have to cooperate and communicate to perform complex tasks and computations in different settings and in different contexts. The universe of interest is modeled by assuming the notion of *processes* that autonomously and concurrently can proceed in their computation but which have also the possibility to communicate and synchronize among themselves. *Process algebra* formalisms are built from the basic operations of this framework. The processes can perform actions (which may represent computation steps).

Let us see the CCS *process algebra*[2]. The main operator is the parallel composition between processes, namely $p||q$. The intuition is that the parallel composition of two processes performs an action whenever anyone of the two processes performs an action. Moreover, processes can communicate. The notion of communication considered is a synchronous one, namely both the processes must agree on performing the communication at the same time. In CCS the communication between two processes composed in parallel is modeled by a simultaneous performing of complementary actions. This event is represented by a synchronization action (or internal action) τ.

The complementary actions can be seen as $sending$ and $receiving$ activities on the same channel.

The signature of CCS is defined as follows. Given a finite set of actions $\mathcal{L}$, the set of complementary actions $\overline{\mathcal{L}}$ is $\{\overline{a} \mid a \in \mathcal{L}\}$ where $^{-}$ is a bijection with $\overline{\overline{a}} = a$. Let Act be $\mathcal{L} \cup \overline{\mathcal{L}} \cup \{\tau\}$, where τ is a special action that denotes an internal computation step (or communication). Π is a set of constant symbols that can be used to define processes with recursion.

$$F_{CCS} = \{\mathbf{0}, +, ||\} \cup \{a. \mid a \in Act\} \cup \{\backslash L \mid L \subseteq \mathcal{L} \cup \overline{\mathcal{L}}\} \cup \{[f] \mid f : \mathcal{L} \cup \overline{\mathcal{L}} \mapsto \mathcal{L} \cup \overline{\mathcal{L}}\} \cup \Pi.$$

We have $ar(\mathbf{0}) = 0$ and for every $\pi \in \Pi$ we have $ar(\pi) = 0$. The symbols $||$ and $+$ denote binary operators and we will use the infix notation for them. The symbol $a.$ denotes a unary operator and we use the prefix notation for it. The $[f]$ and $\backslash L$ symbols denote unary operators and we use the postfix notation for them. Let Σ_{ccs} be (F_{CCS}, ar). The operational semantics of the CCS closed terms is given by means of the GSOS system in figure 2.1, where for every action $a \in Act$ we have the rules $p_a, +^l_a, +^r_a, ||^l_a, ||^r_a$; for every subset L of $\mathcal{L} \cup \overline{\mathcal{L}}$ and for every action a not in $L \cup \overline{L}$ we have a rule $res_{a,L}$; for every function $f : \mathcal{L} \cup \overline{\mathcal{L}} \mapsto \mathcal{L} \cup \overline{\mathcal{L}}$ and action a' s.t. $a' = f(a)$ we have a rule $rel_{a,f}$. Moreover for every action $l \in \mathcal{L} \cup \overline{\mathcal{L}}$ we have a com_l rule.

Informally a (closed) term $a.p$ represents a process that performs an action a and then behaves as p.

The term $p + q$ represents the non deterministic choice between the processes p and q. After the choice has been resolved, by choosing the action of one of the two components, the other is dropped.

[2] Actually we present a slight different formulation of Milner's CCS, since we deal with the restrictions of GSOS format. In particular we do not assume an infinitary choice operator and we have only a finite set of actions. Moreover instead of a recursion construct like $recX.P$ we assume an infinite set of constants.

$$\frac{}{a.x \xrightarrow{a} x}(p_a)$$

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'}(+^l_a) \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}(+^r_a)$$

$$\frac{x \xrightarrow{a} x'}{x||y \xrightarrow{a} x'||y}(||^l_a) \qquad \frac{y \xrightarrow{a} y'}{x||y \xrightarrow{a} x||y'}(||^r_a)$$

$$\frac{x \xrightarrow{l} x' \quad y \xrightarrow{\bar{l}} y'}{x||y \xrightarrow{\tau} x'||y'}(com_l)$$

$$\frac{x \xrightarrow{a} x'}{x\backslash L \xrightarrow{a} x'\backslash L}(res_{a,L}) \qquad \frac{x \xrightarrow{a} x'}{x[f] \xrightarrow{a'} x'[f]}(rel_{a,f})$$

Figure 2.1: GSOS system for CCS.

The term $p||q$ represents the parallel composition of the two processes p and q. The rules $||^l_a$ and $||^r_a$ show that the compound term $p||q$ can perform an action if one of the two can perform an action, and this does not prevent the capabilities of the other process. The rule com is characteristic of this calculus, it expresses that the communication between processes happens whenever both can perform complementary actions. The resulting process is given by the parallel composition of the successors of each component, respectively.

The process $p\backslash L$ behaves like p but the actions in $L \cup \overline{L}$ are forbidden. To force a synchronization on an action between parallel processes, we have to use the restriction operator in conjunction with the parallel one.

The process $p[f]$ behaves like p but the actions are renamed *via* f.

Examples

Some examples follow of systems described in CCS. As a first simple example we show how to serialize the activity of two processes that run concurrently. Since in general the $||$ operator does not impose a precedence among the activities of processes, i.e. if p can perform an action a and q can perform and action b then their parallel composition can perform both the a and b actions. We need some way to synchronize the two activities. Suppose we want to give precedence to the first process. After the first process has performed its activity (modeled by an execution of an action a) then a complementary action is performed, namely $\overline{start}$. The other process, before starting its activity (say b) waits for a complementary action of $start$. To force the synchronization on this action we prevent the whole process to perform $start$ actions. Let us review this small system

$(a.\overline{start}.\mathbf{0}||start.b.\mathbf{0})\backslash\{start\}$. Its behaviour is described by the following LTS:

$$\begin{tikzcd}
(a.\overline{start}.\mathbf{0}||start.b.\mathbf{0})\backslash\{start\} \arrow[d, "a"'] \\
(\overline{start}.\mathbf{0}||start.b.\mathbf{0})\backslash\{start\} \arrow[d, "\tau"'] \\
(\mathbf{0}||b.\mathbf{0})\backslash\{start\} \arrow[d, "b"'] \\
(\mathbf{0}||\mathbf{0})\backslash\{start\}
\end{tikzcd}$$

By following [39] we treat recursion by considering an infinite set of constant symbols Π. Every constant must be equipped with a constant definition, say $\{\pi \doteq t_\pi \mid \pi \in \Pi, t \in T(\Sigma_{CCS})\}$. If $\pi \doteq t$ is a definition of constant then this means that the behaviour of the process π is given by the behaviour of its body t_π (it is worthwhile noticing that the body can contain other constant symbols). Formally this is expressed by adding the following rule:

$$\frac{t_\pi \xrightarrow{a} t'}{\pi \xrightarrow{a} t'}$$

Now suppose that we want to model a system where a user sends a file for printing to a spooler, which in turn chooses the printer that will actually do this job and returns this information to the user. We have four agents, the user represented by a constant U, the spooler S and the two printers, say P_1 and P_2. Let us see their CCS specifications:

$$\begin{aligned}
U &\doteq Preparing.\overline{SendSpooler}.(Printed1.U_1 + Printed2.U_2)\\
S &\doteq SendSpooler.(\overline{SendPrinter1}.Finished.\overline{Printed1}.S+\\
&\qquad\qquad\qquad\quad \overline{SendPrinter2}.Finished.\overline{Printed2}.S)\\
P1 &\doteq SendPrinter1.Printing1.\overline{Finished}.P1\\
P2 &\doteq SendPrinter2.Printing2.\overline{Finished}.P2
\end{aligned}$$

where U_1 and U_2 are the continuations of the process User, after it has received the information about the Printer actually used. The system consists of the parallel composition of the four elements, with the restriction on the communication actions. We can observe that the set of actions:

$$L' = \{SendPrinter1, SendPrinter2, Finished\}$$

regards only communications between the Spooler and the printers, while $SendSpooler$, $Printed1$, $Printed2$ are communication actions between the User and the Spooler. So the system may be modeled in the following way:

$$Sys = (U||(S||(P_1||P_2)\backslash L'))\backslash\{SendSpooler, Printed1, Printed2\}.$$

The behaviour of the composed system, determined by the GSOS system, is represented in figure 2.2. In the figure we have omitted the terms of the transition system, for example

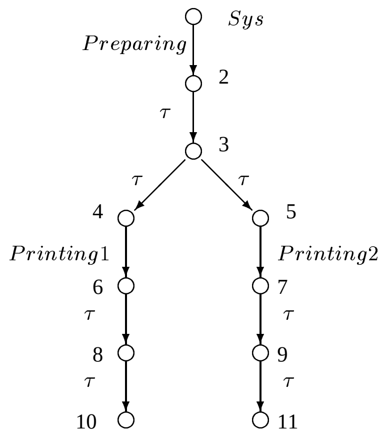


Figure 2.2: The LTS of the process Sys, for sake of simplicity we have named only the interesting states of the LTS.

the term relative to the state labeled with 4 is:

$$((Printed1.U_1 + Printed2.U_2)\|((Finished.\overline{Printed1}.S\| \\ (Printing1.\overline{Finished}.P1\|P2))\backslash L')\backslash\{SendSpooler, Printed1, Printed2\}.$$

while the term of the state 10 is:

$$(U_1\|(S\|(P1\|P2))\backslash L')\backslash\{SendSpooler, Printed1, Printed2\}.$$

Auxiliary transition relations

Let us define some auxiliary transition relations, that will be used in the next sections.

Definition 2.5 *Let $t = a_1 \dots a_n \in Act^*$ and $a \in Act \setminus \{\tau\}$. Then*

$$\begin{array}{l}
\xrightarrow{t} = \xrightarrow{a_1} \dots \xrightarrow{a_n} \\
\overset{\tau}{\Longrightarrow} = \xrightarrow{\tau}^{*} \\
\overset{a}{\Longrightarrow} = \overset{\tau}{\Longrightarrow}\xrightarrow{a}\overset{\tau}{\Longrightarrow} \\
\longrightarrow = \bigcup_{a\in Act} \xrightarrow{a}
\end{array}$$

Given a process p let $Der(p) = \{q \mid p \longrightarrow^* q\}$ be the set of its derivatives. A process p is said finite-state if $Der(p)$ is finite.

2.2.1 Equivalences

In general, it is interesting to study when two processes (terms) can be considered equivalent, by abstracting from irrelevant aspects. The consideration of relevant aspects, mainly depends on which way the process is used. Due to the huge number of different settings that arise in the analysis of concurrent systems, not surprisingly, many different theories of equivalence have been proposed in literature.

Labeled Transition Systems are used as models for reactive systems, hence we have to consider not the *internal* state of the system but the *behavioural* capacity to react with the outside world. Even though there is a general agreement on the *behavioral* nature of equivalence, it is not always reasonable to take into account the same aspects as the relevant ones. For example, it is commonly accepted that the two terms $a.\mathbf{0}$ and $a.\mathbf{0} + a.\mathbf{0}$ (where $\mathbf{0}$ is constant process that does nothing and $+$ is the CCS choice operator) should be regarded as equivalent, since both can perform an a action and then idle forever. But some researchers have argued that in same specific situation they should be considered different. In fact if we take into account the availability of resources in fault tolerant systems, we could consider the first process less "robust" than the other since it has only one possible resource to perform the a action, while the second seems to have two resources. Hence, it is important to identify which properties of the system must be analyzed, and which properties are preserved by a certain equivalence notion. Here, we briefly introduce some well known equivalences among processes (for a deeper discussion see [13, 23, 24, 100]). A first attempt to give a reasonable notion of equivalence that fits the general case may be the following:

p and q are trace equivalent iff $\forall t \in Act^*$ we have $p \xrightarrow{t} \Longleftrightarrow q \xrightarrow{t}$.

This equivalence, even though rather intuitive, is not completely satisfactory from several points of view. Concurrent systems may present deadlocks, i.e. the system cannot proceed and cannot perform its task. The above equivalence does not take into account deadlocks, i.e. the following terms are considered equivalent: $a.\mathbf{0} + a.b.\mathbf{0}$ and $a.b.\mathbf{0}$. The first can perform an a action and then reaches a deadlocked configuration, while the other one cannot perform an a action by reaching a deadlock state.

Another negative consideration about trace equivalence is that it does not take in account the branching structure of the processes. In fact the following processes are trace equivalent: $a.b.\mathbf{0} + a.c.\mathbf{0}$ and $a.(b.\mathbf{0} + c.\mathbf{0})$. Their associated LTSs are presented in figure 2.3. The first process "chooses" to perform a b or c action at the beginning of its computation, while the latter after performing an a action; if this action can influence the choice of the following behaviour of the process then it is reasonable to consider that the second process has a more decisional power.

A possible solution is to imagine a game-like equivalence such as:

p and q are equivalent iff if $p \xrightarrow{a} p'$ then $q \xrightarrow{a} q'$ and p' and q' are equivalent and *vice versa*.

More formally we can define the notion of *strong bisimulation* by following Park (see [76]).

Definition 2.6 *A relation $\mathcal{R}$ between states of an LTS $L = \langle S, A, \{\xrightarrow{a}\}_{a \in A}\rangle$ is a strong bisimulation if for each $(p, q) \in \mathcal{R}$ and for each $a \in A$:*

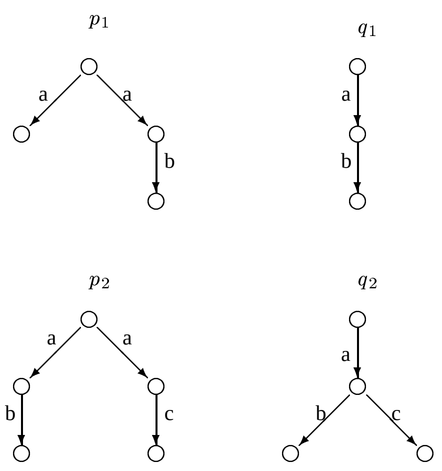


Figure 2.3: Example of trace equivalent processes.

if $p \xrightarrow{a} p'$ *then there exists* $q' : q \xrightarrow{a} q'$ *and* $(p', q') \in \mathcal{R}$.

if $q \xrightarrow{a} q'$ *then there exists* $p' : p \xrightarrow{a} p'$ *and* $(p', q') \in \mathcal{R}$.

Two processes p and q are strong bisimilar if there exists a strong bisimulation $\mathcal{R}$ s .t. $(p, q) \in \mathcal{R}$. The maximal strong bisimulation is $\sim$ which is the union of every strong bisimulation. It is easy to check that this relation is still a strong bisimulation and moreover is reflexive, symmetric and transitive.

From the definition, it appears clear that the processes p_1 and q_1 are not strong bisimilar. Analogously the processes p_2 and q_2 are not strong bisimilar. In fact q_2 after an a action can reach a state where both b and c actions can be performed, while p_2, after performing an a action, can be in a state where b can be performed (but not c) or else in a state where c can be performed (but not b).

Strong bisimulation is the finest equivalence that is commonly accepted and enjoys several good properties. First of all, this equivalence is also a congruence w.r.t. all CCS operators and moreover Bloom *et al.* proved that strong bisimulation is preserved by all $GSOS$ definable operators.

Theorem 2.1 *([13]) Let $\mathcal{G}$ be a GSOS system. Then strong bisimulation is a congruence w.r.t. the operations in $\mathcal{G}$.*

Another important aspect of bisimulation is that it can be logically characterized. In fact if we consider as $\mathcal{L}_{HML}$ the set of HML logic formulas (see section 1.4) we have:

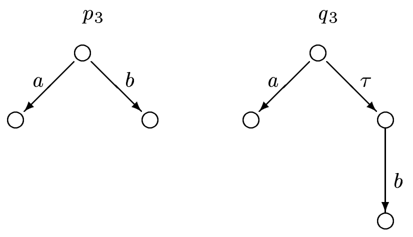
Proposition 2.1 *([70]) If p and q are finitely branching processes then*

$$p \sim q \text{ iff } \forall \phi \in \mathcal{L}_{HML}(p \models \phi \Longleftrightarrow q \models \phi).$$

It is interesting to note that the previous proposition is true also for $\mu-$calculus formulas. This strong connection between modal logics and models of our concurrent languages is one of the major advantages in considering interleaving semantics for concurrency w.r.t. other models that take into account more realistic and peculiar aspects of concurrency such as Petri nets and event structures. This connection plays a central role in our thesis and in the next chapters, we show how results from temporal logic can be applied in concurrency and results in concurrency can lead to new developments and ideas in temporal logic theory.

Observational equivalence or bisimulation

Until now we do not have assumed a distinguished role for the τ action, but actually this action is used to model a communication internal to the system or an internal computation step (not visible to the outside world). So we would like to abstract from those actions when comparing two systems. This can be fruitfully exploited in a step-wise development strategy because we are able to substitute concise specifications by larger ones without affecting the overall visible behaviour of the system. For example, we can imagine to substitute a process with two others that perform the same visible task, but committing some internal communication. Actually, we cannot simply abstract the internal actions since they can affect the visible behaviour of a system. Look at the following example:



The processes p_3 and q_3 cannot be considered equivalent, since the second can perform an internal action (and so independently of the environment) by reaching a state where an action a is no longer possible. So the non visible behaviour of the system, represented by the τ action, can modify its visible behaviour.

The equivalence proposed by Milner (see [69]), namely *observational equivalence* or *bisimulation* (or else *weak bisimulation*) is the following:

Definition 2.7 *A relation $\mathcal{R}$ between states of an LTS $L = \langle S, A, \{\xrightarrow{a}\}_{a\in A}\rangle$ is a bisimulation if for each $(p,q) \in \mathcal{R}$ and for each $a \in A$:*

if $p \xrightarrow{a} p'$ then there exists $q' : q \overset{a}{\Longrightarrow} q'$ and $(p',q') \in \mathcal{R}$.

if $q \xrightarrow{a} q'$ then there exists $p' : p \overset{a}{\Longrightarrow} p'$ and $(p',q') \in \mathcal{R}$.

Two processes p and q are bisimilar if there exists a bisimulation $\mathcal{R}$ s.t. $(p,q) \in \mathcal{R}$. The maximal bisimulation is $\approx$ which is the union of every bisimulation. It is easy to check that this relation is still a bisimulation and moreover is reflexive, symmetric and transitive.

Moreover bisimulation is a congruence w.r.t. all CCS operators, excepted for summation (+).

The bisimulation is a very interesting equivalence. It is decidable in polynomial time for finite-state processes (see [47]). Moreover quite elegant proof techniques exist for proving that two processes p and q are bisimilar. Actually, it is sufficient to provide a bisimulation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$. In this way we can prove bisimilarity among possibly infinite state processes (see example 5.6).

2.3 Partial evaluation of requirements

In general in the construction of complex systems it is not convenient to build directly the precise and complete description of every component. It is better to follow a *top down* design methodology. It permits to detect in an earlier stage of the development possible errors of the project and repair them, without wasting too much time. Moreover the management of the whole project is easier.

In chapter 1 we have seen how properties of LTS can be expressed in a process logic. We have seen that concurrent programs (i.e. processes) can be described through $LTSs$. In the previous sections we have described a way for formally deriving the behavior of a system, in a compositional manner, by inspecting of the behaviour of its subcomponents.

Since it is quite unusual to have all the requirements at the beginning of a project it is common to leave unspecified some part of the system. We would like not to specify too many things in an earlier stage of the development. Moreover, the specification of components should not lead to a system whose subcomponents cannot be provided, because of the already specified parts do not permit any possible implementation of the unspecified ones. These problems have been tackled in several settings, in particular Larsen and Xinxin (see [55]) and Andersen (see [4, 5]) have proposed an automated methodology to find the minimal properties that unspecified components must satisfy in order that the whole system can verify a certain property. We show how their ideas can be rephrased by starting from GSOS definitions of languages.

The intuitive idea

First, we define the formal language in which the system properties are specified. For the simplicity of operators and for its significance in the theory of LTS many authors argue that $\mu-$calculus is a good specification language. Here we start our analysis from the sublogic HML that, as we have seen in the previous section, has characterizing power w.r.t. strong bisimulation.

We believe that the intuitive idea can be understood better in this limited framework. Moreover the presence of fixpoint operators introduce some technical problems. We briefly present how this approach works and the problems that must be tackled.

Consider a simple CCS term, i.e. $t = a.x$ where x is a variable. Hence, we want to get the minimal requirements on closed terms s, that can be substituted to x in the term t, s.t. $t[s/x] \models (\langle a \rangle \mathbf{T}) \wedge ([a]\langle b \rangle \mathbf{T})$. First of all we note, by observing the interpretation of the logical connective $\wedge$, that this problem is equivalent to analyze the two subproblems 1) $t[s/x] \models \langle a \rangle \mathbf{T}$ and 2) $t[s/x] \models [a]\langle b \rangle \mathbf{T}$. Let us study separately the two problems:

1. By looking at the definition of the $GSOS$ rule for prefixing operator in figure 2.1, we note that we can have $a.s \xrightarrow{a} s$ for every closed term s. Hence, the condition on the terms s is $\mathbf{T}$.

2. By observing the semantics of the $[a]$ modality, we can note that it requires that for every $a-$successor of $t[s/x]$ holds $\langle b \rangle \mathbf{T}$. By looking again at the definition of the $GSOS$ rule for prefix operator in figure 2.1, we can have only an $a-$transition from $a.s$ in s. Then s must satisfy $\langle b \rangle \mathbf{T}$.

By putting together the conditions on s we get $t[s/x] \models (\langle a \rangle \mathbf{T}) \wedge ([a]\langle b \rangle \mathbf{T})$ iff $s \models \mathbf{T} \wedge \langle b \rangle \mathbf{T}$.

Let us consider the following CCS term $t = a.\mathbf{0} + x$ and the formula $[a][a]\mathbf{F}$. As before we have to ensure that every reachable term from $t[s/x]$ through an a action must satisfy $[a]\mathbf{F}$, (that is to say that it cannot perform an a action). By looking at the definition of the $GSOS$ rule for the choice operator in figure 2.1 we can have that $t[s/x] \xrightarrow{a} \mathbf{0}$, by means of the transition $a.\mathbf{0} \xrightarrow{a} \mathbf{0}$, or else $s \xrightarrow{a} s'$ for some s'. In the former case there are no requirements on s (since $\mathbf{0} \models [a]\mathbf{F}$) and in the latter it is required that $s' \models [a]\mathbf{F}$. By summarizing, the latter property can be equivalently expressed as $s \models [a][a]\mathbf{F}$.

The analysis proceeds in a compositional manner in the structure of the formula and the only interesting case is the treatment of the capabilities of processes. Since the formula is finite and the possible successors of each term are finite, in the case of HML formulas this strategy always terminates.

2.3.1 Theoretical framework for the partial evaluation

We will consider a particular format of $GSOS$ rules, i.e. the kind of rules that do not permit copying.

Definition 2.8 *A rule r of a GSOS system $\mathcal{G}$ is not copying if and only if for every variable z in $source(r)$ the set $(Succ(r,z) \cup \{z\}) \cap Vars(g(\vec{x},\vec{y}))$ is empty or it is a singleton $\{w\}$ and in this case we have $Mvars(g(\vec{x},\vec{y}))(w) \leq 1$.*

The necessity of restricting ourselves to this kind of rules relies on the fact that we want that the number of unspecified components of the system cannot grow. This is clear if we look at the following rule:

$$\frac{x \xrightarrow{a} x' \qquad x \xrightarrow{b} x''}{f(x) \xrightarrow{a} g(x', x'')}$$

If we have a term with one unspecified component, say $f(x)$, it is possible that after a transition a we have to deal with a term with two unspecified components, since we cannot predict the successors of x. Even though it is possible to deal in some framework (see [55]) with more than one unspecified component, we prefer to restrict ourselves to this situation. Indeed, this analysis is sufficient for our purposes. Another property that we impose to the format of the rules is the following:

Definition 2.9 *A rule r of a GSOS system $\mathcal{G}$ keeps the nesting of terms if every variable in $target(r)$ appears as argument of the top level operator, or target(r) is a variable.*

The following rule shows an example of nesting:

$$\frac{x \xrightarrow{a} x'}{f(x) \xrightarrow{a} f(f(x'))}$$

Assumption 2.1 *We start our analysis under the assumption that every rule in $\mathcal{G}$ is not* copying *and keeps the nesting.*

The following rule violates both conditions:

$$\frac{x \xrightarrow{fork} x'}{f(x) \xrightarrow{\tau} parent(x')||child(x')}$$

As notation we write $t[\vec{x}]$ where t is a term and $Vars(t)$ is a subset of the variables in $\vec{x}$. We write $t[z]'$ if the variable z may appear in t at most once, and as argument of the top level operator or $t = z$.

Remark 2.1 *In general for a context $t[z]'$ we use a particular notation where z is equal to the variable that is the formal parameter in the rules for the top level operator of t (we implicitly assume that for every rule and every operator f always the same variables and in the same order in the source are used)*[3].

Let us make some requirements on the contexts $t[z]'$.

Since we treat terms with one possible unspecified component, we are interested in partial assignments for a rule r and a context $t[x]'$, that do not need to be defined for all variables involved by the rule r. More formally a *partial effective* assignment γ for a rule r and a context $f(t_1, \ldots, x_h, \ldots t_k)$ is a function from variables to closed terms, not defined for the variables in $\{x_h\} \cup Succ(r, x_h)$ such that:

1. $\gamma(x_i) = t_i$ for $1 \leq i \leq k, i \neq h$;
2. for all i, j with $1 \leq i \leq k, i \neq h$ and $1 \leq j \leq m_i$, it holds that $\gamma(x_i) \xrightarrow{a_{ij}} \gamma(y_{ij})$;
3. for all i, j with $1 \leq i \leq k, i \neq h$ and $1 \leq j \leq n_i$, it holds that $\gamma(x_i) \stackrel{b_{ij}}{\not\longrightarrow}$.

[3]Alternatively one can assume as equality among context the α–equivalence.

$$1 \quad \frac{x \xrightarrow{a} x' \qquad x \xrightarrow{b} x''}{f(x) \xrightarrow{c} g}$$

$$2 \quad \frac{y \xrightarrow{a} y'}{x \| y \xrightarrow{a} x \| y'}$$

$$3 \quad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'}$$

Figure 2.4: Examples of rules of types 1,2 and 3 with respect to the variable x.

If the context $t[x]'$ is actually a closed term then a *partially effective* assignment is *effective*. Let $\mathcal{PE}^{r}_{t[x]'}$ be the set of *partially effective* assignments for a rule r and a context $t[x]'$. Let $Rules^{op(t)}_{a}$ be the subset of rules of $\mathcal{G}$ whose operator is $op(t)$, and whose action is a. For the sake of simplicity, if $t[x]'$ is a closed term then let $pos(r, x) = neg(r, x) = \emptyset$. This notion permits us to give a definition of the transition relation between contexts. We observe:

- $t[x]' \rightsquigarrow t'[x']'$ iff $\exists \gamma'$ *partially effective* for the rule r and the context $t[x]'$ such that $t'[x']' = \gamma'(target(r))$.

Definition 2.10 *The set of derivatives w.r.t. the above transition relation for a context* $t[x]'$ *is* $DerC(t[x]') = \{t'[y]' \mid t[x]' \rightsquigarrow^{*} t'[y]'\}$.

In the sequel we will restrict ourselves to the class of *well behaved* contexts.

Definition 2.11 *A context* $t[x]'$ *is well-behaved* iff $DerC(t[x]')$ *is a finite set.*

Please note that if in the previous definition we do not assume a unique representation for contexts then we could have an infinity of derivatives of the term $t[z]$ simply renaming z.

By assumption 2.1 on the nature of the rules, we can divide them among three types w.r.t. a variable x in the source of a rule r:

- Type 1: $(\{x\} \cup Succ(r, x)) \cap Vars(target(r)) = \emptyset$;
- Type 2: $x \in Vars(target(r))$;
- Type 3: $Succ(r, x) \cap Vars(target(r)) \neq \emptyset$.

The next technical lemma shows an alternative characterization of the transition relation between closed terms.

Lemma 2.1 *Given a context $t[x]'$ and a closed term s we have $t[s/x]' \xrightarrow{a} t'$* iff *one of the following conditions holds:*

- *$\exists r$ of type 1 s.t. $\exists\gamma$ partially effective for r and $t[x]'$ and $\forall c \in Pos(r,x) : s \xrightarrow{c}$ and $\nexists c \in Neg(r,x) : s \xrightarrow{c}$, and $\gamma(target(r)) = t'$,*
- *$\exists r$ of type 2 s.t. $\exists\gamma$ partially effective for r and $t[x]'$ and $\forall c \in Pos(r,x) : s \xrightarrow{c}$ and $\nexists c \in Neg(r,x) : s \xrightarrow{c}$, s.t. $t' = (\gamma(target(r)))[s/x']'$,*
- *$\exists r$ of type 3 s.t. $\exists\gamma$ partially effective for r and $t[x]'$ and $\forall\ c \in Pos(r,x) : s \xrightarrow{c}$ and $/\exists c \in Neg(r,x) : s \xrightarrow{c}$, let y_{ij} be in $Succ(r,x) \cap Vars(target(r))$ then $\exists s' : s \xrightarrow{a_{ij}} s'$ and $t' = (\gamma(target(r)))[s'/x']'$.*

Proof:

Suppose that $t[s/x]' \xrightarrow{a} t'$. By definition there must be a rule r with operator $op(t)$ and action a and an *effective* γ' for r and $t[s/x]'$ s.t. $t' = \gamma'(target(r))$. We know that r must be of one of the three types $1, 2$ or 3. So we continue by inspection of the type of r. First of all we note that in every case, since γ' is *effective*, we must have $\forall\ c \in Pos(r,x) : s = \gamma'(x) \xrightarrow{c}$ and $\nexists c \in Neg(r,x) : s = \gamma'(x) \xrightarrow{c}$. Please note that if $Pos(r,x)$ or $Neg(r,x)$ are empty the conditions are trivially fulfilled. As notation we write $\gamma \setminus [X]$ for the assignment that coincides with γ with the exception of the variables in X, where it is undefined.

Then let us consider the different types of rules:

1. Let γ be $\gamma' \setminus [\{x\} \cup Succ(r,x)]$, then γ is partially effective.

2. In this case we have that x appears in $Vars(target(r))$, so $\gamma'(x)$ is defined and $t' = \gamma'(target(r))$, let γ be $\gamma' \setminus [\{x\} \cup Succ(r,x)]$ then $t' = \gamma'(target(r)) = (\gamma(target(r)))[s/x']')$.

3. In this case we have that y_{ij} appears in $Succ(r,x) \cap Vars(target(r))$, then by hypothesis on effectiveness of γ' we have $s = \gamma'(x) \xrightarrow{a_{ij}} \gamma'(y_{ij}) = s'$. As above consider $\gamma = \gamma' \setminus [\{x\} \cup Succ(r,x)]$ then $t' = \gamma'(target(r)) = (\gamma(target(r)))[s'/x']')$.

The other direction can be proved by similar arguments. $\square$

The following proposition states the correctness of the partial evaluation function σ given in figure 2.5[4].

[4]We use indexed disjunctions and conjunctions, with the usual semantics, in particular $\bigvee_{i\in\emptyset} = \mathbf{F}$ and $\bigwedge_{i\in\emptyset} = \mathbf{T}$.

$$\begin{aligned}
\sigma(x[x]', A) &= A \\
\sigma(t[x]', \mathbf{T}) &= \mathbf{T} \\
\sigma(t[x]', A_1 \wedge A_2) &= \sigma(t[x]', A_1) \wedge \sigma(t[x]', A_2) \\
\sigma(t[x]', \neg A) &= \neg \sigma(t[x]', A) \\
\sigma(t[x]', \langle a \rangle A) &= \bigvee_{r \in Rules_a^{op(t)}} \bigvee_{\gamma \in \mathcal{PE}^r_{t[x]'}} ((\bigwedge_{c \in Neg(r,x)} \neg \langle c \rangle \mathbf{T}) \wedge \\
&\qquad (\bigwedge_{c \in Pos(r,x)} \langle c \rangle \mathbf{T}) \wedge \sigma'(r, x, A))
\end{aligned}$$

where $W = Succ(r, x) \cap Vars(target(r))$ in

$$\sigma'(r, x, A) = \begin{cases} \sigma(\gamma(target(r)), A) & r \text{ of type 1 or 2} \\ \langle a_{ij} \rangle \sigma(\gamma(target(r)), A)) & r \text{ of type 3 and } y_{ij} \in W \end{cases}$$

Figure 2.5: Partial evaluation function for HML logic.

Proposition 2.2 *Given a* well behaved *context* $t[x]'$*, and an HML formula* A*, then for every closed term* s *we have:*

$$t[s/x]' \models A \text{ iff } s \models \sigma(t[x]', A).$$

Proof: By structural induction on A:

- $A = \mathbf{T}$, we have $t[s/x]' \models \mathbf{T}$ for every closed term s, or equivalently $s \models \mathbf{T}$.
- $A = A_1 \wedge A_2$, we have $t[s/x]' \models A$ iff $t[s/x]' \models A_1$ and $t[s/x]' \models A_2$. By induction hypothesis we have that $t[s/x]' \models A_1$ iff $s \models \sigma(t[x]', A_1)$, and $t[s/x]' \models A_2$ iff $s \models \sigma(t[x]', A_2)$. Hence we have $t[s/x]' \models A$ iff $s \models \sigma(t[x]', A_1)$ and $s \models \sigma(t[x]', A_2)$, by definition of the semantics of $\wedge$, this leads to $s \models \sigma(t[x]', A_1) \wedge \sigma(t[x]', A_2)$.
- $A = \neg A_1$, we have $t[s/x]' \models A$ iff $t[s/x]' \not\models A_1$, and by inductive hypothesis the latter statement is equivalent to $s \not\models \sigma(t[x]', A_1)$ and equivalently $s \models \neg\sigma(t[x]', A_1)$.
- $A = \langle a \rangle A_1$, we have $t[s/x]' \models \langle a \rangle A_1$ iff there exists t' s.t. $t[s/x]' \xrightarrow{a} t'$ and $t' \models A_1$. By using the lemma 2.1 we can have one of the following cases:
  - $\exists r$ of type 1 s.t. $\exists \gamma$ partially effective for r and $t[x]'$ and $\forall\, c \in Pos(r, x)$: $s \xrightarrow{c}$ and $\not\exists c \in Neg(r, x) : s \xrightarrow{c}$, and $\gamma(target(r)) = t' \models A_1$,
  - $\exists r$ of type 2 s.t. $\exists \gamma$ partially effective for r and $t[x]'$ and $\forall\, c \in Pos(r, x)$: $s \xrightarrow{c}$ and $\not\exists c \in Neg(r, x) : s \xrightarrow{c}$, s.t. $t' = (\gamma(target(r)))[s/x']'$, and $t' \models A_1$,

– $\exists r$ of type 3 s.t. $\exists \gamma$ partially effective for r and $t[x]'$ and $\forall\ c \in Pos(r,x)$: $s \xrightarrow{c}$ and $\nexists c \in Neg(r,x) : s \xrightarrow{c}$, let y_{ij} be in $Succ(r,x) \cap Vars(target(r))$ then $\exists s' : s \xrightarrow{a_{ij}} s'$ and $t' = (\gamma(target(r)))[s'/x']'$ and $t' \models A_1$.

The above conditions can be equivalently expressed by:

– $\exists r$ of type 1 s.t. $\exists \gamma$ partially effective for r and $t[x]'$ and $s \models (\bigwedge_{c \in Neg(r,x)} \neg\langle c\rangle \mathbf{T}) \wedge \bigwedge_{c \in Pos(r,x)} \langle c\rangle \mathbf{T})$ and $t' \models A_1$. By induction hypothesis $t' \models A_1$ is equivalent to $s \models \sigma(t', A_1)$.

– $\exists r$ of type 2 s.t. $\exists \gamma$ partially effective for r and $t[x]'$ and $s \models (\bigwedge_{c \in Neg(r,x)} \neg\langle c\rangle \mathbf{T}) \wedge \bigwedge_{c \in Pos(r,x)} \langle c\rangle \mathbf{T})$ and $t' \models A_1$. In this case $t' = (\gamma(target(r)))[s/x']'$, and by inductive hypothesis we have $t' \models A_1$ iff $s \models \sigma(\gamma(target(r)), A_1)$.

– $\exists r$ of type 3 s.t. $\exists \gamma$ partially effective for r and $t[x]'$ and $s \models (\bigwedge_{c \in Neg(r,x)} \neg\langle c\rangle \mathbf{T}) \wedge \bigwedge_{c \in Pos(r,x)} \langle c\rangle \mathbf{T})$ and $t' \models A_1$. In this case, we have $s \xrightarrow{a_{ij}} s'$ and $t' = (\gamma(target(r)))[s'/x']'$, and hence by inductive hypothesis we have $t' \models A_1$ iff $s' \models \sigma(\gamma(target(r)), A_1)$.
So we have another condition $s \models \langle a_{ij}\rangle \sigma(\gamma(target(r)), A_1)$.

By grouping together the similar conditions w.r.t. s, we obtain the desired formula. $\square$

2.4 Adding recursion to the logical language

In chapter 1 we have seen that the HML logic, without recursion, does not permit to express properties on infinite computation of a system. Thus here we extend the previous theory to handle this kind of properties. We use the equational $\mu-$calculus as specification language[5].

Given a well behaved context $t[x]'$ let C be the set of its derivatives $DerC(t[x]')$. Since $t[x]'$ is well behaved then C is finite, in particular we assume in the following $C = \{c_1, \ldots, c_n\}$.

We start with a little example that highlights the necessity of using equational $\mu-$calculus, instead of modal $\mu-$calculus. Let us consider the $\mu-$calculus formula $F = \mu X.(\langle a\rangle X \vee \langle b\rangle \mathbf{T})$. Since we want to perform the analysis in a compositional way in the structure of the formula we need the translation of the subformula $(\langle a\rangle X \vee \langle b\rangle \mathbf{T})$ w.r.t. a context $t[z]'$. During this translation it is possible that the same variable X (or better the properties expressed by a variable) must be analyzed simultaneously in different contexts!

In fact, suppose to have the CCS process $P = a.a.P + a.c.\mathbf{0}$. Now consider the context $t = (P \| y)$. Hence we get:

$$DerC(t[y]') = \{P\|y, a.P\|y, c.\mathbf{0}\|y, \mathbf{0}\|y\}.$$

[5]Actually, we use an equational $\mu-$calculus with negation in assertions, since it has a smaller number of operators and modalities. There is always the syntactic condition on the nesting of negations.

Then the formula $(\langle a \rangle X \vee \langle b \rangle \mathbf{T})$ must be reduced, in particular we have to reduce the two subformulas $\langle a \rangle X$ and $\langle b \rangle \mathbf{T}$. Let us concentrate on the former formula. It follows that if we apply the standard reduction for $\langle a \rangle X$ we should get:

$$\langle a \rangle \sigma((P \| y), X) \vee \sigma((a.P \| y), X) \vee \sigma((c.\mathbf{0} \| y), X)$$

where $\sigma(t[y]', X)$ is the evaluation of a variable in a context (for a formal definition see below).

Hence the properties expressed by X must be "simultaneously" evaluated in three different contexts.

As Larsen and Xinxin, we avoid the problem of considering effectively every reachable context, simply by inserting an equation for every context! Andersen applies this strategy only for the parallel operator of his generic process algebra. For the other operators he is able to analyze correctly only the strictly necessary part of the system. The price he has to pay is a more complex translation function.

So every equation in the list E is substituted by a list of equations, one for every context in the derivatives of $t[z]'$.

Let $tr^C(E)$ our syntactic translation function from definition lists to definition lists.

$$tr^C(E) = \begin{cases} [] & E = [] \\ X^{c_1} =_{\sigma_1} \sigma(c_1, A), \ldots, X^{c_n} =_{\sigma_1} \sigma(c_n, A), tr^C(E') & E = \ (X =_{\sigma_1} A), E' \end{cases}$$

moreover we add the following clause for logical variables to the partial evaluation function of figure 2.5.

$$\sigma(t[x]', Y) = Y^{t[x]'}.$$

Our goal is to prove that:

$$c[s/z]' \in [\![E]\!]_\rho(Y) \ \text{ iff } \ s \in [\![tr^C(E)_\rho]\!](Y^c) \tag{2.4}$$

In particular we are interested in the case where $\rho = \perp$. Hence, the above statement says that checking if a system satisfies an equational specification, when the unspecified component is replaced by a closed term, is equivalent to check if the closed term satisfies a translated equational specification.

First we introduce a semantic counterpart of the $tr^C(E)$ translation. Let $\mathcal{TR}^C(\rho)(X^c) = \{s \mid c[s/z]' \in \rho(X)\}$ be a semantical translation function from environments to environments.

In the following proof we will use a technical lemma proved by Andersen in [4], the so called *reduction* lemma.

Lemma 2.2 *Suppose that D and E are powersets over countable sets, and $in : D \mapsto E$ an $\omega-$continuous function with $in(\perp_D) = \perp_E$. Moreover suppose to have $g : E \mapsto E$ and $f : D \mapsto D$ both monotonic and with the property:*

$$in \circ f = g \circ in.$$

We can conclude that

$$\mu g = in(\mu f).$$

The in function can be seen as a decomposition function. Given a set of contexts, it returns for every context the set of closed terms that fit in the unspecified component (see definition in the proof below). The *reduction* lemma is used to reason about simultaneous evaluations of $\mu-$calculus variables, which are evaluated in different contexts.

Instead of proving the equivalence 2.4 we prove the following result from which 2.4 follows.

Lemma 2.3 *Let* E *be a list of equations,* ρ *an environment and* $C = DerC(t[x]')$, *then we have:*

$$\mathcal{TR}^C([\![E]\!]_\rho) = [\![tr^C(E)]\!]_{\mathcal{TR}^C(\rho)}.$$

Proof: Let in be:

$$\begin{array}{rcl} in(U) & = & (in_{c_1}(U), \dots, in_{c_n}(U)) \quad \text{with} \\ & & in_c(U) \; = \; \{s \mid c[s/z]' \in U\} \;\; \text{for } c \in C. \end{array}$$

We prove the thesis by induction on the length n of the definition list.

- $\mid E \mid = 0$. The result trivially follows.
- $\mid E \mid = \mid (X_1 \; =_{\sigma_1} \; A_1), E' \mid = n$. By induction on $i \in \{1, \dots, n\}$ and for $c \in C$ we prove that:

  $$\mathcal{TR}^C([\![E]\!]_\rho)(X_i^c) = [\![tr^C(E)]\!]_{\mathcal{TR}^C(\rho)}(X_i^c)$$

  - $i = 1$. We can see that $\mathcal{TR}^C([\![E]\!]_\rho)(X_1^c) = \pi_c(in([\![E]\!]_\rho(X_1)))$, where π_c is a projection.

    Let $f(U_1)$ be $[\![A_1]\!]_{\rho \sqcup [U_1/X_1] \sqcup \rho'(U_1)}$ where $\rho'(U_1) = [\![E']\!]_{\rho \sqcup [U_1/X_1]}$. Now we have $[\![E]\!]_\rho(X_1) = \sigma_1 U_1.f(U_1)$. We assume $\sigma_1 = \mu$ in the sequel, otherwise one can use a *reduction* lemma for maximal fixpoints.

    Let $g(U_1, \dots, U_n)$ be $([\![\sigma(c_1, A_1)]\!]_{\rho_1}, \dots, [\![\sigma(c_n, A_1)]\!]_{\rho_1})$ where $\rho_1 = \mathcal{TR}^C(\rho) \sqcup [U_1/X_1^{c_1}, \dots, U_n/X_1^{c_n}] \sqcup \rho''(U_1, \dots, U_n)$ and $\rho''(U_1, \dots, U_n)$ is equal to $[\![tr^C(E')]\!]_{\mathcal{TR}^C(\rho) \sqcup [U_1/X_1^{c_1}, \dots, U_n/X_1^{c_n}]}$. Hence we can observe that

    $$\mathcal{TR}^C(\rho \sqcup [U_1/X_1]) = \mathcal{TR}^C(\rho) \sqcup [in_{c_1}(U_1)/X_1^{c_1}, \dots, in_{c_n}(U_1)/X_1^{c_n}]. \quad (2.5)$$

    By inductive hypothesis on the length of E we have:

    $$\mathcal{TR}^C([\![E']\!]_{\rho \sqcup [U_1/X_1]}) = [\![tr^C(E')]\!]_{\mathcal{TR}^C(\rho \sqcup [U_1/X_1])}.$$

We can look at the minimum fixpoint of g as the *block* semantics in the equational $\mu-$calculus for the variables $X_1^{c_1}, \ldots, X_1^{c_n}$. In particular let $B = tr^C(X_1 =_\mu A_1)$ then

$$\mu g = \mu U_1 \ldots U_n . [\![\mathsf{rhs}(\vec{B})]\!]'_{\rho_1}$$

Thus for $c \in C$, we get $\pi_c(\mu g) = [\![B, tr^C(E')]\!]'_{\mathcal{TR}^C(\rho)}(X_1^c)$. Hence, by lemma 1.3 we have:

$$\pi_c(\mu g) = [\![tr^C(E)]\!]'_{\mathcal{TR}^C(\rho)}(X_1^c) = [\![tr^C(E)]\!]_{\mathcal{TR}^C(\rho)}(X_1^c)$$

In order to prove the thesis it is sufficient to prove $in([\![E]\!]_\rho(X_1)) = in(\mu f) = \mu g$. By using the *reduction* lemma we can restrict ourselves to prove:

$$in(f(U)) = g(in(U)).$$

By structural induction on A_1.

* $A_1 = X_1$, then $in(f(U)) = in(U)$, while $g(in(U))$:
  $$\begin{array}{ll} ([\![X_1^{c_1}]\!]_{\rho_1(in(U))}, \ldots, [\![X_1^{c_n}]\!]_{\rho_1(in(U))}) & = \\ (in_{c_1}(U), \ldots, in_{c_n}(U)) & = \\ in(U) & \end{array}$$
  and the result follows.
* $A_1 = Y, Y \neq X_1$, then if $\rho(Y)$ is defined we have that $in(f(U)) = in(\rho(Y))$, and $g(in(U))$ is equal to:
  $$\begin{array}{ll} ([\![Y^{c_1}]\!]_{\rho_1(in(U))}, \ldots, [\![Y^{c_n}]\!]_{\rho_1(in(U))}) & = \\ (\mathcal{TR}^C(\rho)(Y^{c_1}), \ldots, \mathcal{TR}^C(\rho)(Y^{c_n})) & = \\ (in_{c_1}(\rho(Y)), \ldots, in_{c_n}(\rho(Y))) & = \\ in(\rho(Y)) & \end{array}$$
* $A_1 = X_j$, then $in(f(U)) = in([\![E']\!]_{\rho \sqcup [U/X_1]}(X_j))$, and $g(in(U))$ is equal to:
  $$\begin{array}{ll} ([\![X_j^{c_1}]\!]_{\rho_1(in(U))}, \ldots, [\![X_j^{c_n}]\!]_{\rho_1(in(U))}) & = \\ ([\![tr^C(E')]\!]_{\rho_2}(X_j^{c_1}), \ldots, [\![tr^C(E')]\!]_{\rho_2}(X_j^{c_n})) & \overset{(2.5)}{=} \\ (\text{where } \rho_2 = \mathcal{TR}^C(\rho) \sqcup [in_{c_1}(U_1)/X_1^{c_1}, \ldots, in_{c_n}(U_1)/X_1^{c_n}])) & \\ ([\![tr^C(E')]\!]_{\mathcal{TR}^C(\rho \sqcup [U_1/X_1])}(X_j^{c_1}), \ldots, [\![tr^C(E')]\!]_{\mathcal{TR}^C(\rho \sqcup [U_1/X_1])}(X_j^{c_n})) & = \\ (\mathcal{TR}^C([\![E']\!]_{\rho \sqcup [U_1/X_1]})(X_j^{c_1}), \ldots, \mathcal{TR}^C([\![E']\!]_{\rho \sqcup [U_1/X_1]})(X_j^{c_n})) & = \\ (in_{c_1}([\![E']\!]_{\rho \sqcup [U_1/X_1]}(X_j)), \ldots, in_{c_n}([\![E']\!]_{\rho \sqcup [U_1/X_1]}(X_j))) & = \\ in([\![E']\!]_{\rho \sqcup [U_1/X_1]}(X_j)) & \end{array}$$
* For $A_1 = \neg A_2, A_1 = A_2 \wedge A_3$ or $A_1 = \langle a \rangle A_2$ the proofs are similar to those of proposition 2.2.

By applying the *reduction* lemma 2.2, we get $in(\mu f) = \mu g$ and the result straightforwardly follows.

– $i = i' + 1$,
$\mathcal{TR}^C(\llbracket E \rrbracket_\rho)(X^c_{i'+1}) \quad =$
$\{s \mid c[s/z]' \in \llbracket E \rrbracket_\rho(X_{i'+1})\} \quad =$
$\{s \mid c[s/z]' \in \llbracket E_{i'+1,n} \rrbracket_{\rho'}(X_{i'+1})\} \quad =$
(where $\rho' = \rho \sqcup [\llbracket E \rrbracket_\rho(X_1)/X_1, \ldots, \llbracket E \rrbracket_\rho(X_{i'})/X_{i'}]$
$\mathcal{TR}^C(\llbracket E_{i'+1,n} \rrbracket_{\rho'})(X^c_{i'+1}) \quad = \quad (ind.hp.)$
$\llbracket tr^C(E_{i'+1,n}) \rrbracket_{\mathcal{TR}^C(\rho')}(X^c_{i'+1}) \quad =$
$\llbracket tr^C(E_{i'+1,n}) \rrbracket_{\mathcal{TR}^C(\rho) \sqcup \rho''}(X^c_{i'+1})$
where $\rho'' = [\mathcal{TR}^C(\llbracket E \rrbracket_\rho)(X^{c_1}_1)/X^{c_1}_1, \ldots, \mathcal{TR}^C(\llbracket E \rrbracket_\rho)(X^{c_n}_1)/X^{c_n}_1, \ldots,$
$\mathcal{TR}^C(\llbracket E \rrbracket_\rho)(X^{c_1}_{i'})/X^{c_1}_{i'}, \ldots, \mathcal{TR}^C(\llbracket E \rrbracket_\rho)(X^{c_n}_{i'})/X^{c_n}_{i'}]$.
By induction hypothesis on i we know that

$$\mathcal{TR}^C(\llbracket E \rrbracket_\rho)(X^c_j) = \llbracket tr^C(E) \rrbracket_{\mathcal{TR}^C(\rho)}(X^c_j)$$

for $1 \leq j \leq i'$ and $c \in C$. So we can rewrite ρ'' as:
$[\llbracket tr^C(E) \rrbracket_{\mathcal{TR}^C(\rho)}(X^{c_1}_1)/X^{c_1}_1, \ldots, \llbracket tr^C(E) \rrbracket_{\mathcal{TR}^C(\rho)}(X^{c_n}_1)/X^{c_n}_1, \ldots,$
$\llbracket tr^C(E) \rrbracket_{\mathcal{TR}^C(\rho)}(X^{c_1}_{i'})/X^{c_1}_{i'}, \ldots, \llbracket tr^C(E) \rrbracket_{\mathcal{TR}^C(\rho)}(X^{c_n}_{i'})/X^{c_n}_{i'}]$.
Hence it follows that :

$$\llbracket tr^C(E_{i'+1,n}) \rrbracket_{\mathcal{TR}^C(\rho) \sqcup \rho''}(X^c_{i'+1}) = \llbracket tr^C(E) \rrbracket_{\mathcal{TR}^C(\rho)}(X^c_{i'+1})$$

and indeed the thesis.

□

The following proposition shows the correctness of our partial evaluation function for list of equations with respect to contexts.

Proposition 2.3 *Let E be a closed list of equations, and $t[x]'$ a well behaved context, with $C = DerC(t[x]')$, then for every closed term s we have:*

$$t[s/x]' \in \llbracket E \rrbracket(X_i) \text{ iff } s \in \llbracket tr^C(E) \rrbracket(X_i^{t[x]'}).$$

Proof: We recall the semantical translation from environments:

$$\mathcal{TR}^C(\rho)(X^c) = \{s \mid c[s/z]' \in \rho(X)\},$$

it follows that $t[s/x]' \in \rho(X)$ iff $s \in \mathcal{TR}^C(\rho)(X^{t[x]'})$.

By using the above equivalence with $\rho = \llbracket E \rrbracket$, we can rewrite the thesis as:

$$s \in \llbracket tr^C(E) \rrbracket(X_i^{t[x]'}) \text{ iff } s \in \mathcal{TR}^C(\llbracket E \rrbracket)(X_i^{t[x]'}).$$

Since E is closed so $tr^C(E)$ will be closed. So let $\perp$ be the void environment, we have $\mathcal{TR}^C(\perp) = \perp$. By lemma 2.3 we get:

$$\mathcal{TR}^C(\llbracket E \rrbracket) = \mathcal{TR}^C(\llbracket E \rrbracket_\perp) =_{lemma\ 2.3} \llbracket tr^C(E) \rrbracket_{\mathcal{TR}^C(\perp)} = \llbracket tr^C(E) \rrbracket.$$

This concludes the proof. □

2.5 Examples

In this section we show some simple applications of the partial evaluation function.

2.5.1 Bergstra and Klop's priority operator

As first little example we consider the priority operator of Bergstra and Klop ([6]). It permits to give priorities among the actions that a process can perform. More formally, given a partial order $\leq$ among actions, then for every action a we have a rule:

$$\frac{x \xrightarrow{a} x' \ \ \{x \not\xrightarrow{b} \mid a \leq b, a \neq b\}}{\theta(x) \xrightarrow{a} \theta(x')}$$

We study the case with two actions l, h, with $\leq = \{(h, l)\}$ (l has a higher priority than h) and so we have the following two rules which, as we can easily see, respect our assumption 2.1 on the structure of the rules.

$$(\theta_1)\ \frac{x \xrightarrow{h} x' \quad x \not\xrightarrow{l}}{\theta(x) \xrightarrow{h} \theta(x')} \qquad (\theta_2)\ \frac{x \xrightarrow{l} x'}{\theta(x) \xrightarrow{l} \theta(x')}$$

Let us see the requirements that a term s must satisfy in order to have $\theta(x)[s/x]' \models \langle l\rangle\mathbf{T} \wedge \langle h\rangle\mathbf{T}$. First of all let us note that the only partially effective assignment for both the rules θ_1 and θ_2 and the context $\theta(x)$ is the void assignment. By applying our translation function in figure 2.5 we get:

$$\begin{array}{lcl} \sigma(\theta(x), \langle l\rangle\mathbf{T} \wedge \langle h\rangle\mathbf{T}) & = & \sigma(\theta(x), \langle l\rangle\mathbf{T}) \wedge \sigma(\theta(x), \langle h\rangle\mathbf{T}) \\ \sigma(\theta(x), \langle l\rangle\mathbf{T}) & = & \langle l\rangle\sigma(\theta(x), \mathbf{T}) \wedge \langle l\rangle\mathbf{T} \\ \sigma(\theta(x), \langle h\rangle\mathbf{T}) & = & \langle h\rangle\sigma(\theta(x), \mathbf{T}) \wedge \neg\langle l\rangle\mathbf{T} \wedge \langle h\rangle\mathbf{T} \\ \sigma(\theta(x), \mathbf{T}) & = & \mathbf{T} \end{array}$$

So $\sigma(\theta(x), \langle l\rangle\mathbf{T} \wedge \langle h\rangle\mathbf{T}) = \langle l\rangle\mathbf{T} \wedge \langle h\rangle\mathbf{T} \wedge \neg\langle l\rangle\mathbf{T} \wedge \langle h\rangle\mathbf{T}$ is equivalent to $\mathbf{F}$. In other words, there is no term s such that $\theta(x)[s/x]' \models \langle l\rangle\mathbf{T} \wedge \langle h\rangle\mathbf{T}$. This is effectively part of the intended semantics of the operator.

Let us show an example involving the use of the recursion in the logical language. The property we would like to analyze is the following, where s is a closed term:

$$\theta(x)[s/x]' \in [\![X =_\nu \langle h\rangle X]\!](X)$$

It expresses the fact that the process $\theta(x)[s/x]'$ can perform an infinite sequence of h actions. Let us apply the partial evaluation function for the logical language with recursion:

$$tr^C(X =_\nu \langle h\rangle X) = (X^{\theta(x)} =_\nu \sigma(\theta(x), \langle h\rangle X))$$

where $\sigma(\theta(x), \langle h\rangle X) = \neg\langle l\rangle\mathbf{T} \wedge \langle h\rangle X^{\theta(x)}$. So it must be $s \in [\![X^{\theta(x)} =_\nu \neg\langle l\rangle\mathbf{T} \wedge \langle h\rangle X^{\theta(x)}]\!](X^{\theta(x)})$, intuitively it means that the process s must be able to perform an infinite sequence of h actions, and during this sequence the process does not have the possibility of performing an l action. It is worthwhile noticing that both the approaches of Larsen and Xinxin and the one of Andersen cannot deal directly with this operator (see [39]).

2.5.2 *fictious clock* parallel operator

The next example shows an extension of the CCS parallel operator that cannot be handled by the approaches of Larsen and Anderson. This operator will be used in chapter 5 of this thesis and it is called *fictious clock* parallel operator. This name follows from the fact that it forces synchronization of both components in the t (or $tick$) actions, by modeling the elapsing of one unit of time. Assume $t, \tau \notin A$ and $Act = A \cup \{t, \tau\}$, then a natural way to give the operational semantics of the parallel operator w.r.t. the action t is the following:

$$\frac{x \xrightarrow{t} x' \quad y \xrightarrow{t} y' \quad \forall a \in A \quad \neg(x \xrightarrow{a} \wedge y \xrightarrow{\overline{a}})}{x \| y \xrightarrow{t} x' \| y'}$$

Unfortunately, the above rule does not respect the format of 2.3. We can give a set of rules in GSOS format that model the intended behaviour of the parallel operator w.r.t. the t action, by providing for every $A' = \{a_1, \dots a_n\} \subseteq A$ and $A' \setminus A = \{b_1, \dots, b_m\}$ a rule like:

$$\frac{\{x \xrightarrow{a_i} x_i\}^{1 \le i \le n} \quad x \xrightarrow{t} x' \quad y \xrightarrow{t} y' \quad \{x \not\xrightarrow{b_i}\}^{1 \le i \le m} \quad \{y \not\xrightarrow{\overline{a_i}}\}^{1 \le i \le n}}{x \| y \xrightarrow{t} x' \| y'}$$

Let us see the minimal requirements that a closed term q must satisfy in order to have $(p \| x)[q/x]' \models \langle t \rangle B$. The partial evaluation for a formula $\langle t \rangle B$ is the following:

$$(\langle t \rangle B)//p = \begin{cases} \langle t \rangle B//p' \wedge \bigwedge_{a \in A_1} [\overline{a}]\mathbf{F} & if p \xrightarrow{t} p', A_1 = \{a \in A \mid p \xrightarrow{a}\} \\ \mathbf{F} & otherwise \end{cases}$$

2.6 Overcoming the restrictions on the structure of the rules

In this section we show that our requirements on the structure of the rules of the GSOS system permit us to analyze complex terms. Even though, we require rules to keep the nesting of the parameters, this does not imply that we can analyze only terms where the variable is a top level operand. Let us see an example with the CCS parallel operator and the simple HML logic. Suppose to have the term $t = (p_1 \| (p_2 \| (\dots p_n \| x)))$, where $p_1, \dots, p_n$ are closed terms. We want the weakest condition on closed terms s s.t.:

$$(p_1 \| (p_2 \| (\dots p_n \| z)))[s/z] \models A$$

Our approach is not directly applicable to this term, but consider the term $t_2 = p_1 \| x_2$, hence we get:

$$(p_1 \| (p_2 \| (\dots p_n \| z)))[s/z] = t_2[((p_2 \| (\dots p_n \| x))[s/z])/x_2].$$

```
Evaluate($f(t_1,\ldots,t_n),E$) =
        **If**$f(t_1,\ldots,t_n)$ is a context
          **Then**
           **Let** $C = DerC(f(t_1,\ldots,t_n))$ **in**
             $tr^C(E)$
          **Else**
           {let $t_i$ be the sub term that strictly contains the variable }
           **Let** $C = DerC(f(t_1,\ldots,t_{i-1},x,t_{i+1},\ldots,t_n))$ **in**
             **Let** $E' = tr^C(E)$ **in**
               Evaluate($t_i, E'$)
```

Figure 2.6: Partial evaluation function for general terms with one variable.

Now we can apply the partial evaluation function to the context t_2 and let s_1 be $((p_2\|(\ldots p_n\|x))[s/z])$, so we get by lemma 2.2:

$$t_2[s_1/x_2] \models A \text{ iff } s_1 \models \sigma(t_2, A).$$

We have almost completed our task, since by applying $n-1$ times this reasoning we can get the weakest condition on s. It is possible to generalize this strategy to all contexts $t[x]'$ s.t. the variable x is not a top level parameter.

To treat with the full equational $\mu-$calculus we need some technical tools to handle variables. During the syntactic translation $tr^C()$, we supscript variables with contexts. In particular it is possible to supscribe more than once a variable X, for example $(X^{c_1[z_1]})^{c_2[z_2]}$. We implicitly rename this variable as $X^{c_1[(c_2[z_2])/z_1]}$, please note that $c_1[(c_2[z_2])/z_1]$ is still a term with one variable z_2.

In figure 2.6 we give the procedure that performs the partial evaluation for a generic term with one variable. Its correctness is stated by the following proposition.

Proposition 2.4 *Given a term t with one variable z, and a list of equations $E = (X_1 =_{\sigma_1} A_1, \ldots, X_n =_{\sigma_n} A_n)$, then we have for every closed term s:*

$$t[s/z] \in [\![E]\!](X) \ \text{ iff } \ s \in [\![Evaluate(t[z],E)]\!](X^{t[z]}).$$

Proof: By induction on number n of recursive calls of the function $Evaluate$.

- $n = 1$, then $Evaluate(t[z],E) = tr^C(E)$ and the thesis follows by proposition 2.3.
- $n = n' + 1$, then $t = f(t_1,\ldots,t_i,\ldots,t_n)$ with z appears in t_i. Let $t'[z']' = f(t_1,\ldots,z',\ldots,t_n)$ and $C = DerC(t'[z']')$. By definition of $Evaluate$ we get:

$$Evaluate(t[z],E) = Evaluate(t_i[z],E')$$

with $E' = tr^C(E)$. Hence, we get:

$$\begin{array}{lll}
t[s/z] \in [\![E]\!](X) & \text{iff} & \\
t'[(t_i[s/z])/z'] \in [\![E]\!](X) & \text{iff} & (prop.2.3) \\
t_i[s/z] \in [\![E']\!](X^{t'[z']}) & \text{iff} & (ind.hyp.) \\
s \in [\![Evaluate(t_i[z], E')]\!]((X^{t'[z']})^{t_i[z]}) & \text{iff} & \\
s \in [\![Evaluate(t_i[z], E')]\!](X^{t[z]}) & \text{iff} & \\
s \in [\![Evaluate(t[z], E)]\!](X^{t[z]}) & &
\end{array}$$

□

2.7 Related work

In this chapter we have rephrased the compositional analysis techniques, proposed by some authors, for a variant of GSOS format (that subsumes De Simone's format). In this way we try to follow a line of research that is devoted to produce general tools for analysis of systems, by starting from the semantic definition of their language.

In particular we refer to the work of Aceto *et al.* in [3], where the authors propose a procedure for converting any GSOS definition in a complete axiom system (possibly with one infinitary induction principle) which precisely characterizes (*strong bisimulation*) equivalence. Similarly, Simpson in [90] gives a generic proof system applicable to any language with an operational semantics defined in the GSOS format. The specification language is the so called HML logic, less expressive than $\mu-$calculus. So it seems that SOS systems, for specifying the operational behaviour of languages, can represent a foundational language from which several verification methods can be naturally developed.

In [5] Andersen proposed the compositional analysis (or partial evaluation) techniques for avoiding the so called *state-explosion* problem that arises in the analysis of concurrent programs. In fact the size of the LTS which describes the behaviour of a term is exponential in the number of parallel operators that are in the term. He considers the following verification problem: how to check efficiently that $p_1 \| p_2$ satisfies a formula ϕ. It is easy to prove that $p_1 \| p_2$ is strong bisimilar to $p_1 \| (p_2 \| \mathbf{0})$. So by theorem 2.1 the formula ϕ is satisfied by the former process if and only if is satisfied by the latter. Hence, we can apply many times the partial evaluation function for parallel operator. Moreover, at each time the reduced formula is substituted by a semantically equivalent formula, but with smaller size. Finally, we have the following verification problem: $\mathbf{0} \models \phi'$ and ϕ' is the resulting reduced and simplified formula. The key point is that is very simple to check if **0** satisfies a formula, it is sufficient to change every *possibility* modality in **F** and every *necessity* modality in **T** and then applying constant simplifications. Andersen named this technique *partial model checking*.

Chapter 3

Analysis of *Open systems*

In this chapter we face the analysis of *open* systems. In particular we study the verification problem called *module checking* by Kupferman and Vardi. We show how to reduce this problem for several temporal logics to a validity problem in deterministic (equational) $\mu-$calculus. In particular we use the compositional analysis techniques recalled in chapter 2. We show a simple example of application of the proposed theory.

3.1 Introduction

In the field of formal specification and verification of systems, temporal logics have been widely recognized as a valuable tool. In particular for the analysis of so called *reactive* and *non terminating* systems.

By following [42] we can distinguish between *closed* and *open* systems. The behaviour of a *closed* system is not influenced by its external environment. While the behaviour of an *open* system may depend on the interaction with the environment. For example, suppose to have a machine that serves drinks. A *closed* machine may work in the following way: it boils water, and *nondeterministically* serves *coffee* or *tea*. An *open* machine boils water, allows the environment to choose for a drink and then *deterministically* serves the chosen drink (see [44]).

A powerful automatic method for the verification of properties of (finite-state) systems is the so called *model checking* (see [19]). The idea is to consider a system as a Kripke model for a temporal logic. The Kripke semantics, also known as *possible worlds* semantics, propose a view of a system that we could refer to as closed. In fact a transition of system is always considered possible, regardless of the environment. Alternatively, we can imagine that this situation is equivalent to check the system in a setting where the environment does not limit its possibilities.

Sometimes, we are interested in establishing properties that an (open) system must satisfy when it operates in conjunction with arbitrary environments. This uncertainty about the behaviour of the environment makes *open* systems difficult to be programmed and analyzed.

In [50, 51, 101] Kupferman and Vardi have defined a simple and clear theoretical framework for the analysis of *open* systems. It is based on a refinement of the structure of Kripke model. The states (or worlds) are partitioned in two sets, the *system* states and the *environment* ones; the former may actually access every successors, the latter have a reduced set of possible worlds according to the environment. While this framework is surely theoretical interesting, it has also a practical interest.

3.2 *Module Checking*

In this section we briefly recall some preliminary definitions by following the treatment of [50]. A Kripke model is a tuple $P = \langle AP, W, R, w_{init}, l\rangle$ where AP is a finite set of atomic propositions, W is a set of worlds (or states), $R \subseteq W \times W$ is the accessibility relation (which is required to be total), w_{init} is the initial world, and l is a labeling function from W to 2^{AP}. Here, we suppose to have an initial state because we want to state properties that hold at the beginning of a computation.

A Kripke model is sometimes called program. A path on a program P is an infinite sequence of states $w_0, w_1, \ldots$ such that for every $0 \leq j$, we have $(w_j, w_{j+1}) \in R$. As notation we write $P \models \phi$ if a formula ϕ is satisfied in w_{init} (see chapter 1 for the formal definition of the relation $\models$ for several temporal logics).

A closed system is a program whose behaviour is completely determined by the state of the system, so every world related, *via* R, with the current state, can be effectively accessed.

To describe *open* systems Kupferman and Vardi define the structure of a module, i.e. a tuple $M = \langle AP, W_e, W_s, R, w_0, l\rangle$, by dividing the set W of states of a Kripke model in two sets W_s and W_e. For each state $w \in W_s \cup W_e$ let $succ(w)$ be the set of directly accessible states, i.e. $\{w'|(w, w') \in R\}$.

The different behaviour of the system in the two kinds of states, is reflected by considering $Step(s)$ as the set of sets of states that can be accessed from a state s. In fact for a state s in W_s we define $Step(w_s) = \{succ(w_s)\}$, while for a state s in W_e, due to the intended nature of W_e, we have to suppose that the environment can allow whatever subset of $succ(w_e)$. The only limitation is the (technical) necessity that the environment has to permit at least a state to be accessible from w_e. Hence let us define $Step(w_e)$ as $\{S|S \subseteq succ(w_e) \wedge S \neq \emptyset\}$.

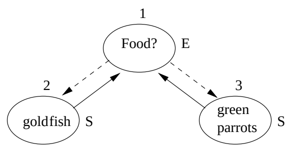


Figure 3.1: Module for food gathering machine.

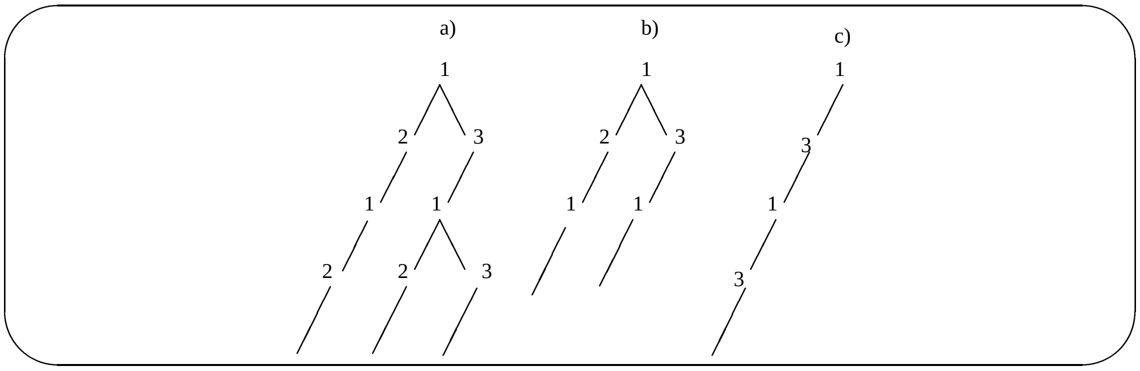


Figure 3.2: Some labeled trees in Exec(M).

As example we recall in figure 3.1 the module for the food gathering machine explained in the introduction. The state 1 is an environment state and we have $succ(1) = \{2, 3\}$ and $Step(1) = \{\{2\}, \{3\}, \{2, 3\}\}$, while $succ(2) = \{1\}$ and $Step(2)$ is $\{\{1\}\}$.

Following [50] we consider an infinite tree as a set $T \subseteq \mathbb{N}^*$ (i.e. a set of finite sequences of naturals) such that if $x.c \in T$ where $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$ then also $x \in T$, and for all $0 \leq c' < c$, we have that $x.c' \in T$. Moreover $x.0$ must be in T if $x \in T$. The empty word ϵ is the root of the tree and the other strings are the nodes.

A $\Sigma-$labeled tree is a pair $\langle T, V \rangle$, where T is a tree and V is a labelling function from each node of T to Σ (in our case Σ will be W or 2^{AP}).

We can unwind a module M in the tree $\langle T_M, V_M \rangle$ of its computations, by considering it as a Kripke model. Actually, the tree $\langle T_M, V_M \rangle$ corresponds to a situation where the module M is composed with the environment that allows the system to access every possible world, *via* R. Every environment induces a particular computation tree when it interacts with a module (open system). In order to consider all the possible computation trees of a *module* M that arise when the module is composed with arbitrary environments, one has to consider a forest of trees. Call this forest $Exec(M)$.

Informally every tree in the forest can be obtained from T_M by pruning some subtrees, whose root is a successor of an environment state w_e. Roughly speaking this means that the environment does not allow the system to access the state which is the root of the pruned subtree.

More formally let us define $Exec(M)$ in the following way. With the notation $\mathbb{N}^n$ we mean the sequences of length n of naturals. Let $T^{|n|}$ be $\bigcup_{0 \leq i \leq n} T \cap \mathbb{N}^n$ then we have $\langle T, V \rangle \in Exec(M)$ iff:

- $\epsilon \in T$ and $V(\epsilon) = w_0$,
- For all $x \in T$, if $V(x) = w$ then there exists $\{w_0, \ldots, w_n\} \in Step(w)$ such that $\{t \mid t \in T, \exists i \in \mathbb{N} \quad \text{s.t. } t = x.i\} = \{x.0, \ldots, x.n\}$ and for all $0 \leq c' \leq n$, we have that $V(x.c') = w_{c'}$.

Moreover it is assumed that trees $\langle T, V\rangle \in Exec(M)$ can be also $2^{AP}-$labeled trees, whose label function is $l(V(x))$ for $x \in T$. The appropriate labeling function will be clear from the context. To every labeled tree $\langle T, V\rangle \in Exec(M)$ we can associate a Kripke model $\langle T, V\rangle^K = \langle AP, T, \{(t, t.i) \mid t, t.i \in T, i \in \mathbb{N}\}, \epsilon, l \circ V\rangle$.

Remark 3.1 *It is worthwhile noticing that, given a labeled tree $\langle T, V\rangle \in Exec(M)$ and an order $\leq_\Sigma$ on Σ, there exists always a labeled tree $\langle T, V_1\rangle$ s.t. if $t.i, t.j \in T$ then $i \leq j$ iff $V_1(t.i) \leq_\Sigma V_1(t.j)$. Moreover there is an isomorphism f between $\langle T, V\rangle$ and $\langle T, V_1\rangle$ s.t. $f(\epsilon) = \epsilon$.*

Some trees of $Exec(M)$ where M is the food gathering machine are in figure 3.2. In the introduction we wonder if the food gathering machine can satisfy the CTL formula $(\forall \mathsf{G}\ \exists \mathsf{F}\ (goldfish))$ for every environment in which it operates. Clearly, by looking at the tree c in figure 3.2 it turns out that the answer is NO, since there is no state labeled by the proposition $goldfish$.

The authors of [50] define the problem of checking that every tree $\langle T, V\rangle$ in $Exec(M)$ satisfies ϕ (i.e. $\langle T, V\rangle^K \models \phi$), where ϕ can be a PTL, CTL, CTL^* formula, as module checking ($M \models_r \phi$). They prove that module checking problem is $EXPTIME$ and $2EXPTIME$ complete, respectively for CTL and CTL^*.

3.3 Technical framework

Here, we show how *module checking* problems can be reduced to a validity problem in the deterministic (equational) $\mu-$calculus. The idea is to "internalize" the environment. Then we may treat the environment as an unspecified component of a system and hence we apply partial evaluation techniques.

3.3.1 Partial evaluation technique

This technique relies upon compositional methods for proving properties of concurrent processes, specified in terms of a *process algebra* (see chapter 2). It has been presented in this formulation by Andersen in [5]. We follow his approach since Andersen has been among the firsts to propose these functions and moreover a prototype implementation has been produced ([58]) for performing partial evaluation. Together with our implementation of a *proof checker* for modal $\mu-$calculus we have a prototype implementation of the theory we are going to explain (see chapter 5).

First, we introduce explicitly an operator for parallel composition of LTS. It is classical CCS parallel operator, defined over LTS. Other parallel composition operators for processes could be taken in account. The parallel composition $E_1 \| E_2$ of two processes $E_1 = \langle P, A, \{\xrightarrow{a}\}_{a\in A}\rangle$ and $E_2 = \langle Q, A, \{\xrightarrow{a}\}_{a\in A}\rangle$ (i.e. their LTS) consists of the LTS whose set of states is given by $\{p\|q : p \in p, q \in Q\}$ and for every action $a \in A$ the relation $\xrightarrow{a}$ is given by the least relation induced by the following rules:

$$\frac{p \xrightarrow{a} p'}{p||q \xrightarrow{a} p'||q} \qquad \frac{q \xrightarrow{a} q'}{p||q \xrightarrow{a} p||q'} \qquad \frac{p \xrightarrow{a} p' \quad q \xrightarrow{\overline{a}} q'}{p||q \xrightarrow{\tau} p'||q'}$$

Given a process $E = \langle P, A, \{\xrightarrow{a}\}_{a \in A}\rangle$ the restricted process $E\backslash L$, where $L \subseteq A$ consists of a set of states $\{p\backslash L \mid p \in P\}$, and for every action $a \in A$ the relation $\xrightarrow{a}$ is given by the least relation induced by the following rule:

$$\frac{p \xrightarrow{a} p' \quad a \notin L}{p\backslash L \xrightarrow{a} p'\backslash L}$$

The logical language for the specification of the properties is the equational $\mu-$calculus (see section 1.7.2).

The intuitive idea is the following: proving that $E_1||E_2$ satisfies F is equivalent to prove that E_2 satisfies a modified specification $F//E_1$, where $//E_1$ is the partial evaluation function for the parallel composition operator (see table 3.1). Hence, the behaviour of a component has been partially evaluated and the requirements are changed in order to respect this evaluation. It is worthwhile noticing that, if we avoid the technical difficulties due to the presence of fixpoint operators, the partial evaluation function for the modal formulas can be seen as driven by the operational semantics rules (see chapter 2). This appears clear if one analyzes the partial evaluation rule for the formula $\langle\tau\rangle A$ w.r.t. the $||$ operator. By inspecting the inference rules, we can note that the process $p||q$ (with q unspecified component) can perform a τ action by exploiting one of the three possibilities:

- the process q performs an action τ going in a state q' and $p||q'$ satisfies A; this is taken into account by the formula $\langle\tau\rangle(A//p)$,
- the process p performs an action τ going in a state p' and $p'||q$ satisfies A, and this is considered by the disjunctions $\bigvee_{p \xrightarrow{\tau} p'} A//p'$, where every formula $A//p'$ takes into account the behavior of q in composition with a τ successor of p.
- the last possibility is that the τ action is due to the performing of two complementary actions by the two processes. So for every $a-$derivative p' of p there is a formula $\langle\overline{a}\rangle(A//p')$.

The following lemmas are given in [4, 5], where E_1 is a finite state process (we could also use the results of chapter 2).

Lemma 3.1 *Given a process $E_1||E_2$ and an equational specification $D\downarrow X$ we have:* $E_1||E_2 \models (D\downarrow X)$ iff $E_2 \models (D\downarrow X)//E_1$.

The *partial evaluation* function for the restriction operator can be found in [58].

Lemma 3.2 *Given a process $E\backslash L$ and an equational specification $D \downarrow X$ we have:* $(E)\backslash L \models (D\downarrow X)$ iff $E \models (D\downarrow X)//\backslash L$.

$$\begin{array}{rcl}
(D\!\downarrow X)//t & = & (D//t)\!\downarrow X_t \\
\epsilon//t & = & \epsilon \\
(X =_\sigma AD)//t & = & ((X_s =_\sigma A//s)_{s\in Der(t)})(D)//t \\
X//t & = & X_t \\
\langle a\rangle A//s & = & \langle a\rangle(A//s) \vee \bigvee_{s\xrightarrow{a}s'} A//s', \quad \text{if } a\neq\tau \\
\langle\tau\rangle A//s & = & \langle\tau\rangle(A//s) \vee \bigvee_{s\xrightarrow{\tau}s'} A//s' \vee \bigvee_{s\xrightarrow{a}s'} \langle\overline{a}\rangle(A//s') \\
[a]A//s & = & [a](A//s) \wedge \bigwedge_{s\xrightarrow{a}s'} A//s', \quad \text{if } a\neq\tau \\
[\tau]A//s & = & [\tau](A//s) \wedge \bigwedge_{s\xrightarrow{\tau}s'} A//s' \wedge \bigwedge_{s\xrightarrow{a}s'} [\overline{a}](A//s') \\
A_1\wedge A_2//s & = & (A_1//s)\wedge(A_2//s) \\
A_1\vee A_2//s & = & (A_1//s)\vee(A_2//s) \\
\mathbf{T}//s & = & \mathbf{T} \\
\mathbf{F}//s & = & \mathbf{F}
\end{array}$$

Table 3.1: Partial evaluation function for parallel operator $\|$.

3.4 Solution of *module checking* for $\mu-$calculus

In this section we show how the deterministic equational $\mu-$calculus can be used to solve the *module checking* problem for $\mu-$calculus.

3.4.1 The reduction to a validity problem in deterministic $\mu-$calculus

First of all, please note that there are several versions of temporal logics, that are interpreted over Kripke models and not directly on Labelled Transition Systems. Since our analysis framework is based on the notion of LTS we need some encodings[1]. The states of $LTSs$ do not carry information, i.e. states are not labeled by propositional symbols. On the other hand, Kripke models assume a unique accessibility relation among worlds. We can encode the accessibility relation of the Kripke model as a particular transition relation, say $\xrightarrow{\tau}$. Moreover we can encode the fact that a proposition holds in a state as the capability of performing a certain action when the system is in that state.

For example we may define two functions h_1 and h_2, respectively from Kripke models to LTSs and from CTL formulas to modal $\mu-$calculus formulas interpreted over LTS (without propositional symbols).

Let us define the function h_1 which, given a Kripke model $M = \langle AP, W, R, w_{init}, l\rangle$, returns the following LTS $h_1(M) = \langle S^W, A^{AP}\cup\{\tau\}, \{\xrightarrow{a}\}_{a\in A^{AP}\cup\{\tau\}}\rangle$, where

[1]In [25], the authors propose the notion of Doubly Labeled Transition Systems, which are $LTSs$ with information added in the states.

$$\begin{array}{rcl} S^W & = & \{s_w | w \in W\} \\ A^{AP} & = & \{a_p | p \in AP\} \\ \xrightarrow{\tau} & = & \{(s_w, s_{w'}) | (w, w') \in R\} \\ \xrightarrow{a_p} & = & \{(s_w, s_w) | p \in l(w), w \in W\} \end{array}$$

Formally, h_2 is defined inductively below:

$$\begin{array}{rcl} h_2(T) & = & \mathbf{T} \\ h_2(F) & = & \mathbf{F} \\ h_2(p) & = & \langle a_p \rangle \mathbf{T} \\ h_2(\neg\phi) & = & \neg h_2(\phi) \\ h_2(\phi \vee \phi') & = & h_2(\phi) \vee h_2(\phi') \\ h_2(\phi \wedge \phi') & = & h_2(\phi) \wedge h_2(\phi') \\ h_2(EX\phi) & = & \langle \tau \rangle h_2(\phi) \\ h_2(AX\phi) & = & [\tau] h_2(\phi) \\ h_2(\exists \phi \mathcal{U} \phi') & = & \mu X. h_2(\phi') \vee (h_2(\phi) \wedge \langle \tau \rangle X) \\ h_2(\forall \phi \mathcal{U} \phi') & = & \mu X. h_2(\phi') \vee (h_2(\phi) \wedge [\tau] X) \end{array}$$

Now we can state the following lemma:

Lemma 3.3 $w, M \models \phi$ iff $s_w, h_1(M) \models h_2(\phi)$.

Similar encodings and corresponding lemmas can be stated for the propositional $\mu-$calculus interpreted over Kripke models, CTL^* and $ECTL^*$.

Remark 3.2 *In the sequel we are interested to study module checking problems for equational $\mu-$calculus defined over Kripke models. Hereafter we assume as given the semantics*[2] *of this logic and moreover that a function h_2' is given s.t. for every equational specification ϕ and every Kripke model M we have $w, M \models \phi$ iff $s_w, h_1(M) \models h_2'(\phi)$.*

We now describe a translation from *modules* to LTSs.

Definition 3.1 *Given a* module $M = \langle AP, W_e, W_s, R, w_0, l \rangle$ *we define the following LTS* $P^M = \langle S, A, \{\xrightarrow{a}\}_{a \in A} \rangle$ *where:*

$$\begin{array}{rcl} S & = & \{s_w | w \in W\} \\ A & = & A^{AP} \cup A_s \cup A_e \\ & & \quad A^{AP} = \{a_p | p \in AP\} \\ & & \quad A_s = \{a^s_{w,w'} | (w, w') \in R \wedge w \in W_s\} \\ & & \quad A_e = \{a^e_{w,w'} | (w, w') \in R \wedge w \in W_e\} \\ \forall p \in AP : \xrightarrow{a_p} & = & \{(s_w, s_w) \mid p \in l(w)\} \\ \forall w \in W_s : \xrightarrow{a^s_{w,w'}} & = & \{(s_w, s_{w'}) \mid (w, w') \in R\} \\ \forall w \in W_e : \xrightarrow{a^e_{w,w'}} & = & \{(s_w, s_{w'}) \mid (w, w') \in R\}. \end{array}$$

[2]A similar definition can be straightforwardly obtained from the definition of equational $\mu-$calculus in section 1.7.2, by considering a unique label (τ) and the standard clause for propositional constants.

Characterization of *Exec(M)* as *LTSs*

Our intention is to describe the set of trees in $Exec(M)$ up to isomorphism as LTSs. To achieve this aim we characterize a set of processes, that we call $Envs(M)$.

We will show that given a module M, the set $Exec(M)$ can be characterized by the set $Comp(M) = \{\hat{u}((P^M \| X)\backslash L) | X \in Envs(M)\}$, where $\hat{u}$ is an unwinding function (see below) and $L = A_s \cup A_e \cup \overline{A_s} \cup \overline{A_e}$. In particular, each tree in $Exec(M)$ is isomorphic to $\hat{u}((P^M \| X)\backslash L)$ for a process X in $Envs(M)$ and *vice versa*. We consider rooted LTS, i.e. LTS with a distinguished initial state s_{init}.

Let us see the constraints that processes in $Envs(M)$ must satisfy in order to obtain the above characterization. Let o be in $\{s, e\}$ and $X = \langle S, \overline{A_s} \cup \overline{A_e}, \{\xrightarrow{a}\}_{a \in \overline{A_s} \cup \overline{A_e}}\rangle$ be a process in $Envs(M)$ then:

(1) The set of actions that can be performed by processes in $Envs(M)$ is limited to the complementary actions of the actions in $A_s \cup A_e$, namely $\overline{A_s \cup A_e} = \{\overline{a} \mid a \in A_s \cup A_e\}$. There are neither actions tied to propositional symbols nor τ actions.

(2) $\forall s \in S$, if $s \xrightarrow{\overline{a^o_{w,w'}}} s' \wedge w' \in W_s \Longrightarrow \forall a \in \{a^s_{w',w''} | w'' \in succ(w')\} \quad s' \xrightarrow{\overline{a}}$.
It states that after a complementary action that leads the other component to a *system* state, the process X must permit, by offering a complementary action, every action of the other component.

(2') $\forall s \in S$, if $s \xrightarrow{\overline{a^o_{w,w'}}} s' \wedge w' \in W_e \Longrightarrow \exists A^1 \subseteq \{a^e_{w',w''} | w'' \in succ(w')\}, A^1 \neq \emptyset : \forall a \in A^1 \quad s' \xrightarrow{\overline{a}}$.
It states that after a complementary action that leads the other component to an *environment* state, the process X must permit, by offering complementary actions, a non empty subset of the actions that the other component could perform.

(3) If the initial state of P^M is $s_{w_{init}}$ with $w_{init} \in W_s$ then $\forall a \in \{a^s_{w',w''} \mid w' = w_{init}, w'' \in succ(w')\} \quad s_{init} \xrightarrow{\overline{a}}$.
It means that the initial state of a process X in $Envs(M)$, where w_{init} is a system state, must offer every complementary action of the actions performed by the translated module in its initial state $s_{w_{init}}$.

(3)' If the initial state of P^M is $s_{w_{init}}$ with $w_{init} \in W_e$ then $\exists A^1 \subseteq \{a^e_{w',w''} | w' = w_{init}, w'' \in succ(w')\}, A^1 \neq \emptyset : \forall a \in A^1 \quad s_{init} \xrightarrow{\overline{a}}$.
It means that the initial state of a process X in $Envs(M)$, where w_{init} is a environment state, must offer a non empty subset of the complementary actions of the actions performed by the translated module in its initial state $s_{w_{init}}$.

(4) The processes in $Envs(M)$ must be deterministic.

The requirements (1-3') can be simply translated in a list of equations in the equational $\mu-$calculus as follows:

$$
\begin{aligned}
Y_1 &=_\nu (\bigwedge_{a\in\overline{A_e}\cup\overline{A_s}}[a]Y_1)\wedge(\bigwedge_{a\in A_e\cup A_s\cup\{\tau\}\cup A^{AP}}[a]\mathbf{F}) && (3.1)\\
Y_2 &=_\nu (\bigwedge_{a\in\overline{A_e}\cup\overline{A_s}}[a]Y_2)\wedge(\bigwedge_{a\in\{a^o_{w,w'}|w'\in W_s\}}[\overline{a}](\bigwedge_{a\in\{a^s_{w',w''}|w''\in succ(w')\}}\langle\overline{a}\rangle\mathbf{T})) && (3.2)\\
Y_{2'} &=_\nu (\bigwedge_{a\in\overline{A_e}\cup\overline{A_s}}[a]Y_{2'})\wedge(\bigwedge_{a\in\{a^o_{w,w'}|w'\in W_e\}}[\overline{a}](\bigvee_{a\in\{a^e_{w',w''}|w''\in succ(w')\}}\langle\overline{a}\rangle\mathbf{T})) && (3.3)\\
Y_3 &=_\nu \bigwedge_{a\in\{a^s_{w',w''}|w'=w_{init},w''\in succ(w')\}}\langle\overline{a}\rangle\mathbf{T} && (3.4)\\
Y_{3'} &=_\nu \bigvee_{a\in\{a^e_{w',w''}|w'=w_{init},w''\in succ(w')\}}\langle\overline{a}\rangle\mathbf{T} && (3.5)
\end{aligned}
$$

Given a module $M=\langle AP,W_e,W_s,R,w_{init},l\rangle$, we associate to M a list of equations C^M defined in the following way:

if $w_{init}\in W_s$ then C^M consists of the equations 3.1, 3.2, 3.3 and 3.4,

if $w_{init}\in W_e$ then C^M consists of the equations 3.1, 3.2, 3.3 and 3.5.

For the requirement 4 we use the theory presented in chapter 1 for the interpretation of (equational) $\mu-$calculus formulas over deterministic $LTSs$.

We formally define the unwinding function $\hat{u}$. This simply unwinds the LTS along the τ actions. Consider the LTS $L=\langle S,AP\cup\{\tau\},\{\xrightarrow{a}\}_{a\in AP\cup\{\tau\}}\rangle$ s.t. if $s\in S,a\in AP$ and $s\xrightarrow{a}s'$ then $s'=s$. Let s be a state of an LTS L then the unwinding of L by starting from s is a tree like LTS $\hat{u}(s)=\langle S',AP\cup\{\tau\},\{\xrightarrow{a}'\}_{a\in AP\cup\tau}\rangle$ rooted in s^ϵ where let the relation u^r be $\cup_{0\leq n}u^{\epsilon,n}(s)$, and with $\gamma\subseteq S^*$ and $n\in\mathbb{N}$ we have:

$$
\begin{aligned}
u^{\gamma,0}(s) &= \emptyset\\
u^{\gamma,n+1}(s) &= \{(s^\gamma,\tau,s'^{\gamma s})|s\xrightarrow{\tau}s'\}\cup\bigcup_{s':s\xrightarrow{\tau}s'}u^{\gamma s,n}(s')
\end{aligned}
$$

and $\xrightarrow{\tau}'=\{(s^\gamma,s'^{\gamma s})\mid(s^\gamma,\tau,s'^{\gamma s})\in u^r\}$ and $\xrightarrow{a}'=\{(s^\gamma,s^\gamma)\mid s\xrightarrow{a}s\}$ for $a\in AP$ and $S'=\{s^\gamma\mid s^\epsilon\longrightarrow'^* s^\gamma\}$ with $\longrightarrow'=\bigcup_{a\in AP\cup\{\tau\}}\xrightarrow{a}'$. Please note that $u^{\epsilon,n}(s)\subseteq u^{\epsilon,n+1}(s)$. Sometimes we refer to the unwinding of a process by meaning the unwinding of the LTS associated with the process by starting from its initial state.

Moreover we can give the following lemma, that states that a formula is satisfied in a state s of an LTS iff it is satisfied in its unwinding (see [27]).

Lemma 3.4 *Given a (equational) $\mu-$calculus formula ψ and an LTS $L=\langle S,A,\{\xrightarrow{a}\}_{a\in A}\rangle$ then we have:*

$$L,s\models\psi \text{ iff } \hat{u}(s),s^\epsilon\models\psi.$$

Suppose to have a module $M = \langle AP, W_e, W_s, R, w_{init}, l\rangle$ and an order $\leq_W$ on $W = W_e \cup W_s$. Hence in the rest of this section we refer to every set of worlds $\{w_0, \ldots, w_n\}$ in such a way that $w_i \leq_W w_j$ iff $i \leq j$.

We define a function from processes in $Comp(M)$ to labeled trees. Actually, the codomain of this function is the labelling of a tree, from which a tree can be simply derived[3].

Given a state u^γ in the set of states of $\hat{u}((P^M \| X)\backslash L)$ let $succ_\tau(u^\gamma) = \{u'^{\gamma u} : u^\gamma \xrightarrow{\tau} u'^{\gamma u}\}$. We assume a derived order among the elements of $succ_\tau(u^\gamma)$ in the following way $((s_{w_i} \| X_i)\backslash L)^{\gamma u'} \leq_D ((s_{w_j} \| X_j)\backslash L)^{\gamma u'}$ iff $w_i \leq_W w_j$. As for set of worlds, in the sequel we refer to set of derivatives $\{((s_{w_0} \| X_0)\backslash L)^{\gamma u'}, \ldots, ((s_{w_n} \| X_n)\backslash L)^{\gamma u'}\}$ in such a way that $((s_{w_i} \| X_i)\backslash L)^{\gamma u'} \leq_D ((s_{w_j} \| X_j)\backslash L)^{\gamma u'}$ iff $w_i \leq_W w_j$.

Hence, given a process $\hat{u}((P^M \| X)\backslash L)$ in $Comp(M)$ whose set of states is S' then let $\overline{\phi}(\hat{u}((P^M \| X)\backslash L))$ be:

$$\bigcup_{0 \leq n} \; \bigcup_{s \in \{s^\gamma \in S' : |\gamma| = n\}} \phi^n(s)$$

where:

$$\begin{array}{rcl}
\phi^0(((s_{w_{init}} \| X)\backslash L)^\epsilon) & = & \langle \epsilon, w_{init}\rangle \\
\phi^{n+1}(((s_w \| X)\backslash L)^{\gamma u'}) & = & \text{let } \langle t', w'\rangle = \phi^n(u'^\gamma) \text{ in} \\
& & \text{let } succ_\tau(u'^\gamma) = \{((s_{w_0} \| X_0)\backslash L)^{\gamma u'}, \ldots, ((s_{w_n} \| X_n)\backslash L)^{\gamma u'}\} \\
& & \text{in case } ((s_w \| X)\backslash L)^{\gamma u'} = ((s_{w_c} \| X_c)\backslash L)^{\gamma u'} \text{ then} \\
& & \langle t'.c, w_c\rangle
\end{array}$$

Remark 3.3 *Suppose to have $((s_w \| X)\backslash L)^\gamma$ in $Der(\hat{u}((P^M \| X)\backslash L))$ with $\mid \gamma \mid = n$, then by observing the function $\overline{\phi}$ we note that $\phi^n(((s_w \| X)\backslash L)^\gamma) = \langle t, w\rangle$ for some $t \in \mathbb{N}^n$.*

The following lemma states that $\overline{\phi}$ returns a W–labelling of a tree.

Lemma 3.5 *Given a process $\hat{u}((P^M \| X)\backslash L)$ in $Comp(M)$ then $\overline{\phi}(\hat{u}((P^M \| X)\backslash L))$ is a W–labelling of a tree.*

Proof:

Assume to have $\hat{u}((P^M \| X)\backslash L)$ in $Comp(M)$ whose set of states is S'. We show that the functions ϕ^n for $n \in \mathbb{N}$ are injective, that is to say for $s^\gamma, s'^{\gamma'} \in S'$ if $\phi^n(s^\gamma) = \phi^n(s'^{\gamma'})$ then $s^\gamma = s'^{\gamma'}$. At the same time we prove that $\overline{\phi}(\hat{u}((P^M \| X)\backslash L))$ is a function from $\mathbb{N}^*$ to W.

The proof proceeds by induction on n. For $n = 0$ it is obvious. For $n > 0$ suppose to have $((s_w \| X)\backslash L)^\gamma$ and $((s_{w'} \| X')\backslash L)^{\gamma'}$ such that $\phi^n(((s_w \| X)\backslash L)^\gamma) = \langle t, w\rangle$ and $\phi^n(((s_{w'} \| X')\backslash L)^{\gamma'}) = \langle t, w\rangle$. Hence it follows from remark 3.3 that $w' = w$; moreover we have $\gamma = \gamma_1 u_1$ and $\gamma' = \gamma'_1 u'_1$. So let $\langle t_1, w_1\rangle$ be $\phi^{n-1}(u_1^{\gamma_1})$ and $\langle t'_1, w'_1\rangle$ be $\phi^{n-1}(u_1'^{\gamma'_1})$, and it must be $t_1 = t'_1$. By inductive hypothesis on n we know that also $w_1 = w'_1$. Now

[3]Simply, one takes as tree the set of sequences of naturals where the labeling is defined.

we get by inductive hypothesis on injectiveness of ϕ^{n-1} that $u_1^{\gamma_1} = u_1'^{\gamma_1'}$. So it follows the thesis since the processes in $Envs(M)$ are deterministic and hence $X = X'$.

Now we prove that $\overline{\phi}(\hat{u}((P^M \| X)\backslash L)) = V$ is actually a labelling of a tree. We have that $\langle \epsilon, w_{init} \rangle \in V$. Moreover suppose to have $\langle t, w \rangle \in V$ with $\mid t \mid = n$, hence there exists a state $u^\gamma \in S'$ s.t. $\phi^n(u^\gamma) = \langle t, w \rangle$. By definition of ϕ^n it follows that there are k (with $k > 0$) τ successors of u^γ and so k couples $\langle t.0, w_0 \rangle, \ldots, \langle t.k, w_k \rangle$ in V.

$\square$

In the sequel it is technically preferable to consider $\overline{\phi}(\hat{u}((P^M \| X)\backslash L))$ as a labeled tree, since the following results can be more easily stated.

Assume to have a process $\hat{u}((P^M \| X)\backslash L)$ in $Comp(M)$ and let $\overline{\phi}(\hat{u}((P^M \| X)\backslash L))$ be $\langle T, V \rangle$. Given $u^\gamma, u'^{\gamma'} \in Der(\hat{u}((P^M \| X)\backslash L))$ with $\mid \gamma \mid = n$ and $\mid \gamma' \mid = n + 1$ then it is easily checked that:

$$u^\gamma \xrightarrow{\tau} u'^{\gamma'} \text{ iff } \begin{array}{l} \phi^n(u^\gamma) = \langle t, w \rangle \text{ and} \\ \phi^{n+1}(u'^{\gamma'}) = \langle t.i, w' \rangle \text{ and} \\ (w, w') \in R \end{array}$$

Now consider $\langle T, V \rangle$ as a 2^{AP} labeled tree. We have for all $p \in AP$ and for all $u^\gamma \in Der(\hat{u}((P^M \| X)\backslash L))$ that if $\overline{\phi}(u^\gamma) = \langle t, w \rangle$ then:

$$u^\gamma \xrightarrow{a_p} u^\gamma \text{ iff } p \in l(w).$$

This property is guaranteed since for every s_w in the set of states of P^M we have $s_w \xrightarrow{a_p} s_w$ iff $p \in l(w)$. Furthermore, we prevented processes in $Envs(M)$ from performing actions in A^{AP}, and the restriction operator $\backslash L$ does not avoid these actions.

Let $X \in Envs(M)$ then we show that there exists a labeled tree $\langle T, V \rangle$ in $Exec(M)$ s.t. $\overline{\phi}(\hat{u}((P^M \| X)\backslash L)) = \langle T, V \rangle$.

Lemma 3.6 *For every* $X \in Envs(M)$ *we have* $\overline{\phi}(\hat{u}((P^M \| X)\backslash L)) \in Exec(M)$.

Proof:

We show that $\overline{\phi}(\hat{u}((P^M \| X)\backslash L)) = \langle T, V \rangle$ is in $Exec(M)$. We follow the definition of $Exec(M)$ and we see that $\epsilon \in T$ and $V(\epsilon) = w_{init}$.

Suppose to have $t' \in T \cap \mathbb{N}^n$ with $V(t') = w'$. Hence by definition of $\overline{\phi}$ there exists $u'^{\gamma'} = ((s_{w'} \| X')\backslash L)^{\gamma'} \in Der(\hat{u}((P^M \| X)\backslash L))$ with $\mid \gamma' \mid = n$ s.t. $\phi^n(u'^{\gamma'}) = \langle t', w' \rangle$.

We may have the following cases:

- $w' \in W_s$, then if $\gamma' = \epsilon$ then u' is the initial state of $\hat{u}((P^M \| X)\backslash L)$ and the state in P^M is $s_{w_{init}}$. So condition (3) on processes in $Envs(M)$ ensures that every possible transition from $s_{w_{init}}$ will be allowed, i.e. X' must perform every complementary action of the actions in $a \in \{a^s_{w',w''} \mid w' = w_{init}, w'' \in succ(w')\}$.

  Let $succ(w_{init})$ be $\{w_0, \ldots, w_n\}$, by construction of P^M we get $s_{w_{init}} \xrightarrow{a^s_{w_{init},w_i}} s_{w_i}$ for $i \in \{1, \ldots, n\}$. Hence let $\{((s_{w_0} \| X_0')\backslash L)^{\gamma' u'}, \ldots, ((s_{w_n} \| X_n')\backslash L)^{\gamma' u'}\}$ be the set of τ successors of $((s_{w'} \| X')\backslash L)^{\gamma'}$.

Then by definition of the function ϕ^{n+1} we have $\phi^{n+1}(((s_{w_i} \| X_i')\backslash L)^{\gamma' u'}) = \langle t'.i, w_i \rangle$ for every $((s_{w_i} \| X_i')\backslash L)^{\gamma' u'}$, hence it follows that $\{t \mid t \in T, \exists i \in \mathbb{N} \text{ s.t. } t = t'.i\} = \{t'.0, \ldots, t'.n\}$, and for every c' with $0 \leq c' \leq n$ we have $V(t'.c') = w_{c'}$.

Analogous is the case that $\gamma' \neq \epsilon$, simply by using condition (2).

- $w' \in W_e$, then if $\gamma' = \epsilon$ then u' is initial is the initial state of $\hat{u}((P^M \| X)\backslash L)$ and the state in P^M is $s_{w_{init}}$. So condition (3') may be applied, by ensuring that a non empty subset of the transitions of $s_{w_{init}}$ will be allowed. The proof follows as before. If $\gamma' \neq \epsilon$ the condition (2') help us.

□

Conversely, now we show that for every labeled tree $\langle T, V_1 \rangle \in Exec(M)$ a process X in $Envs(M)$ exists s.t. $\overline{\phi}(\hat{u}((P^M \| X)\backslash L))$ is isomorphic to $\langle T, V_1 \rangle$. First of all note that, by remark 3.1, we can find an a labeled tree $\langle T, V \rangle$ isomorphic to $\langle T, V_1 \rangle$ s.t. if $\{t'.i \mid t'.i \in T, t = t', i \in \mathbb{N}\} = \{t.0, \ldots, t.n\}$ then we have $V(t.0) \leq_W V(t.1) \leq_W \ldots \leq_W V(t.n)$. Call *well ordered* this kind of trees in $Exec(M)$.

Lemma 3.7 *Given a well ordered tree $\langle T, V \rangle \in Exec(M)$ we find a process X in $Envs(M)$ s.t. $\overline{\phi}(\hat{u}((P^M \| X)\backslash L)) = \langle T, V \rangle$.*

Proof:

We build inductively such process X (whose associated LTS will be tree like). Moreover, by induction on n' we show that $\langle T, V \rangle$ and $\overline{\phi}(\hat{u}((P^M \| X)\backslash L))$ are equal up to depth of n'.

For the root ϵ of the tree with $V(\epsilon) = w_{init}$ we have $u = (P^M \| X)\backslash L$, where $Der(X) = \{X\}$, and the initial state of P^M is $s_{w_{init}}$.

Let us suppose to have built such a process X with a depth of at most n'.

Now consider $t \in T$ and $|t| = n'$ with $V(t) = w$. So, by inductive hypothesis, there exists a state u^γ with $\mid \gamma \mid = n'$ in the states of $\hat{u}((P^M \| X)\backslash L))$ s.t. $\phi^{n'}(u^\gamma) = \langle t, w \rangle$. Hence, u^γ will be of the form $((s_w \| X')\backslash L)^\gamma$, where $X' \in Der(X)$ and $Der(X') = \{X'\}$. Let us build its set of successors s.t. if $\{t'.i \mid t'.i \in T, t = t', i \in \mathbb{N}\} = \{t.0, \ldots, t.n\}$ and for every c' with $0 \leq c' \leq n$ we have $V(t'.c') = w_{c'}$ then $succ_\tau(u^\gamma) = \{((s_{w_0} \| X_0')\backslash L)^{\gamma u}, \ldots, ((s_{w_n} \| X_n')\backslash L)^{\gamma u}\}$.

From the last observation we should have that for every τ successor $((s_{w_i} \| X_i')\backslash L)^{\gamma u}$ of u^γ we get $\phi^{n'+1}(((s_{w_i} \| X_i')\backslash L)^{\gamma u} = \langle t.i, w_i \rangle$ from which follows that $\langle T, V \rangle$ and $\hat{u}((P^M \| X)\backslash L))$ are equal up to strings of length $n' + 1$.

We may have the following cases:

- if $w \in W_s$ then let $X' = \sum_{a \in A_s} \overline{a}.X_a$, this permits X' to enjoy condition (2). Moreover with this extension we have that $((s_w \| X')\backslash L)^\gamma$ can go, by performing a τ action, in one of the following states $\{((s_{w_0} \| X_{a^s_{w,w_0}})\backslash L)^{\gamma u}, \ldots, ((s_{w_n} \| X_{a^s_{w,w_n}})\backslash L)^{\gamma u}\}$, where $\{w_0, \ldots, w_n\}$ is $succ(w)$.

  In fact by construction of P^M we have $s_w \overset{a^s_{w,w_i}}{\longrightarrow} s_{w_i}$ for every $i \in \{1, \ldots, n\}$.

  If $\gamma = \epsilon$ then w is w_{init} and so condition 3 is ensured.

- if $w \in W_e$ then let $\{w_0, \ldots, w_n\} \subseteq W$ such that $\{t.0, \ldots, t.n\} = \{t'.i \mid t'.i \in T, t = t', i \in \mathbb{N}\}$, and $V(t.i) = w_i$ for $i \in \{0, \ldots, n\}$. So we have $(w, w_i) \in R$ for $i \in \{0, \ldots, n\}$; let X' be $\sum_{a \in A'} \overline{a}.X_a$, where $A' = \{a^e_{w,w'} \mid w' \in \{w_0, \ldots, w_n\}\}$. Clearly, X' satisfies condition $2'$.

  So, we have $((s_w \| X') \backslash L)^\gamma$ can go by performing a τ action in one of the following states $\{((s_{w_0} \| X_{a^e_{w,w_0}}) \backslash L)^{\gamma u}, \ldots, ((s_{w_n} \| X_{a^e_{w,w_n}}) \backslash L)^{\gamma u}\}$.

  In fact by construction of P^M we have $s_w \xrightarrow{a^e_{w,w_i}} s_{w_i}$ for every $i \in \{1, \ldots, n\}$.

  If $\gamma = \epsilon$ then w is w_{init} and so condition $3'$ is ensured.

During the above construction, we defined a process X that performs only actions in $\overline{A_s \cup A_e}$, and so condition 1 is ensured. Moreover X is a deterministic process, since for every action a in $\overline{A_s \cup A_e}$ we have defined only an a–successor for every derivatives of X. So X satisfies the conditions of membership in $Envs(M)$. □

Hence we get:

Proposition 3.1 *Given an equational $\mu-$calculus formula ψ and a module M we have:*

$$\forall \langle T, V \rangle \in Exec(M) : h_1(\langle T, V \rangle^K) \models \psi$$
$$\text{iff}$$
$$\forall X \in Envs(M) : \hat{u}((P^M \| X) \backslash L) \models \psi.$$

Proof: ($\Longrightarrow$)

By contradiction. Suppose to have $X \in Envs(M)$ s.t. $\hat{u}((P^M \| X) \backslash L) \not\models \psi$. Then by lemma 3.6 there exists $\langle T, V \rangle \in Exec(M)$ s.t. $\overline{\phi}(\hat{u}((P^M \| X) \backslash L)) = \langle T, V \rangle$. It turns out that $h_1(\langle T, V \rangle^K)$ is equal (up to renaming of states) to $\hat{u}((P^M \| X) \backslash L)$. Hence we get a contradiction since by hypothesis $h_1(\langle T, V \rangle^K) \models \psi$.

The other direction can be proved by using a symmetric argument. □

Suppose to have a module $M = \langle AP, W_e, W_s, R, w_{init}, l \rangle$ and an equational $\mu-$calculus formula $E \downarrow_Z$ that can be interpreted over Kripke models. Hence we can define the following list of equations:

$$F = (Z' =_\nu A'), (((h'_2(E \downarrow_Z)) // \backslash L) // P^M), C^M \tag{3.6}$$

where Z' is a new variable not in $Def(E) \cup Def(C^M)$ and A' is equal to

$$\bigwedge_{Y \in Def(C^M)} Y \Longrightarrow Z_{P^M}.$$

It is worthwhile noticing that Z_{P^M} is in $Def((((h'_2(E \downarrow_Z)) // \backslash L) // P^M))$. Hence we can state the following proposition:

Proposition 3.2 *Given an equational $\mu-$calculus formula $E\downarrow_Z$, and a module M, we have:*

$$\begin{array}{c} M \models_r E\downarrow_Z \\ \text{iff} \\ F\downarrow_{Z'} \text{ is valid in deterministic equational } \mu\text{calculus.} \end{array}$$

Proof:

$$\begin{array}{lll}
M \models_r E\downarrow_Z & \text{iff} & \\
\forall\langle T,V\rangle \in Exec(M) : \langle T,V\rangle^K \models E\downarrow_Z & \text{iff} & \text{(see remark 3.2)} \\
\forall\langle T,V\rangle \in Exec(M) : h_1(\langle T,V\rangle^K) \models h_2'(E\downarrow_Z) & \text{iff} & (prop.\ 3.1) \\
\forall X \in Envs(M) : \hat{u}((P^M \| X)\backslash L) \models h_2'(E\downarrow_Z) & \text{iff} & (lem.\ 3.4) \\
\forall X \in Envs(M) : (P^M \| X)\backslash L \models h_2'(E\downarrow_Z) & \text{iff} & (lem.\ 3.1, 3.2) \\
\forall X \in Envs(M) : X \models (((h_2'(E\downarrow_Z))//\backslash L)//P^M) & \text{iff} & \text{(see list 3.6)} \\
\forall X \text{deterministic} : X \models F\downarrow_{Z'} & \text{iff} & \\
F\downarrow_{Z'} \text{ is valid } (deterministic\ equational\ \mu - calculus) & &
\end{array}$$

□

3.5 Complexity analysis

In this section we study the complexity of our approach for the solution of the *module checking* problem. Let us give below the notion of *simple* assertion.

Definition 3.2 *An assertion is* simple *iff it is of the form:*

$$A := \mathbf{T} \mid \mathbf{F} \mid X_1 \wedge X_2 \mid X_1 \vee X_2 \mid \langle a\rangle X \mid [a]X$$

A list of equations E is simple *iff every assertion in the list is* simple.

The partial evaluation function for || has a good property, in fact if the structure of the assertion is simple then the *size* of reduced assertion is polynomial in the size of the initial assertion. The necessity to restrict ourselves to this kind of list of equations relies on the following observation (see [4]):

Suppose to have an equation list $E = (X =_\mu \langle a\rangle[a]\ldots[a]\langle a\rangle X)$ (with l modalities). P is a process whose associated LTS $\langle S, \{a\}, \{\xrightarrow{a}\}\rangle$ has n states and every state can reach the others (and itself) by mean of an action a. By applying the partial evaluation function for parallel operator with P as known component and by supposing that the unknown component cannot perform τ actions, we have:

$$((\langle\tau\rangle[\tau]\ldots[\tau]\langle\tau\rangle X)//P) \doteq \bigvee_{s\in S}(\langle\overline{a}\rangle \bigwedge_{s\in S}[\overline{a}]\ldots\bigvee_{s\in S}\langle\overline{a}\rangle X_s).$$

Hence the *size* of the resulting assertion is $\mathcal{O}(n^l)$.

Under the assumption that the assertion is *simple* Andersen has proved:

Lemma 3.8 *If A is a simple assertion then for every state s of a transition system $T = \langle S, A, \{\xrightarrow{a}\}_{a \in A}\rangle$ we have $\mid A// \setminus L//s \mid = \mathcal{O}(\mid A \mid\mid S \mid)$.*

Since the translation $tr^C()$ for every assertion in D produces $\mid S \mid$ assertions, we have that *size* of the resulting list of assertions after the translation and the partial evaluation is $\mathcal{O}(\mid T \mid\mid D \mid)$.

To treat with modal $\mu-$calculus, we observe that Andersen has also given a linear translation from $\mu-$calculus to equational one s.t. the resulting list of equations is *simple*. Now we have the technical tools to state the main result of this chapter.

Theorem 3.1 *The* module checking *for the $\mu-$calculus is decidable.*

Proof: We can simply observe that given a $\mu-$calculus formula ψ and a module M, we can translate ψ in a simple list of equations E_ψ whose *size* is polynomial in the length of ψ, and we can associate to M the LTS P^M, whose size is linear in the size of M. The partial evaluation of the equation list E_ψ produces a list whose size is polynomial in the length of the original formula and in the size of the LTS P^M. Moreover the size of the list of equations C^M which is used to express the conditions on $Envs$ processes, is linear in the size of M. Hence the problem is reduced to a validity problem in deterministic equational $\mu-$calculus.

A simple decision procedure for equational deterministic $\mu-$calculus is translating the list of equations in a $\mu-$calculus formula, by using the translation tr of section 1.7.3. Then by applying the theorem 1.1 for the satisfiability (validity) problem of deterministic $\mu-$calculus, we obtain a complexity that is double exponential in the product of length of the original formula and the the size of the module. □

3.6 Solving the *module checking* for other temporal logics

Bhat and Cleaveland proposed interesting translations from CTL,CTL^* and $ECTL^*$ into equational $\mu-$calculus (see [10]). They exploited these translations for efficient temporal logic model checking for branching time logics. In particular in this way the space-efficient model checking procedures developed for $\mu-$calculus (see [5, 16]), can be used also for CTL^* and $ECTL^*$.

Proposition 3.3 *The following results are proved in [10]:*

- *for every formula $\phi \in CTL$ there is a* simple *list of equations E s.t. $E \downarrow X_1$ is equivalent to ϕ. The list of equations has* size *linear in the length of ϕ and can be calculated in linear time.*

- *for every formula $\phi \in CTL^*$ there is a* simple *list of equations E s.t. $E \downarrow X_1$ is equivalent to ϕ. The list of equations has* size *exponential in the length of ϕ and can be calculated in exponential time.*

- *for every formula $\phi \in ECTL^*$ there is a* simple *list of equations E s.t. $E \downarrow X_1$ is equivalent to ϕ. The list of equations has* size *linear in the length of ϕ and can be calculated in linear time.*

By means of our result on the decidability *module checking* problem for equational $\mu-$calculus we can state:

Proposition 3.4 *Given a module M then:*

- *There is a procedure that solves the* module checking *problem in (deterministic double) exponential time on the length of a CTL formula ϕ and in the* size *of M.*
- *There is a procedure that solves the* module checking *problem in (deterministic triple) exponential time on the length of a CTL* formula ϕ and in the* size *of M.*
- *There is a procedure that solves the* module checking *problem in (deterministic double) exponential time on the length of a ECTL* formula ϕ and in the* size *of M.*

Conjecture 3.1 *Kupferman and Vardi have shown that module checking problem for CTL and CTL^* is respectively $EXPTIME$ and $2EXPTIME$ complete. Hence, there is an exponential blow up from the complexity of their procedure and ours. We conjecture that is possible to find a suitable decision procedure for deterministic equational $\mu-$calculus (or simultaneous one) such that our procedure has a complexity that is exponential in the size of the formula and in the size of the module for $\mu-$calculus. Hence we could obtain optimal results for CTL and CTL^*. We are encouraged in this supposition since for simultaneous $\mu-$calculus, Street and Emerson claim the desired complexity in [94]. We leave the proof of this conjecture as a future work.*

3.7 An example

In this section we show a simple example of application of our theory. Since we have reduced the *module checking* problem to a validity problem in deterministic (equational) $\mu-$calculus, we may use the proof system proposed in chapter 1 to perform the analysis.

We have used the partial evaluation theory for the CCS operator, as proposed by Andersen, for several reasons, in particular since a prototype implementation (called *mudiv*) has been provided by Nielsen (see [58]). By starting from this tool we have developed a software environment that offers some features which permit to solve the module checking problem for simple properties without the necessity to implement the satisfiability procedure for deterministic $\mu-$calculus (see chapter 5 for a better explanation of our tool).

It is worthwhile noticing that our approach is not only algorithmic such as the one of Kupferman and Vardi, but we can prove that a system satisfies a property simply by exploiting our axiomatization for deterministic $\mu-$calculus.

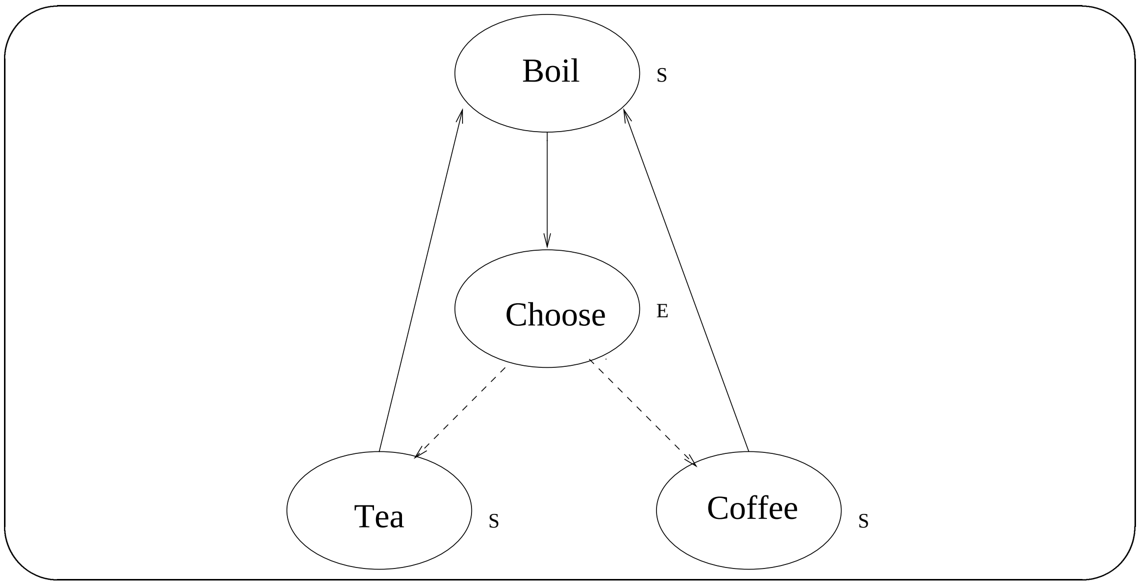


Figure 3.3: The module for a vending machine.

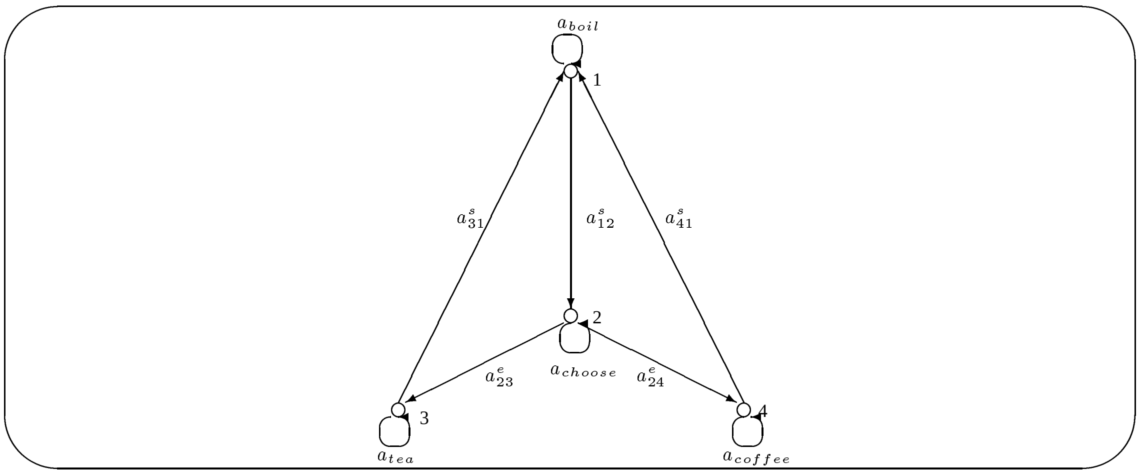


Figure 3.4: The translated module of the vending machine.

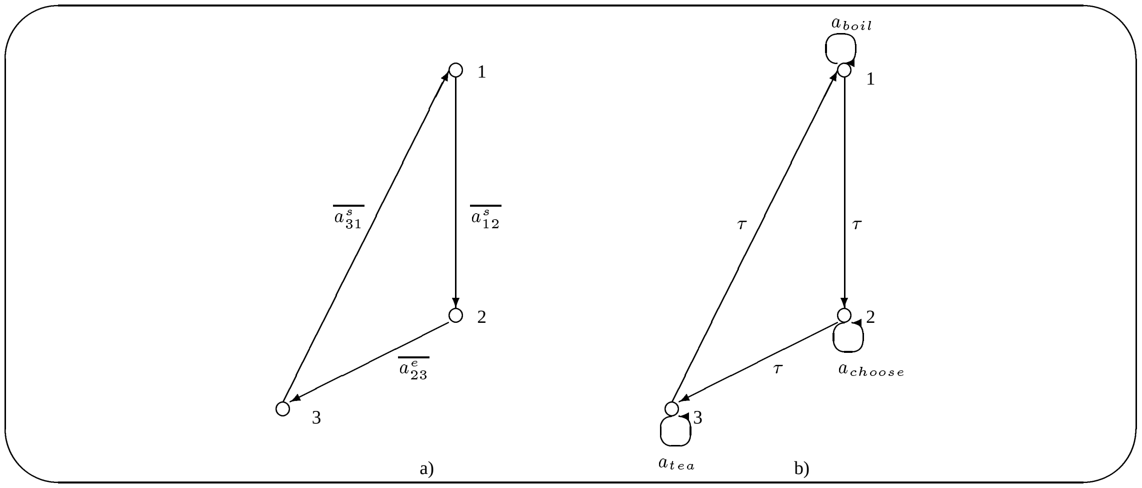


Figure 3.5: An environment process X and the composed system $(P^M \| X) \setminus L$.

Consider the module in figure 3.3. The system boils water and then it reaches a state where it can accept an interaction with the environment, i.e. serving tea of coffee. The choice is performed by the environment. After serving one of the two drinks, it returns in its initial position. A minimal requirement is that the system can always reach a state where it can serve a drink (tea or coffee), for every environment in which it operates. We recall that every environment must allow at least one of the possible transitions that the system has at a certain state. A CTL formula that expresses this property is the following

$$\neg(\exists \mathbf{T} \, \mathsf{U} \, (\neg \exists \mathbf{T} \, \mathsf{U} \, (tea \vee coffee))).$$

By using the translation h_2 we can get the equivalent $\mu-$calculus formula:

$$\phi' = \neg(\mu X. \neg(h_2(\exists \mathbf{T} \, \mathsf{U} \, (tea \vee coffee))) \vee (\mathbf{T} \wedge \langle \tau \rangle X)$$

where

$$h_2(\exists \mathbf{T} \, \mathsf{U} \, (tea \vee coffee)) = \mu Y.(\langle a_{tea} \rangle \mathbf{T} \vee \langle a_{coffee} \rangle \mathbf{T}) \vee \langle \tau \rangle Y.$$

Since we want a positive formula we can consider the equivalent positive formula:

$$\phi'' = \nu X.(\mu Y.(\langle a_{tea} \rangle \mathbf{T} \vee \langle a_{coffee} \rangle \mathbf{T}) \vee \langle \tau \rangle Y) \wedge [\tau] X.$$

This formula can be translated in an equivalent equational definition that is also *simple*:

$$\begin{array}{lll} X & =_\nu & Y \wedge X_2 \\ X_2 & =_\nu & [\tau] X \\ Y & =_\mu & Y_1 \vee Y_2 \\ Y_1 & =_\mu & \langle \tau \rangle Y \\ Y_2 & =_\mu & Y_3 \vee Y_4 \\ Y_3 & =_\mu & \langle a_{tea} \rangle \mathbf{T} \\ Y_4 & =_\mu & \langle a_{coffee} \rangle \mathbf{T} \end{array}$$

The module M for the vending machine is the following tuple $M = \langle AP, W_e, W_s, R, w_{init}, l\rangle$ where:

$$\begin{array}{lcl} AP & = & \{boil, choose, coffee, tea\} \\ W_s & = & \{1,3,4\} \\ W_e & = & \{2\} \\ R & = & \{(1,2),(2,3),(2,4),(3,1),(4,1)\} \\ l & = & \{(1,\{boil\}),(2,\{choose\}),(3,\{tea\}),(4,\{coffee\})\} \\ w_{init} & = & 1 \end{array}$$

The associated LTS P^M is shown in figure 3.4. The set of actions L is $(A_e \cup A_s \cup \overline{A_e} \cup \overline{A_s})$. We have used the program *mudiv* and the tool described in section 5.4 for partially evaluating P^M w.r.t. the previous equation list. After performing some simple simplifications on the resulting list of assertions, that do not modify the semantics of the list, this is translated in a $\mu-$calculus formula.

The output of the program is given in figure 3.6, where `Vi` is the ν operator, `Box` and `Dia` are necessity and possibility modalities, and `~as12` is $\overline{a^s_{1,2}}$ (similarly for the other actions).

```
Vi
   ("Xs1",
    And
      (Dia (["~as12"], Or (Dia (["~ae24"], True), Dia (["~ae23"], True))),
       Box
         (["~as12"],
          And
            (Or (Dia (["~ae24"], True), Dia (["~ae23"], True)),
             And
               (Box (["~ae24"], Box (["~as41"], Var "Xs1")),
                Box (["~ae23"], Box (["~as31"], Var "Xs1")))))),
```

Figure 3.6: Output of the tool.

The formula can be rewritten as follows:

$$\phi''' = \nu Xs1.\phi_1 \wedge \phi_2$$

where:

$$\begin{array}{lcl} \phi_1 & = & \langle\overline{a^s_{1,2}}\rangle\phi_3 \\ \phi_2 & = & [\overline{a^s_{1,2}}](\phi_3 \wedge \phi_4) \\ \phi_3 & = & \langle\overline{a^e_{2,4}}\rangle\mathbf{T} \vee \langle\overline{a^e_{2,3}}\rangle\mathbf{T} \\ \phi_4 & = & [\overline{a^e_{2,4}}][\overline{a^s_{4,1}}]Xs1 \wedge [\overline{a^e_{2,3}}][\overline{a^s_{3,1}}]Xs1 \end{array}$$

3.7.1 A proof

Given a module M let us restate the property expressed by the list of equation C^M in deterministic $\mu-$calculus. Thus we can use the proof system presented in chapter 1.

Hence, given a module M redefine C^M as $C_1 \wedge C_2 \wedge C_{2'}$ and whenever $w_{init} \in W_s$ let I^M be C_3, otherwise $C_{3'}$.

(1) $\nu X.(\bigwedge_{a\in\overline{A_e}\cup\overline{A_s}}[a]X) \wedge (\bigwedge_{a\in A_e\cup A_s\cup\{\tau\}\cup A^{AP})}[a]\mathbf{F})$

(2) $\nu X.(\bigwedge_{a\in\overline{A_e}\cup\overline{A_s}}[a]X) \wedge (\bigwedge_{a\in\{a^o_{w,w'}|w'\in W_s\}}[\overline{a}](\bigwedge_{a\in\{a^s_{w',w''}|w''\in succ(w')\}}\langle\overline{a}\rangle\mathbf{T}))$

(2') $\nu X.(\bigwedge_{a\in\overline{A_e}\cup\overline{A_s}}[a]X) \wedge (\bigwedge_{a\in\{a^o_{w,w'}|w'\in W_e\}}[\overline{a}](\bigvee_{a\in\{a^e_{w',w''}|w''\in succ(w')\}}\langle\overline{a}\rangle\mathbf{T}))$

(3) $\bigwedge_{a\in\{a^s_{w',w''}|w'=w_{init},w''\in succ(w')\}}\langle\overline{a}\rangle\mathbf{T}$

(3') $\bigvee_{a\in\{a^e_{w',w''}|w'=w_{init},w''\in succ(w')\}}\langle\overline{a}\rangle\mathbf{T}$

In this subsection we show that $C^M \wedge I^M \implies \phi'''$ is valid in *deterministic* $\mu-$calculus, by showing that $C^M \wedge I^M \vdash \phi'''$ is provable.

First of all let us see some useful lemmas that will be used during the proof, and assert some capabilities of the processes in $Envs$.

For every $w' \in W_e$ the following sequents are provable:

$$C^M \vdash [a^o_{w,w'}](\bigvee_{a\in\{a^e_{w',w''}|w''\in succ(w')\}}\langle\overline{a}\rangle T) \wedge [a^o_{w,w'}]C^M \tag{3.7}$$

For every $w' \in W_s$ the following sequents are provable:

$$C^M \vdash [a^o_{w,w'}](\bigwedge_{a\in\{a^s_{w',w''}|w''\in succ(w')\}}\langle\overline{a}\rangle T) \wedge [a^o_{w,w'}]C^M \tag{3.8}$$

Since the initial state 1 is in W_s then we have $I^M = \langle a^s_{1,2}\rangle\mathbf{T}$.

The proof for the sequent $C^M \wedge I^M \vdash \phi'''$ is the following. We have used the proof system presented in the chapter 1 with some derived rules such as:

$$\frac{\psi \vdash \alpha(\psi)}{\psi \vdash \nu X.\alpha(X)}(\nu)$$

Let us show the proof:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle\overline{a^e_{2,3}}\rangle \vee \langle\overline{a^e_{2,4}}\rangle, C^M, \mathbf{T} \vdash \phi_3}{\langle\overline{a^e_{2,3}}\rangle \vee \langle\overline{a^e_{2,4}}\rangle, C^M, \mathbf{T}, \neg\phi_3 \vdash}\,\mathcal{L}\neg}{[\overline{a^s_{1,2}}](\langle\overline{a^e_{2,3}}\rangle \vee \langle\overline{a^e_{2,4}}\rangle), [\overline{a^s_{1,2}}]C^M, \langle\overline{a^s_{1,2}}\rangle\mathbf{T}, [\overline{a^s_{1,2}}]\neg\phi_3 \vdash}\,\langle\overline{a^s_{1,2}}\rangle}{[\overline{a^s_{1,2}}](\langle\overline{a^e_{2,3}}\rangle \vee \langle\overline{a^e_{2,4}}\rangle) \wedge [\overline{a^s_{1,2}}]C^M, \langle\overline{a^s_{1,2}}\rangle\mathbf{T}, [\overline{a^s_{1,2}}]\neg\phi_3 \vdash}\,\mathcal{L}\wedge}{C^M, I^M, [\overline{a^s_{1,2}}]\neg\phi_3 \vdash}\,\mathit{cut}\text{ with }3.7}{C^M, I^M \vdash \neg[\overline{a^s_{1,2}}]\neg\phi_3}\,\mathcal{R}\neg}{C^M, I^M \vdash \phi_1}\,\mathit{cut}\text{ with axiom} \qquad \dfrac{\pi_2}{C^M, I^M \vdash \phi_2(C^M \wedge I^M)}}{C^M, I^M \vdash \phi_1 \wedge \phi_2(C^M \wedge I^M)}\,\mathcal{R}\wedge}{C^M \wedge I^M \vdash \phi_1 \wedge \phi_2(C^M \wedge I^M)}\,\mathcal{L}\wedge}{C^M \wedge I^M \vdash \phi'''}\,\nu
$$

π_2 :

$$
\dfrac{\dfrac{\dfrac{\dfrac{\pi_3}{[\overline{a^s_{1,2}}]\phi_3 \wedge [\overline{a^s_{1,2}}]C^M, I^M, \langle\overline{a^s_{1,2}}\rangle\neg(\phi_3 \wedge \phi_4(C^M \wedge I^M)) \vdash}}{C^M, I^M, \langle\overline{a^s_{1,2}}\rangle\neg(\phi_3 \wedge \phi_4(C^M \wedge I^M)) \vdash}\,\mathit{cut}\text{ with }3.7}{C^M, I^M \vdash \neg\langle\overline{a^s_{1,2}}\rangle\neg(\phi_3 \wedge \phi_4(C^M \wedge I^M))}\,\mathcal{R}\neg}{C^M, I^M \vdash \phi_2(C^M \wedge I^M)}\,\mathit{cut}\text{ with axiom}
$$

π_3 :

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\phi_3, C^M \vdash \phi_3} \qquad \dfrac{\dfrac{\pi_4}{\phi_3, C^M \vdash [\overline{a^e_{2,4}}][\overline{a^s_{4,1}}](C^M \wedge I^M)} \quad \dfrac{\pi_5}{\phi_3, C^M \vdash [\overline{a^e_{2,3}}][\overline{a^s_{3,1}}](C^M \wedge I^M)}}{\phi_3, C^M \vdash [\overline{a^e_{2,4}}][\overline{a^s_{4,1}}](C^M \wedge I^M) \wedge [\overline{a^e_{2,3}}][\overline{a^s_{3,1}}](C^M \wedge I^M)}\,\mathcal{R}\wedge}{\phi_3, C^M \vdash (\phi_3 \wedge \phi_4(C^M \wedge I^M))}\,\mathcal{R}\wedge}{\phi_3, C^M, \neg(\phi_3 \wedge \phi_4(C^M \wedge I^M)) \vdash}\,\mathcal{L}\neg}{[\overline{a^s_{1,2}}]\phi_3, [\overline{a^s_{1,2}}]C^M, I^M, \langle\overline{a^s_{1,2}}\rangle\neg(\phi_3 \wedge \phi_4(C^M \wedge I^M)) \vdash}\,\langle\overline{a^s_{1,2}}\rangle}{[\overline{a^s_{1,2}}]\phi_3 \wedge [\overline{a^s_{1,2}}]C^M, I^M, \langle\overline{a^s_{1,2}}\rangle\neg(\phi_3 \wedge \phi_4(C^M \wedge I^M)) \vdash}\,\mathcal{L}\wedge
$$

π_4 :

$$\begin{array}{c l}
\langle\overline{a^s_{1,2}}\rangle\mathbf{T}, C^M \vdash C^M, I^M & \\
\hline\hline
\langle\overline{a^s_{1,2}}\rangle\mathbf{T}, C^M, \neg(C^M \wedge I^M) \vdash & \mathcal{L}\neg, \mathcal{R}\wedge \\
\hline
\langle\overline{a^s_{4,1}}\rangle\mathbf{T}, [\overline{a^s_{4,1}}]\langle\overline{a^s_{1,2}}\rangle\mathbf{T}, [\overline{a^s_{4,1}}]C^M, \langle\overline{a^s_{4,1}}\rangle\neg(C^M \wedge I^M) \vdash & \langle\overline{a^s_{4,1}}\rangle \\
\hline
\langle\overline{a^s_{4,1}}\rangle\mathbf{T}, [\overline{a^s_{4,1}}]\langle\overline{a^s_{1,2}}\rangle\mathbf{T}, [\overline{a^s_{4,1}}]C^M \vdash \neg\langle\overline{a^s_{4,1}}\rangle\neg(C^M \wedge I^M) & \mathcal{R}\neg \\
\hline
\langle\overline{a^s_{4,1}}\rangle\mathbf{T}, [\overline{a^s_{4,1}}]\langle\overline{a^s_{1,2}}\rangle\mathbf{T}, [\overline{a^s_{4,1}}]C^M \vdash [\overline{a^s_{4,1}}](C^M \wedge I^M) & cut \text{ with } axiom \\
\hline
\langle\overline{a^s_{4,1}}\rangle\mathbf{T}, [\overline{a^s_{4,1}}]\langle\overline{a^s_{1,2}}\rangle\mathbf{T} \wedge [\overline{a^s_{4,1}}]C^M \vdash [\overline{a^s_{4,1}}](C^M \wedge I^M) & \mathcal{L}\wedge \\
\hline
\langle\overline{a^s_{4,1}}\rangle\mathbf{T}, C^M \vdash [\overline{a^s_{4,1}}](C^M \wedge I^M) & cut \text{ with } 3.8 \\
\hline
\langle\overline{a^s_{4,1}}\rangle\mathbf{T}, C^M, \neg[\overline{a^s_{4,1}}](C^M \wedge I^M) \vdash & \mathcal{L}\neg \\
\hline
\phi_3, [\overline{a^e_{2,4}}]\langle\overline{a^s_{4,1}}\rangle\mathbf{T}, [\overline{a^e_{2,4}}]C^M, \langle\overline{a^e_{2,4}}\rangle\neg[\overline{a^s_{4,1}}](C^M \wedge I^M) \vdash & \langle\overline{a^e_{2,4}}\rangle \\
\hline
\phi_3, [\overline{a^e_{2,4}}]\langle\overline{a^s_{4,1}}\rangle\mathbf{T} \wedge [\overline{a^e_{2,4}}]C^M, \langle\overline{a^e_{2,4}}\rangle\neg[\overline{a^s_{4,1}}](C^M \wedge I^M) \vdash & \mathcal{L}\wedge \\
\hline
\phi_3, C^M, \langle\overline{a^e_{2,4}}\rangle\neg[\overline{a^s_{4,1}}](C^M \wedge I^M) \vdash & cut \text{ with } 3.8 \\
\hline
\phi_3, C^M \vdash \neg\langle\overline{a^e_{2,4}}\rangle\neg[\overline{a^s_{4,1}}](C^M \wedge I^M) & \mathcal{R}\neg \\
\hline
\phi_3, C^M \vdash [\overline{a^e_{2,4}}][\overline{a^s_{4,1}}](C^M \wedge I^M) & cut \text{ with axiom}
\end{array}$$

π_5 is similar to the proof π_4.

By observing the module for the vending machine it should be clear that if an environment never chooses the $coffee$ option then in this case the system does not have the possibility to serve this drink, in other words the result of module checking of the CTL formula:

$$\psi = \neg(\exists\mathbf{T} \cup (\neg\exists\mathbf{T} \cup coffee))$$

is false. This can be formally proved by providing a process in $Envs(M)$ s.t. $(P^M \| X) \setminus L \not\models \psi'$, where ψ' is the equational $\mu-$calculus translation of ψ. For example the behaviour of the LTS in figure 3.5(a) in composition with P^M is represented in figure 3.5(b). The *mudiv* program can be used to perform the model checking of the resulting system w.r.t. the equation list ψ'.

3.8 Conclusions and future work

In this chapter we have studied an alternative approach for the analysis of module checking problems (see [50, 51, 52, 101]).

Indeed, we propose the use of the equational $\mu-$calculus (actually a deterministic variant) and the compositional analysis techniques for the solution of *module checking* problems. In particular we give a procedure for solving the *module checking* problem for modal $\mu-$calculus, and $ECTL^*$, by extending the results of [50].

In this way we define a unified framework for the analysis of *module checking* problems for several temporal logics.

This work is in the line of research proposed by Bhat, Cleaveland and Emerson (and many others) that propose the $\mu-$calculus as an *intermediate* language into which others logics may be translated for efficient *model checking*. Here we prove that $\mu-$calculus can also be used as an *intermediate* language for *module checking* too.

We propose a solution method that may be applied by using proof theoretic techniques and syntactic translations among formulas. Furthermore we have built a simple software environment which may be used to perform module checking for small systems (see section 3.7).

As a future work we plan to study module checking problems for restricted classes of formulas, in order to obtain more efficient decision procedures. In [50] Kupferman and Vardi studied the module checking problem for $\forall CTL$, i.e. the subset of CTL formulas such that there is only universal quantification over paths, and moreover every quantification is in the scope of an even number of negations. The model checking and the module checking problem coincide for this set of formulas. Hence the module checking problem can be solved in polynomial time in the size of the module and in the size of the formula.

We conjecture that the *disjunctive* $\mu-$calculus formulas, defined by Janin and Walukiewicz in [46], may be exploited in this analysis. In fact the satisfiability problem for these formulas is decidable in linear time (see [46]). We believe that a similar result may be obtained for *disjunctive* deterministic $\mu-$calculus formulas. Furthermore, there is a proper subset of this class, whose corresponding set of negated formulas is closed w.r.t. the partial evaluation function. Hence the idea is to reformulate our approach in such a way to reduce the module checking problem for the negated of this sub class of *disjunctive* $\mu-$calculus formulas to a satisfiability problem for *disjunctive* deterministic formulas.

Chapter 4

A synthesis problem

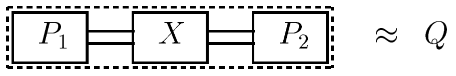
In this chapter we study a synthesis problem that can be defined through underspecification. In the previous chapters we have seen how to verify that, for each component that may be substituted for an unspecified one, the resulting system satisfies a property (in particular for terms of a process algebra). Here we study a method for finding a suitable system for inserting in the unspecified component, if it exists, such that the whole system respects a specification. Actually in this case the specification is represented by another system and not by a logical formula.

The techniques used are different from the compositional ones, even though these could be used too, since in this way it is possible to define more efficient synthesis procedure.

4.1 Introduction

In the field of automatic synthesis of programs one often has an abstract specification of the whole program and an incomplete implementation, and he would like to automatically derive a complete implementation that satisfies the specification.

The problem for concurrent systems has been dealt with firstly by Merlin and Bochmann in [68], where specifications and implementations are expressed in terms of execution sequences and trace equivalence is used as a criterion of satisfiability. An implementation of the method is in [88]. In [86] Shields models the same problem in terms of CCS process algebra (see [69]) and starts from the problem of two given modules p_1, p_2 that interact through an unspecified interface X and must satisfy an abstract specification q:



This can be written as $\exists X : (p \| X) \setminus L \approx q$, where $p = p_1 \| p_2$. The equation is called the *interface equation* by Shields. In the equation p, X, q are CCS terms, $\|$ is the parallel composition, $\setminus L$ is the restriction on synchronization actions L and $\approx$ is Milner's *observational equivalence*. This equivalence abstracts from internal communications (i.e.

two processes are equivalent iff they are not distinguished by an external observer), and is more discriminating with respect to comparing execution sequences, in fact it is sensitive to potential deadlocks. Shields gives necessary and sufficient conditions for the solution of the problem when p, q are finite state processes and q is deterministic and rigid (i.e. without internal actions). The work by Qin and Lewis [81] extends the work of Shields by giving an algorithm to find the most general solution (a solution that simulates every other solution) if it exists.

More recently Haghverdi and Ural in [41] have proposed an algorithm based on sets of derivatives, claiming the naturalness of their approach, but their algorithm still has a complexity that, in the worst case, is exponential in the product of the state space of p and q. In [77] Parrow presents an alternative and elegant method for solving the problem. This is based on successive transformations of equations into simpler ones, together with the generation of a solution. The method is semi-automatic and tries to find the most general solution, but the strategy is not complete. A complexity measure for the algorithm is not given, but the procedure uses backtracking. An extension for the solution of general context equations is developed in an original way by Larsen and Liu in [54]. In this paper q can be a generic CCS process, but the equivalence considered is *strong bisimulation* [70], which does not abstract from internal behaviour of a system, and so it is not well suited for a top down design methodology. All the algorithms proposed in [41, 54, 81, 86] have complexity in time and space exponential in the product of the state space of p and q. In this chapter we modify the algorithm proposed in [81] and obtain a complexity in time and space of $O(2^{|p|\times\log|q|})$. Algorithms proposed in [41, 86] can be modified similarly.

4.2 Theoretical framework for the solution of the interface equation

We follow the treatment given by Shields in [86]. Let Act be a set of actions and $\Lambda(p) \subseteq Act$ be the set of actions of p. We use a parallel operator $\|_L$ which corresponds to the composition of the CCS operators $||$ and $\backslash L$. The process, parallel composition of two processes, can perform an action not in L if one of the components can perform it, but to do a synchronization (τ action) both components must be able to perform complementary actions in L. The set L is called synchronization actions set. So the LTS associated with $p\|_L q$ has states $p'\|_L q'$ with $p' \in Der(p), q' \in Der(q)$, actions in $(\Lambda(p) \cup \Lambda(q))\backslash L \cup \{\tau\}$, initial state $p\|_L q$, and, for every $a \in \Lambda(p\|_L q)$, $\xrightarrow{a}$ is the least relation that can be inferred by the following rules:

$$\frac{p\xrightarrow{a}p',\quad a\notin L}{p\|_L q\xrightarrow{a}p'\|_L q} \qquad \frac{q\xrightarrow{a}q',\quad a\notin L}{p\|_L q\xrightarrow{a}p\|_L q'}$$

$$\frac{p\xrightarrow{l}p' \quad q\xrightarrow{\bar{l}}q' \quad (l,\bar{l}\in L)}{p\|_L q\xrightarrow{\tau}p'\|_L q'}$$

We recall the definition of simulation among processes.

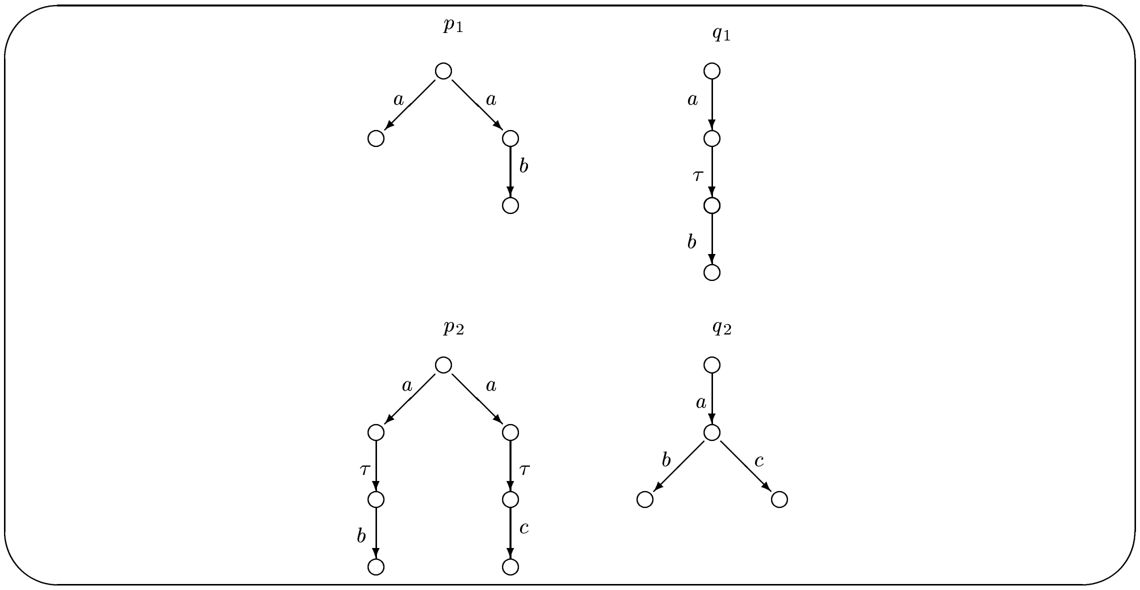


Figure 4.1: Examples of the simulation relation. We have $p_1 \leq q_1$ and $q_1 \leq p_1$, but $p_1 \not\approx q_1$. Moreover $p_2 \leq q_2$ and $q_2 \not\leq p_2$.

Definition 4.1 *A relation $\mathcal{R}$ between states is a simulation if for each $(p, q) \in \mathcal{R}$ and for each $a \in Act \cup \{\tau\}$:*

if $p \xrightarrow{a} p'$ then there exists $q' : q \overset{a}{\Longrightarrow} q'$ and $(p', q') \in \mathcal{R}$.

We write $p \leq q$ (i.e. q is more general than p) if a simulation relation $\mathcal{R}$ exists such that $(p, q) \in \mathcal{R}$.

Some examples of the simulation relation are shown in figure 4.1.

Assume $\Lambda(q) \cap L = \emptyset$ with $L = A \cup \overline{A}$, and $\overline{A} \cap \Lambda(p) = \emptyset = \Lambda(p) \cap \overline{\Lambda(q)}$. So there is a distinction between the set of actions of the process p and the one of a possible solution of the *interface equation*. A process p is *deterministic* if for all $p' \in Der(p)$ if $p' \xrightarrow{a} p''$ and $p' \xrightarrow{a} p'''$ then $p'' = p'''$, and is *rigid* if for all $p' \in Der(p), p' \not\xrightarrow{\tau}$.

Definition 4.2 *For $(p', q'), (p'', q'') \in Der(p) \times Der(q)$ and $\mu \neq \tau$:*
(a) $(p', q') \xrightarrow{\mu}_I (p'', q'')$ iff *$p' \xrightarrow{\mu} p'', q' \xrightarrow{\mu} q''$,*
(b) $(p', q') \xrightarrow{\tau}_P (p'', q'')$ iff *$p' \xrightarrow{\tau} p''$ and $q' = q''$.*
We call $\longrightarrow_{I\tau}$ the reflexive and transitive closure of $(\xrightarrow{\mu}_I \cup \xrightarrow{\tau}_P)$.

Definition 4.3 *Let $B_{I\tau}(p', q') = \{(p'', q'') | (p', q') \longrightarrow_{I\tau} (p'', q'')\}$.*

The importance of the sets $B_{I\tau}$ is shown by the following proposition.

Proposition 4.1 *If $p \|_L x \approx q$ and q is deterministic and rigid, then for every $(p', q') \in B_{I\tau}(p, q)$ we have $p' \|_L x \approx q'$.*

This leads to represent states of the solution as set of pairs of the state space of p and q, where every pair is an instance of an interface equation that is solved by a process with the state represented by the set as initial state. Assume $\Psi(p, q)$ to be the set of possible representations of state of solutions i.e. $\{\cup_{(p',q')\in X} B_{I\tau}(p', q') | X \subseteq Der(p) \times Der(q)\}$. Shields has studied the kind of systems corresponding to states of a solution of interface equations and characterized them in a strict manner. Firstly he equips elements of $\Psi(p, q)$ with a transition structure that should reflect the transition structure of a solution R.

Definition 4.4 *For $(p', q') \in Der(p) \times Der(q)$:*
(a) $(p', q') \xrightarrow{\mu}_O (p'', q'')$ *if $p' = p''$ and $q' \xrightarrow{\mu} q''$ and $\mu \in \Lambda(q) \backslash \Lambda(p)$ and $\mu \neq \tau$,*
(b) $(p', q') \xrightarrow{\mu}_C (p'', q'')$ *if $q' = q''$ and $p' \xrightarrow{\overline{\mu}} p''$ and $\mu \in \overline{A}$ and $\mu \neq \tau$.*

Definition 4.5 *For $K, K' \in \Psi(p, q)$:*
(a) $K \xrightarrow{\mu}_O K'$ iff *for all $(p', q') \in K$, $(p', q') \xrightarrow{\mu}_O (p'', q'')$ and $(p'', q'') \in K'$.*
(b) $K \xrightarrow{\mu}_C K'$ iff
(i) there exists $(p', q') \in K$ and $(p'', q'') \in K'$ s.t. $(p', q') \xrightarrow{\mu}_C (p'', q'')$,
(ii) for all $(p', q') \in K$ if $(p', q') \xrightarrow{\mu}_C (p'', q'')$ then $(p'', q'') \in K'$.

Only a particular kind of subsets of $\Psi(p, q)$ reflects the solution of an interface equation $p\|_L x \approx q$. These sets are called *uncompromised* systems, i.e. they are $I - complete$ and $O - complete$ in the following sense.

Definition 4.6 *Assume $S \subseteq \Psi(p, q)$ and $K \in S$. The set K is $I - complete$* iff *for all $(p', q') \in K$ if $p' \xrightarrow{\mu} p''$ and $\mu \notin A \cup \{\tau\}$ then $q' \xrightarrow{\mu} q''$*

Definition 4.7 *Assume $S \subseteq \Psi(p, q)$ and $K \in S$. The set K is $O - complete$ w.r.t. to S* iff *for all $(p', q') \in K$ if $q' \xrightarrow{\mu} q''$ and $\mu \neq \tau$ then there exist $n \geq 0$,$\mu_1, \ldots, \mu_n \in A$, K_0,$\ldots, K_n, K'' \in S$ and p_0,$\ldots, p_n \in Der(p)$ such that:*
(1) $p' = p_0 \overset{\mu_1}{\Longrightarrow} p_1, \ldots, p_{n-1} \overset{\mu_n}{\Longrightarrow} p_n$
(2) $K = K_0 \xrightarrow{\overline{\mu_1}}_C K_1 \ldots K_{n-1} \xrightarrow{\overline{\mu_n}}_C K_n$
(3) Either
(a) $p_n \xrightarrow{\mu} p''$ and $K_n = K''$ and $(p'', q'') \in K''$ or
(b) $p_n = p''$ and $K_n \xrightarrow{\mu}_O K''$ and $(p'', q'') \in K''$

Intuitively if $K \in S \subseteq \Psi(p, q)$ is not $I - complete$, then $(p, q) \in K$ exists where $p \xrightarrow{\mu} p', \mu \notin L \cup \{\tau\}$ and $q \not\xrightarrow{\mu}$, but there is no solution to this *interface equation*. While *I-completeness* is a property of elements of $\Psi(p, q)$, *O-completeness* is a property of subsets. Informally it says that if there exists a pair $(p, q) \in K$ and $q \xrightarrow{\mu} q'$ then through a sequence of synchronizations, the whole system goes to a state in which μ action can be performed. The importance of this characterization is shown by the following theorem.

Theorem 4.1 **[Shields]** *An interface equation $p\|_L x \approx q$ has a solution* iff *$\Psi(p, q)$ has an* uncompromised *system.*

4.3 An algorithm for submodule construction

Now we present the algorithm proposed by Qin and Lewis in a formulation that uses the concepts of Shields we have recalled in the previous section. This will allow us to state easily a proposition which permits a modification of the algorithm with an improvement in complexity. Let $Ex(X) = \Lambda(q) \backslash \Lambda(p)$ be the set of external actions and $Comm(X) = \overline{\Lambda(p)} \cap L$ be the communication actions of the possible solution. A *box* consists of a set $K \in \Psi(p, q)$ together with a set of ports.

Intuitively, the algorithm, after checking whether the equation is obviously unsolvable ($Step$ 1), tries to find the *uncompromised* system that reflects the most general solution. In $Step$ 2 every Box containing $K \in \Psi(p, q)$ *I-complete*, which is reachable from the initial Box that contains $B_{I\tau}(p, q)$, is considered; reachable Boxes whose K is not $I - complete$ are marked BAD. The system constructed up to this point (by abstracting away BAD Boxes) is $I - complete$, and a possible *uncompromised* system that reflects the most general solution must be a subset of this system, if a solution exists. Then the system is refined, boxes whose set of pairs is not $O - complete$ w.r.t. the current system are marked BAD ($Step$ 3), until an *uncompromised* system is found or box X_0 is marked BAD. In $Step$ 4 the LTS for the possible solution is constructed. Note that a box X_j has a port $u \in Ex(X)$ not marked BAD if all pairs in X_j can perform a u action, while this is not necessary for actions in $Comm(X)$. In the following algorithm there is a reference to the *anywhere* state. Every process may be substituted to this state and the resulting process is still a solution.

Step 1 Generate all $B_{I\tau}$ sets. If $B_{I\tau}(p, q)$ is not $I - complete$ then finish and report failure.

Step 2 Associate to each box X that will be created in the following a set of ports labeled with actions in Ex(X) and Comm(X). Create an initial Box X_0 with pairs $B_{I\tau}(p, q)$, and mark box X_0 *unprocessed*. Do the following until there are no *unprocessed* boxes X_i:

0 Remove the mark *unprocessed* from Box X_i; mark Box X_i BAD if it is not $I - complete$ else do (1) and (2) below.

1 For each symbol $u \in Ex(X)$, do the following:
Let $X_i(u) = \cup\{B_{I\tau}(p', q'') | (p', q') \in X_i, q' \xrightarrow{u} q''\}$.
If $(p', q') \in X_i$ exists such that $q' \not\xrightarrow{u}$ then mark port u on box X_i BAD else do (a) and (b) below:

(a) Check whether there is a box X_j containing exactly all pairs in $X_i(u)$. If not, create such a box Xj and mark it *unprocessed*.

(b) Create a transition (X_i, u, X_j). Create a link from $(p', q') \in X_i$ to $(p', q'') \in X_j$ labeled with u.

2 For each symbol $\overline{\lambda} \in Comm(X)$, do the following:
Let $X_i(\lambda) = \cup\{B_{I\tau}(p'', q')|(p', q') \in X_i, p' \xrightarrow{\lambda} p''\}$.
If there is no pair $(p', q') \in X_i$ such that $p' \xrightarrow{\lambda}$ then create a transition $(X_i, \overline{\lambda}, anywhere)$ else do (a) and (b) below:

(a) Check whether there is a box X_j containing exactly all pairs in $X_i(u)$. If not, create such a box Xj and mark it *unprocessed*.

(b) Create a transition $(X_i, \overline{\lambda}, X_j)$. If $(p', q') \in X_i$ and $p' \xrightarrow{\lambda} p''$ then create a link from $(p', q') \in X_i$ to $(p'', q') \in X_j$ labeled with $\overline{\lambda}$.

Step 3 Repeat:

(1) If there is a u transition ($u \in Ex(X) \cup Comm(X)$) from a Box X_i to a BAD box X_j, mark port u on X_i BAD. Delete all u transitions from X_i. Delete all u links from pairs in X_i.

(2) Mark box X_i BAD if there is a pair $(p', q') \in X_i$ such that $q' \xrightarrow{u} q''$ for some $u \in \Lambda(q)$, but there does not exist a sequence of action links $\lambda_1, \ldots, \lambda_n \in Comm(X) \cup \{\tau\}$ such that $(p', q') \xrightarrow{\lambda_1} (p'_1, q') \xrightarrow{\lambda_2} \ldots \xrightarrow{\lambda_n} (p'_n, q') \xrightarrow{u} (p'', q'')$ for some $n \geq 0$.

until no new BAD boxes are found in (1) and (2) above.

Step 4 If initial box X_0 is marked BAD then report "Not Solvable". Otherwise construct a transition graph for the process R as follows:

1. The initial state of R is X_0.

2. The transitions of R are the remaining ones between good boxes and the transitions to the state *anywhere* state from good boxes.

3. The states of R are good boxes reachable from X_0.

4. The actions of R are $Ex(X) \cup Comm(X)$.

Report R.

In [81] the authors prove the following proposition.

Proposition 4.2 *The solution R reported by the algorithm, if it exists, is the most general solution to $p\|_L x \approx q$.*

A complexity of $O(2^{|p| \times |q|})$ in time and space is claimed, where $|p|$ is the cardinality of the set of states of a process p. For our purposes it is sufficient to prove the following proposition, which relates the theory of Shields to the results given by the algorithm of Qin and Lewis.

Proposition 4.3 *The sets of good boxes reported by the algorithm is an* uncompromised *system.*

Proof: Let S be the set of good boxes reported by the algorithm. In $Step\ 2.0$ every box $X_j \in S$ whose set of pairs is not $I - complete$ is marked BAD. It is not possible that there exists $X_j \in S$ that is not $O - complete$ w.r.t. S, in fact in $Step\ 3.2$ it would be marked BAD. □

4.3.1 An improvement

In this subsection we study the theory of Shields and we show how to improve the previous algorithm.

Proposition 4.4 *If p, q are deterministic and rigid then $p \approx q$* iff $Traces(p) = Traces(q)$.

Proof: Since in general we have that $p \approx q$ implies $Traces(p) = Traces(q)$ we will concentrate on the other direction. Let us see the other implication. The following relation is a *weak* bisimulation:

$$\mathcal{R} = \{(p, q) \mid p, q \text{ rigid and deterministic and } Traces(p) = Traces(q)\}$$

In fact $\mathcal{R}$ is symmetric and suppose $(p, q) \in \mathcal{R}$ then:

- if $p \xrightarrow{a} p'$ then since $Traces(p) = Traces(q)$ there exists q' s.t. $q \xrightarrow{a} q'$. By contradiction we prove that $Traces(p') = Traces(q')$. In fact without loss of generality we can suppose that $p' \overset{s}{\Longrightarrow}$ and $q' \overset{s}{\not\Longrightarrow}$ and we will prove that in this case we have also $Traces(p) \neq Traces(q)$. The process p can perform the sequence as but this is not true for q, since the only $a - successor$ of q can be q' and $q' \overset{s}{\not\Longrightarrow}$.

□

Lemma 4.1 *If $S \subseteq \psi(p, q)$ is an* uncompromised *system then for every $K \in S$ if $(p, q), (p, q') \in K$ and $p = p_0 \overset{\mu_1}{\Longrightarrow} p_1, \ldots, p_{n-1} \overset{\mu_n}{\Longrightarrow} p_n$ and $K = K_0 \xrightarrow{\overline{\mu_1}}_C K_1 \ldots K_{n-1} \xrightarrow{\overline{\mu_n}}_C K_n$ then $(p_n, q), (p_n, q') \in K_n$.*

Proof: By induction on n. For $n = 0$ it trivially follows. Let us see the inductive step where $n = n' + 1$ and the lemma is true by inductive hypothesis for n'. Then we know that $(p_{n'}, q), (p_{n'}, q') \in K_{n'}$ and $p_{n'} \xrightarrow{\tau}_* p^1_{n'} \xrightarrow{\mu_n} p^2_n \xrightarrow{\tau}_* p_n$. Since $(p_{n'}, q) \in K_{n'}$ then $(p^1_{n'}, q) \in K_{n'}$ (since $B_{I\tau}(p_{n'}, q) \subseteq K_{n'}$), $(p^2_n, q) \in K_n$ (by the hypothesis $K_{n'} \xrightarrow{\overline{\mu_n}}_C K_n$) and at least $(p_n, q) \in K_n$ (since $B_{I\tau}(p^2_n, q) \subseteq K_n$). The same reasoning can be applied to $(p_{n'}, q')$. □

Lemma 4.2 *If $S \subseteq \psi(p, q)$ is an* uncompromised *system then for every $K \in S$ if $(p, q), (p, q') \in K$ and $q \overset{s}{\Longrightarrow} q_1, q' \overset{s}{\Longrightarrow} q'_1$ then there exist $K' \in S$ and $p' \in Der(p)$ s.t. $(p', q_1), (p', q'_1) \in K'$.*

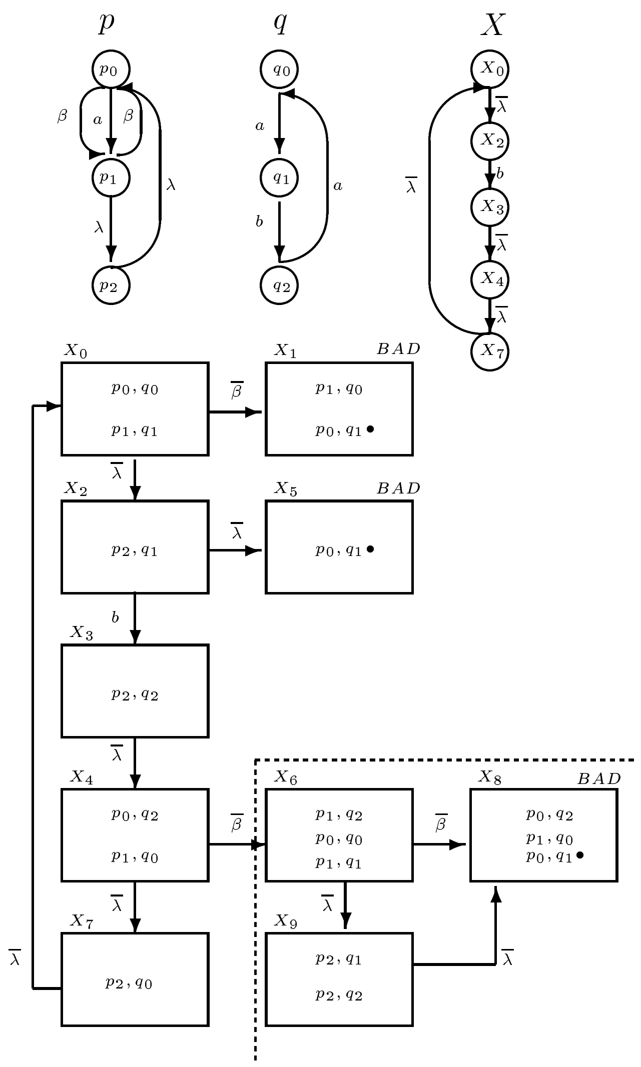


Figure 4.2: An example taken from [81], boxes under dotted line are not considered in the improved algorithm. Transitions to the *Anyware* state are not shown.

Proof: By induction on the lenght of s. If lenght of s is 0 the the result trivially follows. Let us see the inductive step and suppose $s = s'a$. If $q \stackrel{s}{\Longrightarrow} q_1$ and $q' \stackrel{s}{\Longrightarrow} q'_1$ then we know there exist $K^1 \in S, p^1 \in Der(p)$ s.t. $q \stackrel{s'}{\Longrightarrow} q_1^1$ and $q' \stackrel{s'}{\Longrightarrow} q_1'^1$ with $(p^1, q_1^1), (p^1, q_1'^1) \in K^1$. Since q, q' are deterministic then we have $q_1^1 \xrightarrow{a} q_1$ and $q_1'^1 \xrightarrow{a} q'_1$. Since $K^1 \in S, q_1^1 \xrightarrow{a} q_1$ and S is *O-complete* it follows that there exist $p^1 = p_0 \stackrel{\mu_1}{\Longrightarrow} p_1, \ldots, p_{n-1} \stackrel{\mu_n}{\Longrightarrow} p_n$ and $K = K_0 \xrightarrow{\overline{\mu_1}}_C K_1 \ldots K_{n-1} \xrightarrow{\overline{\mu_n}}_C K_n$ and either (a) $p_n \xrightarrow{a} p''$ and $K_n = K''$ and $(p'', q_1) \in K''$ or (b) $p_n = p''$ and $K_n \xrightarrow{\mu}_O K''$ and $(p'', q_1) \in K''$. By lemma 4.1 we have $(p_n, q), (p_n, q') \in K_n$. Then if $a \in \Lambda(q)\backslash\Lambda(p)$ case (a) is excluded and so case (b) remains. But in this case let K' be K'' and p' be p'' then we have the thesis, since if $K_n \xrightarrow{\mu}_O K''$ then also (p_n, q'_1) must be in K'' (see definition of $\longrightarrow_O$ relation). In the other case if $a \in \Lambda(q) \cap \Lambda(p)$ the case (b) can be excluded since $(p_n, q_1^1) \not\xrightarrow{a}_O$ and so only case (a) remains. But in this case let K' be K_n and p' be p'' hence we have the thesis, since $(p_n, q_1'^1) \in K''$ and $(p'', q'_1) \in B_{I\tau}(p_n, q_1'^1)$. □

In the following proposition we formulate a property of *uncompromised* systems that permits us to improve the complexity of the algorithm.

Proposition 4.5 *If $S \subseteq \psi(p, q)$ is an* uncompromised *system then for every $K \in S$ if $(p, q), (p, q') \in K$ then $q \approx q'$.*

Proof: By contradiction, suppose that $K \in S$ exists with $(p, q), (p, q') \in K$ such that $q \not\approx q'$. First of all suppose that $q \xrightarrow{\mu}, q' \not\xrightarrow{\mu}, \mu \in \Lambda(q)\backslash\Lambda(p)$. Then since K is $O - complete$ w.r.t. S there exist $p = p_0 \stackrel{\mu_1}{\Longrightarrow} p_1, \ldots, p_{n-1} \stackrel{\mu_n}{\Longrightarrow} p_n$ and $K = K_0 \xrightarrow{\overline{\mu_1}}_C K_1 \ldots K_{n-1} \xrightarrow{\overline{\mu_n}}_C K_n$ and either (a) $p_n \xrightarrow{\mu} p''$ and $K_n = K''$ and $(p'', q'') \in K''$ or (b) $p_n = p''$ and $K_n \xrightarrow{\mu}_O K''$ and $(p'', q'') \in K''$. But since $\mu \in \Lambda(q)\backslash\Lambda(p)$ case (a) can be excluded. Let us consider case (b). We can see by lemma 4.1 that $(p_n, q), (p_n, q') \in K_n$ so $K_n \xrightarrow{\mu}_O$ is not possible since $(p_n, q') \not\xrightarrow{\mu}$.
Now suppose that $q \xrightarrow{\mu}, q' \not\xrightarrow{\mu}, \mu \in \Lambda(p) \cap \Lambda(q)$; since K must be $I - complete$ then $p \not\xrightarrow{\mu}$, otherwise $q' \xrightarrow{\mu}$. Following a similar reasoning as above, we can exclude case (b) since $\mu \in \Lambda(p)$, but we have to exclude case (a) too, since in case (a) $p_n \xrightarrow{\mu}$ would follow but K_n could not be $I - complete$.
The last possibility is that there exists $s \in \Lambda(q)^*$ such that $q \stackrel{s}{\Longrightarrow} q_1$ and $q' \stackrel{s}{\Longrightarrow} q'_1$ and q_1, q'_1 are in one of the situations considered above (q, q' are deterministic and rigid so $q \approx q'$ *iff*$_{proposition 4.4}$ they have the same traces). Then by lemma 4.2 there exist $p_1 \in Der(p)$ and a set $K' \in S$ such that $(p_1, q_1), (p_1, q'_1) \in K'$; so we are again in the cases considered above. □

Proposition 4.5 suggests to modify the algorithm, by checking for every X_i the condition on *uncompromised* systems, i.e. whether a box X_i has two pairs $(p, q), (p, q')$ and $q \not\approx q'$. In this case we are sure that box X_i will be marked BAD in $Step$ 3, and so it will not be considered in $Step\ 4$. Besides, if (X_i, u, X_j) and X_i is marked BAD then this transition will not be considered in Step 4.4 since X_i is BAD, so even if X_j is not marked BAD it will not be taken into account in building the solution R (surely this is not the case if from another "good" state there is a transition u' to X_j). Since by applying the algorithm proposed in [47] it is possible to decide in polynomial time in the state space of q whether $q' \approx q''$ or not for every $q', q'' \in Der(q)$, it easy to put a check on $Step$ 2.1, 2.2 and mark directly BAD a box that does not satisfy the condition of proposition 4.5.

Example 4.1 *In Fig.4.2 we consider an example taken from [81]. All boxes X_i are generated in Steps $1-2$ and in Step 3 boxes X_1, X_5, X_6, X_8, X_9 are marked BAD, but X_6 is a box whose pairs cannot be in a "good" box, by proposition 4.5. So at the moment in which X_6 is created in Step 2, one is sure that in Step 3 it will be marked BAD. With the improved algorithm X_6 is directly marked BAD, so it is not* processed *and boxes X_8, X_9 are not generated.*

The new procedure permits to reduce the space of possible solutions. In fact, we have that $|\Psi(p,q)| = 2^{|p|\times|q|}$, but we have seen that *uncompromised* systems cannot have a set K with $(p,q),(p,q') \in K$ and $q \not\approx q'$ so we have to find states of solutions on a subset Ψ' of $\Psi(p,q)$, where $\Psi' = \{K | K \in \Psi(p,q) \text{ and } (p,q),(p,q') \in K \Rightarrow q \approx q'\}$. Given a process q_1, by applying the algorithm in [47], it is possible to get a process q such that $q_1 \approx q$ and for every $q', q'' \in Der(q)$ $q' \not\approx q''$. So we can strengthen $\approx$ to equality in definition of Ψ'. Every $K \in \Psi'$ can be seen as a total function from a subset of the states of p (i.e. the first elements in the couples of K) to the states of q. Let $\binom{n}{i}$ be the number of different subsets of cardinality i of a set of cardinality n. The number of total functions from a finite set A to another finite set B is $|B|^{|A|}$. This leads to the following approximation:

$$|\Psi'| \simeq \sum_{i=0}^{|p|} \binom{|p|}{i} |q|^i \leq 2^{|p|} \sum_{i=0}^{|p|} |q|^i \simeq 2^{|p|\times \log|q|}$$

By the above considerations we can state the following theorem.

Theorem 4.2 *The improved algorithm has complexity $O(2^{|p|\times\log|q|})$ in time and space.*

4.4 Conclusions

In this chapter we have analyzed another problem that can be modeled through unspecification, namely finding the solution of interface equations or submodule construction. In the next chapter a property expressed with a similar schema, but with a universal quantification, is used to model security aspects of systems.

We have refined ad hoc techniques for the solution of interface equations, for a restricted class of processes. In particular when the specification is a deterministic process.

Actually we have found a property of *uncompromised* systems, by studying the treatment of the problem given by Shields. Hence we have shown how to use this property for improving the complexity of existing algorithms, in particular the one of Qin and Lewis. This algorithm synthesizes solutions which present a good property, since they are the most general ones.

An alternative approach for the automatic synthesis of reactive systems is based on temporal logic satisfiability procedures (see for example [28]). In our context, a naive approach could be the use of compositional analysis techniques. Roughly, if we consider a process q it is possible to define a $\mu-$calculus formula ϕ_q, such that for every process p we have $p \approx q$ iff $p \models \phi_q$ (see [4, 91]). Hence, the interface equation problem can be rephrased in the following way:

$$\exists X : (p \| X) \setminus L \models \phi_q .$$

Now it is clear that by applying compositional analysis techniques, we can reduce this problem to the satisfiability problem in $\mu-$calculus. Unfortunately, an optimistic view of the complexity of this strategy is $O(2^{|p|^2 \times |q|})$ (see section 3.6 for a discussion). Hence *ad hoc* techniques can produce more efficient procedures. Nevertheless, how we try to put in evidence this thesis, the compositional analysis techniques, which can be applied to solve this synthesis problem, are actually very flexible and may be applied in many other settings.

Part II

Applications to Security Properties

Chapter 5

Analysis of *non interference*

In this chapter we study some information flow properties, in particular we recall the non interference analysis proposed by Focardi and Gorrieri in [31, 34, 35].

They define a family of security properties. We show that the most interesting property among them is conceptually related to a module checking problem. Hence, we show that compositional analysis techniques can be used in this framework. We provide an extension of the Focardi and Gorrieri's approach for modeling non interference properties, in a framework where simple real-time constraints must be taken into account. The techniques developed in chapter 2 are used to solve this kind of properties too. Hence, we present a tool for ensuring that a system enjoys such security properties.

5.1 Introduction

One of the typical problems in computer security is the necessity to guarantee that only legitimate users can access some kind of information.

A well known approach to face this problem is the *Multilevel Security* model ([8]), which is a policy for managing objects at various levels of secrecy (or confidentiality). Every object and every user is bound to a secrecy level and the information flow can be directed only from low users to higher users. The system achieves this aim by permitting neither *write-down* nor *read-up* actions. So the information flow should directly go from lower processes to higher ones. As remarked in [35], this solution is still not satisfactory. Even though, at a first glance, there is no direct way to transmit high level information to lower users, there could be a *covert* channel.

If the low and high users share a bounded common resource, a high level user can establish a communication with a low user simply by filling or emptying the resource. An error message of the operating system will make the link between low and high users. Moreover, if low and high users access common files in mutual exclusion, then a high level user can modify the view of the system perceived by low level users simply by accessing the file in a controlled manner. In these cases, the high users can interfere with the low users, and cause different status of the system in which they operate to be perceived.

This difference is a kind of information that can be exploited to send messages from high users to lower. A drastic solution to this kind of problems is to avoid these possible interferences. A lot of "Non Interference" definitions have been proposed in literature (for the first time in [36]), for quite different formal models of interaction between users (processes).

In [34] Focardi and Gorrieri have translated some of these definitions into the *process algebra* context ([70]). In this way, a lot of well established techniques for specification and verification of program properties can be used. Besides, it is possible, in the same framework, to check functional correctness of the system and security properties. In [31, 34, 35] Focardi and Gorrieri propose a family of information flow security properties called *Non Deducibility on Compositions* (NDC, for short). Intuitively, a system is NDC if, by *interacting* with every possible high level users, it always *appears* the same to low level users. No information at all can be deduced by low level users. The above idea can be instantiated in a lot of ways, by choosing a particular way of interacting between systems and various criteria of equivalence.

In particular Focardi and Gorrieri argue that $BNDC$ (namely NDC when the equivalence considered is weak bisimulation) is the right choice (see [35] for the motivations), but neither a method nor a decidability result is provided for this property. Instead, an approximation is defined which is easy to check (in polynomial time in the size of the system description) and compositional.

The advantages of Focardi and Gorrieri's idea of rephrasing security concepts in a well established framework, such as *process algebra* theory, are put in evidence. This facility is due to the great amount of research work that has been performed in the field of timed process algebras. The flexibility of our methodology for the analysis of NDC like properties, is shown. In fact, the partial evaluation techniques (see chapter 2) can be exploited in this technical framework too.

5.2 Formal model and security properties

We recall *Security Process Algebra* (SPA, for short), the formal model used in [34, 35] for the description of the behaviour of systems, which is a variant of well established models in concurrency theory (see [70]). We have a finite set of visible actions $\mathcal{L}$, union of input actions $I = \{a, b, c \ldots\}$ and output actions $O = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$, a special action τ (which models internal computation, i.e. not visible outside the system), a complementation function $\overline{(\_)} : \mathcal{L} \mapsto \mathcal{L}$, such that $\forall a \in \mathcal{L} : \overline{\overline{a}} = a$ and $\overline{\tau} = \tau$. To reflect different levels of secrecy the set $\mathcal{L}$ of visible actions is partitioned into two sets $ActH$(or H) and $ActL$, closed by complementation function. Let Act be $\mathcal{L} \cup \{\tau\}$. We now describe the syntax for SPA terms and we give their formal semantics by Labeled Transition Systems. The syntax is the following:

$$E ::= \mathbf{0} \mid a.E \mid E_1 + E_2 \mid E_1 \| E_2 \mid E/L \mid E \backslash L \mid E[f] \mid Z$$

$$\frac{}{a.E \xrightarrow{a} E}$$

$$\frac{E_1 \xrightarrow{a} E_1'}{E_1 + E_2 \xrightarrow{a} E_1'} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1 + E_2 \xrightarrow{a} E_2'}$$

$$\frac{E_1 \xrightarrow{a} E_1'}{E_1 \| E_2 \xrightarrow{a} E_1' \| E_2} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1 \| E_2 \xrightarrow{a} E_1 \| E_2'}$$

$$\frac{E_1 \xrightarrow{l} E_1' \quad E_2 \xrightarrow{\bar{l}} E_2'}{E_1 \| E_2 \xrightarrow{\tau} E_1' \| E_2'}(l \in \mathcal{L}) \qquad \frac{E_1 \xrightarrow{a} E_1'}{E_1[f] \xrightarrow{f(a)} E_1'[f]}$$

$$\frac{Z \Longleftarrow E \quad E \xrightarrow{a} E'}{Z \xrightarrow{a} E'} \qquad \frac{E_1 \xrightarrow{a} E_1'}{E_1 \backslash L \xrightarrow{a} E_1' \backslash L}(a \notin L \cup \overline{L})$$

$$\frac{E_1 \xrightarrow{a} E_1'}{E_1 / L \xrightarrow{a} E_1' / L}(a \notin L \cup \overline{L}) \qquad \frac{E_1 \xrightarrow{a} E_1'}{E_1 / L \xrightarrow{\tau} E_1' / L}(a \in L \cup \overline{L})$$

Figure 5.1: The operational semantics of SPA terms.

where $a \in Act$, $L \subseteq \mathcal{L}$, $f : Act \mapsto Act$, with $f(\tau) = \tau$ and Z is a process constant that must be associated with a definition $Z \Longleftarrow E$.

The semantics of SPA closed terms (processes) is given by the least set of action relations induced by the rules in figure 5.1.

Informally **0** (or Nil) is a process that does nothing; $a.E$ is a process that can perform an a action and then behaves as E; $E_1 + E_2$ represents the non deterministic choice between the two processes E_1 and E_2; $E_1 \| E_2$ is the parallel composition of processes that can proceed in an asynchronous way but they must synchronize to make a communication, represented by an internal action τ (this models an handshake communication); $E[f]$ is the process E whose actions are renamed *via* f; $E \backslash L$ is the process E prevented from performing actions in L, and E/L is the process E whose actions in L are transformed into τ (or invisible) actions.

As equivalence relation among SPA terms we use (weak) bisimulation $\approx$ (see figure 5.2 for some examples and chapter 2).

Let $Sort(E) = \{a | a \in Act, \exists E' \in Der(E), E' \xrightarrow{a}\}$, and $\mathcal{E}_H = \{\Pi | Sort(\Pi) \subseteq H \cup \{\tau\}\}$.

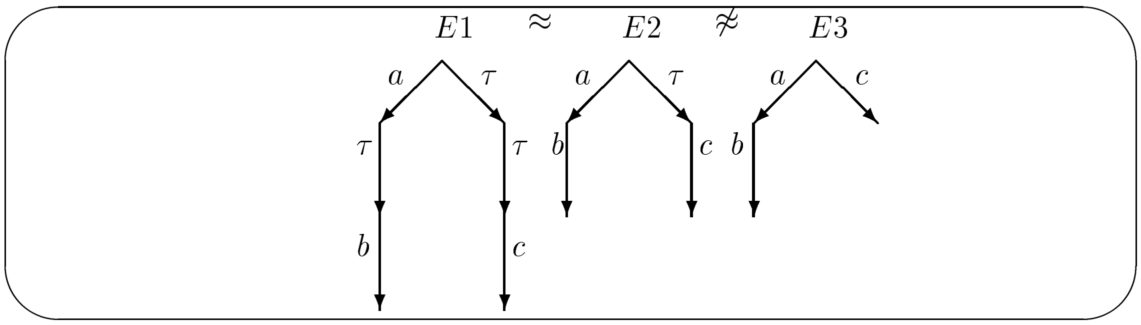


Figure 5.2: Examples of equivalent and inequivalent processes.

5.2.1 Security properties

The underlying idea of NDC for ensuring non interference between high users and low users is that the system behaviour must be invariant w.r.t. the composition with every high user. Hence, there is no possibility of establishing a communication (i.e. sending information); intuitively, it is like a medium where the same signal is always present. In terms of a generic language for the description of systems, where $||$ stands for the composition operator and $\equiv$ for the equivalence relation, we have:

$$\forall \Pi \in High\, users\ E||\Pi \equiv E\ \ w.r.t.\quad Low\ users$$

This property can be instantiated by assuming different notions of composition and equivalence. The method we are going to present is suitable to manage various composition operators and equivalence criteria. In terms of SPA parallel composition operator and *bisimulation* equivalence, we have:

Definition 5.1 $E \in BNDC$ iff $\forall \Pi \in \mathcal{E}_H : (E||\Pi)\backslash H \approx E\backslash H$.

The above definition of BNDC is the one presented in [34], whereas in [30, 35] $E\backslash H$ is replaced by E/H, but the two definitions are equivalent. This formulation makes it easier to state our results. Before discussing this property, we recall two other properties, BSNNI and SBSNNI, which approximate BNDC from above and below, respectively.

Definition 5.2 $E \in BSNNI$ iff $E\backslash H \approx E/H$.

Definition 5.3 $E \in SBSNNI$ iff $\forall E' \in Der(E) : E' \in BSNNI$.

The hiding operator $/H$ used below, where H is closed under complementation, can be replaced by the composition of the argument with the process $TopH \Longleftarrow \sum_{a\in H} a.TopH$, i.e. $E/H \approx (E||TopH)\backslash H$. This shows that $BNDC \subseteq BSNNI$, because the system is checked only for the absence of high users or for high users that can perform every action. This inclusion is strict as shown by the following example:

Example 5.1 *The process $E = l.\mathbf{0} + h.h.l.\mathbf{0}$ belongs to $BSNNI$ since $E\backslash H \approx l.\mathbf{0}$ and $E/H \approx l.\mathbf{0} + \tau.\tau.l.\mathbf{0}$. It is easy to check that $E\backslash H \approx E/H$ but $E \notin BNDC$ since $(E\|\overline{h}.\mathbf{0})\backslash H \xrightarrow{\tau} (h.l.\mathbf{0}\|\mathbf{0})\backslash H$, so it can go, through an internal action, into a state where no l action can be performed.*

By using the algorithm proposed in [47] it is possible to check both $BSNNI$ and $SBSNNI$ for a finite state process E in $O(n^\alpha)$, where α is the constant for multiplication of matrices and n the number of states of the process E. In [35] the authors propose a tool for checking $SBSNNI$ membership for finite state processes, which is able to exploit another aspect of $SBSNNI$, i.e. compositionality. In fact if $E_1, E_2 \in SBSNNI$ then $E_1\|E_2 \in SBSNNI$. This permits us to avoid, in some cases, the so called *state explosion* problem due to the presence of parallel operators in SPA terms (see [35] for more details).

A negative result

Unfortunately the same is not valid for $BNDC$, in fact we can prove the following result.

Proposition 5.1 *BNDC is not compositional.*

Proof: Let $E_1 = l.(\tau + \tau.l.\mathbf{0}) + l.\mathbf{0} + l.\overline{h}.l.\overline{h_1}.\mathbf{0}$ and $E_2 = l_1.(\tau + \tau.l_1.\mathbf{0}) + l.\mathbf{0} + l_1.h.l_1.\mathbf{0}$; we have that $E_1, E_2 \in BNDC$ but $E_1\|E_2$ is not in $BNDC$. In fact, let Π be $h.h_1.\overline{h}.\mathbf{0}$ then $((E_1\|E_2)\|\Pi)\backslash H$ can perform the sequence $l.l_1$ of actions and reach the state $((\overline{h}.l.\overline{h_1}.\mathbf{0}\|h.l_1.\mathbf{0})\|h.h_1.\overline{h}.\mathbf{0})\backslash H$. From this state, it can go into the state $E_3 = ((l.\overline{h_1}.\mathbf{0}\|l_1.\mathbf{0})\|h.h_1.\overline{h}.\mathbf{0})\backslash H$ or the state $E_4 = ((l.\overline{h_1}.\mathbf{0}\|h.l_1.\mathbf{0})\|h_1.\overline{h}.\mathbf{0})\backslash H$, by doing a synchronization action. Now, E_3 is weak bisimilar to $l\|l_1$ and E_4 to $l.l_1$.
But $(E_1\|E_2)\backslash H$ cannot produce a sequence $l.l_1$ reaching a state bisimilar to
$((\overline{h}.l.\overline{h_1}.\mathbf{0}\|h.l_1.\mathbf{0})\|h.h_1.\overline{h}.\mathbf{0})\backslash H$ which is weak bisimilar to $\tau.(l\|l_1) + \tau.l.l_1$. $\square$

5.2.2 A proof system for SBSNNI

We present a simple proof system for proving that a process is in $SBSNNI$ for finite processes (i.e. with no constant process). Let $I = \{i_1, \ldots, i_n\}$ be a finite set of indexes. Let $\sum_{i\in I} P_i$ a syntactic definition for $(P_{i_1}+(P_{i_2}+\ldots+(P_{i_n})\ldots))$. We call finite processes only SPA terms, built using $\mathbf{0}, a.E (a \in Act), E_1 + E_2$. Note that if we exclude constant process definition, every other SPA term has an equivalent (w.r.t. $\approx$) finite process (see [92]) and SBSNNI is closed by *bisimulation* equivalence, (i.e. for proving that $E \in SBSNNI$ we can prove that $E' \in SBSNNI$, where $E \approx E'$).

Proposition 5.2 *The proof system in table 5.1 is sound and complete for deciding SBSNNI membership of finite processes.*

Proof: First, we prove the soundness of the proof system w.r.t. finite processes. The soundness of the rules 1,2 is trivial. Let us study rule 3. By hypothesis we know that every $P' \in Der(P + h.Q)$, with $P' \not\equiv P + h.Q$ is $SBSNNI$. Let us see that $P + h.Q/$

$$1.\ \frac{}{\mathbf{0}\in SBSNNI} \qquad 2.\ \frac{\{P_i\in SBSNNI\}_{i\in I}\ \{l_i\in ActL\cup\{\tau\}\}_{i\in I}}{\sum_{i\in I} l_i.P_i\in SBSNNI}$$

$$3.\ \frac{P,Q\in SBSNNI\ \ P\overset{\tau}{\Longrightarrow}P'\ \ P'\backslash H\approx Q\backslash H\ \ h\in H}{h.Q+P\in SBSNNI}$$

Table 5.1: Proof system for SBSNNI

$H \approx P + h.Q\backslash H$. This is the same to prove that $P + h.Q/H \approx P\backslash H$. We show that $\approx \cup\{(P + h.Q/H, P\backslash H)\}$ is a bisimulation. The only interesting case to analyze is the couple $(P + h.Q/H, P\backslash H)$, and in particular the transition $P + h.Q/H \xrightarrow{\tau} Q/H$. By the premises of the rule we know that there exists P' s.t. $P \overset{\tau}{\Longrightarrow} P'$ and $P'\backslash H \approx Q\backslash H$, and since $Q \in SBSNNI$ we get $P'\backslash H \approx Q/H$. So we have $P\backslash H \overset{\tau}{\Longrightarrow} P'\backslash H$ and $Q/H \approx P'\backslash H$.

Now, we prove the completeness of the proof system w.r.t. finite processes. Every finite processes E may be written as

$$\sum_{i\in I} a_i.P_i$$

where every P_i for $i \in I$ is a finite process. Now we proceed by induction on the maximal length of the possible sequences of transitions of the process E.

- $n = 0$. Then $E = \mathbf{0}$ and we can apply rule 1.
- *Inductive step*. Then by inductive hypothesis we know that every P_i with $i \in I$ is in $SBSNNI$. Now we partition the index set I in three sets I_τ, I_L and I_H. The first set corresponds to the indexes of τ-derivatives of E, the second to l-derivatives of E ($l \in L$) and the third set to h-derivatives of E ($h \in H$). Hence E may be re-written as:

  $$\sum_{i\in I_\tau} \tau.P_i + \sum_{i\in I_L} l_i.P_i + \sum_{i\in I_H} h_i.P_i$$

  Now consider P be $\sum_{i\in I_\tau} \tau.P_i+\sum_{i\in I_L} l_i.P_i$, then by rule 2 we have $P \in SBSNNI$. Now by induction on $|I_H|$ we prove $E \in SBSNNI$. For $|I_H| = 1$ let E be $P + h.P_h$, we show that the premises of rule 3 are satisfied. We prove that there exists P' s.t. $P \overset{\tau}{\Longrightarrow} P'$ and $P'\backslash H \approx P_h\backslash H$. In fact, we know that $E \in SBSNNI$ we get $E\backslash H \approx E/H$, then since $E/H \xrightarrow{\tau} P_h/H$ we must have $E\backslash H \overset{\tau}{\Longrightarrow} P'\backslash H$ and $P_h/H \approx P'\backslash H$. It follows the thesis since $P_h \in SBSNNI$ and so $P_h\backslash H \approx P_h/H$. The inductive step is similar.

□

Since it is computationally easy to calculate $SBSNNI$ membership this proof system has a limited practical interest. But it gives some hints to understanding the nature and locality of this property. In fact rule 3 shows that if a process $h.Q + P$ is in $SBSNNI$ then P must be in $SBSNNI$ and performing h leads the system to a state where it could go autonomously through internal actions. So at every point in the future behaviour, the interaction (namely performing a synchronization or not) with possible *high* users does not influence the behaviour of the resulting system.

Example 5.2 *The process $l.\mathbf{0} + h.l.\mathbf{0}$, where h is a high action and l a low action is in $SBSNNI$. In fact, by rule 1 we have $\mathbf{0} \in SBSNNI$ and hence by rule 2 we have $l.\mathbf{0} \in SBSNNI$. At this point since $l.\mathbf{0}\backslash\{h\} \approx l.\mathbf{0}\backslash\{h\}$ we can use rule 3 and prove that $l.\mathbf{0} + h.l.\mathbf{0} \in SBSNNI$.*

5.2.3 BNDC like properties

Let $\leq_e$ be a preorder over processes, and suppose that $\leq_e$ is a precongruence w.r.t. $\|, \backslash L$. Let $\equiv_e = \leq_e \cap \leq_e^{-1}$ be an equivalence relation over processes; we can parameterize the definition of *Non Deducibility on Compositions* with $\equiv_e$:

Definition 5.4 $E \in eNDC$ iff $\forall \Pi \in \mathcal{E}_H : E\backslash H \equiv_e (E\|\Pi)\backslash H$.

We propose a sufficiency criterion to have a static characterization (i.e. not involving the universal predicate $\forall$) of $eNDC$.

Suppose that there exists a process $Top_H \in \mathcal{E}_H$ such that for every process $E \in \mathcal{E}_H$ we have $E \leq_e Top_H$ and a process $Nil \in \mathcal{E}_H$ such that for every process E we have $Nil \leq_e E$. Under these assumptions, we can state the following proposition:

Proposition 5.3 $E \in eNDC$ iff $E\backslash H \equiv_e (E\|Nil)\backslash H \equiv_e (E\|Top_H)\backslash H$.

Proof: $\Longrightarrow$:obvious,

$\Longleftarrow$: for every process Π with $\Pi \in \mathcal{E}_H$ we have $Nil \leq_e \Pi \leq_e Top_H$, so, since $\leq_e$ is a precongruence w.r.t $\|, \backslash H$ operators, we have:

$$\begin{aligned} E\backslash H \equiv_e (E\|Nil)\backslash H &\leq_e \\ (E\|\Pi)\backslash H &\leq_e \\ (E\|Top_H)\backslash H \equiv_e E\backslash H. \end{aligned}$$

This proves $E\backslash H \equiv_e E\|\Pi\backslash H$ and hence the thesis. $\Box$

Proposition 2.4 of [32] can be seen as an instantiation of the above proposition. The requirements of the above proposition seem to be too strong for permitting us to use a precongruence which is sensible to deadlocks (i.e. it is no possible to find suitable Top and Nil processes), so for the usual equivalences used in concurrency theory we have to develop an alternative solution. First of all we can parameterize the definition of *Non Deducibility on Composition* w.r.t. a set S of *high* users in composition with which the system is checked.

Definition 5.5 $E \in eNDC^S$ iff $\forall \Pi \in \mathcal{E}_H \cap S : E\backslash H \equiv_e (E\|\Pi)\backslash H$.

Since we consider a finite set of actions the only source of infinity is the set of states of this transition system. Let $nd = \{E | \forall E' \in Der(E), \forall a \in Act, F = \{E'' | E' \stackrel{a}{\Longrightarrow} E''\} \text{ is finite}\}$ be the set of processes that cannot perform an infinite sequence of τ actions passing through an infinite number of different states. Let $fs = \{E | Der(E) \text{ is finite}\}$ be the set of *finite state* processes.

Under certain constraints on the set S we can provide a method for reducing the verification of $BNDC^S$ membership for finite state systems to a validity problem in $\mu-$calculus ([49]).

Unfortunately, this reduction to the *satisfiability* problem for the $\mu-$calculus is not usable in practice for real systems, because of the inherent intractability of this problem, which is DEXPTIME complete (see [94]). Nevertheless, we have decided to implement a *proof checker* for the $\mu-$calculus in order to have a verifier that may help the computer to perform what is essentially an exponential task.

5.3 Decidability of $BNDC-$like properties

In this section we propose a decision procedure for BNDC like properties. It relies on compositional analysis techniques and in particular it exploits the partial model checking techniques proposed by Andersen. First, we recall the definition of characteristic formula.

5.3.1 Characteristic formula

Given a finite state process E, we present below the definition of a formula ϕ_E that is characteristic (w.r.t. *weak* bisimulation) for this process E (see [91, 92]). Let $\langle\langle\tau\rangle\rangle\phi$ be a short notation for $\mu X.\langle\tau\rangle X \vee \phi$, $[[\tau]]\phi$ for $\nu X.[\tau]X \wedge \phi$, where X is not free in ϕ. Let $\langle\langle l\rangle\rangle\phi, [[l]]\phi, (l \in \mathcal{L})$ be respectively $\langle\langle\tau\rangle\rangle\langle l\rangle\langle\langle\tau\rangle\rangle\phi, [[\tau]][l][[\tau]]\phi$. It can be shown that these derived modalities can be equivalently expressed in an equational form. Let us see the definition of the characteristic formula:

Definition 5.6 *Given a finite state process E, its characteristic formula (w.r.t. weak bisimulation) $D_E \downarrow X_E$ is defined by the following equations for every $E' \in Der(E)$, $a \in Act$ and $W = Act \backslash \{a | E' \stackrel{a}{\Longrightarrow}\}$:*

$$\begin{aligned} X_{E'} \quad =_\nu \quad & (\textstyle\bigwedge_{a,E'':E' \stackrel{a}{\Longrightarrow} E''} \langle\langle a\rangle\rangle X_{E''}) \wedge \\ & (\textstyle\bigwedge_{a:E' \stackrel{a}{\Longrightarrow}} ([[a]](\bigvee_{E'':E' \stackrel{a}{\Longrightarrow} E''} X_{E''}))) \wedge \\ & (\textstyle\bigwedge_{a \in W} [[a]]F). \end{aligned}$$

Intuitively, for every state of the process there is a variable that encodes the capabilities of that state.

Following [91, 92] if ϕ_{E_2} is characteristic for E_2 (w.r.t. $\approx$) then:

Lemma 5.1 **i** *If $E_1 \approx E_2$ then $E_1 \models \phi_{E_2}$.*

ii *If $E_1 \models \phi_{E_2}$ and E_1 is finite state then $E_1 \approx E_2$.*

A decision method based on partial model checking

Two problems arise in the verification of $BNDC^S$ membership for a process E:

i The fact that the verification involves the composition with every *high* user in S (it could not be a finite set).

ii The checking of bisimulation equivalence between two processes.

The first problem is tackled by reducing it to a validity problem which is decidable. For the second problem, since bisimulation equivalence is in general undecidable, we have to restrict ourselves to the class of *finite state* systems. Moreover, if we consider a system E that is *finite state* then it is possible that the composition with a particular *high* user in S is no longer *finite state*. The following lemmas identify a large class of systems which remain finite state when composed with a particular set of *high* users.

Lemma 5.2 *If E is a finite process and $\Pi \in \mathcal{E}_H \cap nd$, then $(E\|\Pi)\backslash H$ is a finite state process.*

Lemma 5.3 *If E is a finite state process and $\Pi \in \mathcal{E}_H \cap fs$, then $(E\|\Pi)\backslash H$ is a finite state process.*

We have now the technical tools to prove the following result:

Proposition 5.4 *$BNDC^{fs}$ is decidable for all finite state processes E.*

Proof: Let $(D_{E\backslash H} \downarrow X_{E\backslash H})$ be the characteristic formula (up to weak bisimulation) for $E\backslash H$, then:

$E \in BNDC^{fs}$ iff $\forall \Pi \in \mathcal{E}_H \cap fs: \ (E\|\Pi)\backslash H \models (D_{E\backslash H} \downarrow X_{E\backslash H})$.

In fact:

$$\begin{array}{ll} & E \in BNDC^{fs} \\ \Longrightarrow & \forall \Pi \in \mathcal{E}_H \cap fs: \ (E\|\Pi)\backslash H \approx E\backslash H \\ \Longrightarrow_{5.1.i} & \forall \Pi \in \mathcal{E}_H \cap fs: \ (E\|\Pi)\backslash H \models (D_{E\backslash H} \downarrow X_{E\backslash H}) \end{array}$$

On the other hand:

$$\begin{array}{ll} & E \notin BNDC^{fs} \\ \Longrightarrow & \exists \Pi \in \mathcal{E}_H \cap fs : (E\|\Pi)\backslash H \not\approx E\backslash H \\ \Longrightarrow_{5.3} & \exists \Pi \in \mathcal{E}_H \cap fs : ((E\|\Pi)\backslash H \text{ is finite state }) \wedge \\ & (E\|\Pi)\backslash H \not\approx E\backslash H \\ \Longrightarrow_{5.1.ii} & \exists \Pi \in \mathcal{E}_H \cap fs : (E\|\Pi)\backslash H \not\models (D_{E\backslash H} \downarrow X_{E\backslash H}) \\ \Longrightarrow & \neg(\forall \Pi \in \mathcal{E}_H \cap fs : (E\|\Pi)\backslash H \models (D_{E\backslash H} \downarrow X_{E\backslash H}) \end{array}$$

So applying lemmas 3.2 and 3.1 in the order, we have that:

$E \in BNDC^{fs}$ iff $\forall \Pi \in \mathcal{E}_H, \Pi \models ((D_{E\backslash H} \downarrow X_{E\backslash H})//\backslash L)//E.$

Now, applying the translation tr we obtain a closed formula $\phi_E = tr(((D_{E\backslash H} \downarrow X_{E\backslash H})//\backslash L)//E)$ of $\mu-$calculus.

To reduce the problem to a validity problem on $\mu-$calculus, we use the formula $Sort_H = \nu X.[H \cup \{\tau\}]X \wedge [\mathcal{L} \backslash H]F$. It is no difficult to see that $\Pi \in \mathcal{E}_H$ iff $\Pi \models Sort_H$. We can now reduce the decision problem of $BNDC^{fs}$ membership to a validity problem on $\mu-$calculus. Now we have $E \in BNDC^{fs}$ iff $Sort_H \implies \phi_E$ (i.e. $\neg SortH \vee \phi_E)$ is valid.

In fact:

$E \notin BNDC^{fs}$
$\implies \exists \Pi \in \mathcal{E}_H \cap fs\ \Pi \not\models \phi_E$
$\implies \Pi \in \mathcal{E}_H\ \Pi \not\models \phi_E$
$\implies Sort_H \implies \phi_E \text{ is not valid}$

On the other hand:

$Sort_H \implies \phi_E \text{ is not valid}$
$\implies \exists \Pi \in \mathcal{E}_H\ \Pi \not\models \phi_E$
$\implies \exists \Pi \in \mathcal{E}_H \cap fs\ \Pi \not\models \phi_E\ (\mu - \text{calculus enjoys}$
$\text{the finite model property})$
$\implies E \notin BNDC^{fs}$

Since the validity problem on $\mu-$calculus is decidable the result follows. $\square$

By means of a similar proof we can state the following proposition:

Proposition 5.5 *$BNDC^{nd}$ is decidable for all finite processes E.*

5.4 A tool for *non interference* analysis

The tool consists of three elements, one already existing *mudiv*, and two others created for this purpose, namely *spv* and μPA (see figure 5.3).

The first is an enhanced version of a tool for performing partial model checking and it is freely distributed ([58]). One of the main reasons to use this existing tool, instead of writing a simple translation procedure that implements the partial evaluation functions, is that *mudiv* performs heuristic simplifications on the assertions, that in general reduce the number of equations of the final specification, and sometimes they reduce it to a constant (i.e. True or False). Since the strategy of *mudiv* is not complete we have developed a proof assistant for $\mu-$calculus (called μPA) in Coq, by implementing the deduction system of chapter 1, that have been proven complete to establish the validity of $\mu-$calculus formulas by Walukiewicz (see [104]). Moreover, we have implemented a partial decision procedure in *spv*, which exploits the particular form of the validity problem that arises for deciding BNDC like properties. The main element *spv* (or *security property verifier*) contains the following modules:

- (*SPA translator to mudiv process specification language)* It implements the translation functions from the syntax of SPA terms, (actually for an extended version similar to the one proposed in [35]), to the simpler and more restrictive specification language of *mudiv*.

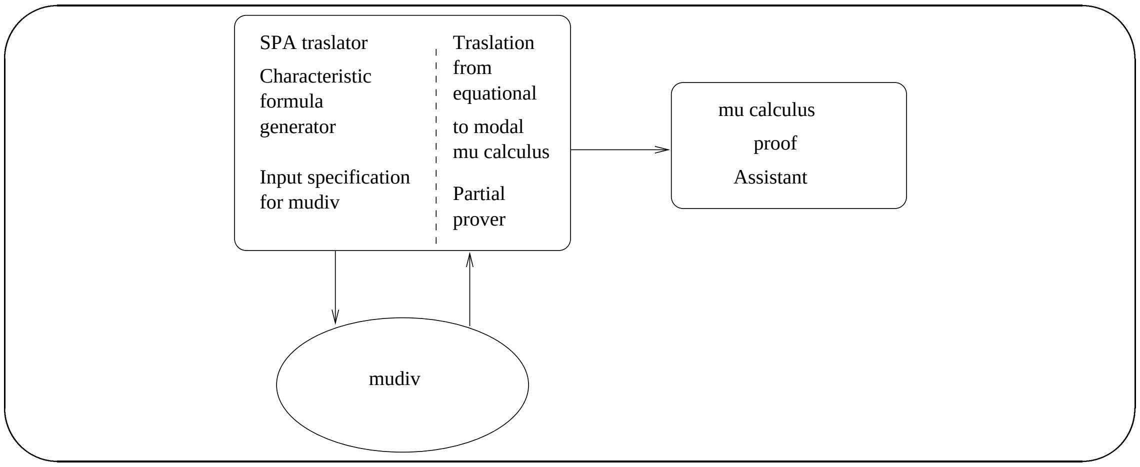

SPA traslator
Characteristic formula generator
Input specification for mudiv
Traslation from equational to modal mu calculus
Partial prover
mu calculus proof Assistant
mudiv

Figure 5.3: Tool's structure.

- (*Characteristic formula generator)* It implements the functions for building the characteristic formula for a process E and producing the correct logical specification for *mudiv*. A function that directly gets a SPA specification and produces a *mudiv* specification is provided.

- (*Translation from equational to modal $\mu-$calculus)* It implements the function tr and a lot of heuristic simplifications on the list of equations, that do not modify the meaning of the top formula but reduce its dimension. Some heuristics are similar to those of *mudiv*, and others are specialized for the particular validity problem and the form of the characteristic formula.

- *(Partial prover)* It implements a subset of the rules of [105], and some other special rules for dealing with particular kinds of formulas. It seems to give promising results, since the following examples are almost instantaneously checked with this feature of *spv* and moderately bigger examples can be solved too. Unfortunately, since it does not implement a complete strategy, sometimes it reports that a system is not $BNDC$ when actually it is in this class.

Another feature of this tool is a module that implements the algorithm of [47] for checking the bisimulation equivalence between two processes, which can be used too for the minimization (w.r.t. bisimulation) of transition systems associated to processes. In fact the process E_2 in figure 5.2 can be obtained by minimizing the transition system of the process E_1, by means of this algorithm.

Moreover, the model checker *mudiv* can be used to check the system w.r.t. particular *high* users, for trying to show that the system does not satisfy particular properties. The proof checker μPA can be used to check that a property is verified by the system in composition with *high* users in a set S of processes, provided that the membership in S of these processes can be expressed by a $\mu-$calculus formula.

Actually, in chapter 3 we have reduced the module checking problem for $\mu-$calculus, to the verification of a similar problem and this tool can be also used to analyze *module checking* problems.

Example 5.3 *A manager for an object, shared between high and low users, could be specified as follows:*

$$C = req_H.C_1 + req_L.write_L.C$$

$$C_1 = \tau.C + \overline{read_H}.C$$

Users must make a request to the manager for accessing the object (high users can read the value of the object and low users can write a value). A kind of timeout is modeled, by means of a τ action, such that the system is not obliged to wait for the communication action from an high user. It can be checked automatically that $C \in BNDC^{fs}$, (actually it is in SBSNNI).

5.5 Adding time modeling feature to SPA

In this section we show a development of *non interference* theory in the line of research proposed by Focardi and Gorrieri.

In particular we note that timing aspects of the system can be used to transmit information. Suppose to have a system that provides services both for *High* and *Low* users. Then an high user can simply make the system not disposable for interaction with low level users for a fixed amount of time and hence by revealing some kind of information. The SPA process algebra is not able to deal with a quantitative time notion, only temporal relations between actions can be expressed.

Fortunately, there has been a lot of research in *process algebra* theory for enhancing the specification languages in order to solve this problem (see [74, 99]). In this section we extend the SPA language with operators that permit to express the elapsing of time. We follow the modeling approach that is called *fictitious clock*, namely a global clock is supposed to be updated whenever all the processes in the system agree on it. A distinguished action *tick* is used to model the elapsing of one unit of time. Other approaches have been explored, in particular by considering a dense time but they are unnecessarily too complex for our analysis and present some technical problems. Actions are assumed to take no time. This is reasonable if we choose a time unit such that the actual time of an action is negligible with respect to the time unit.

Timed Security Process Algebra

In this section we introduce the Timed Security Process Algebra (tSPA, for short). In addition to SPA operators, we add the time prefixing $tick.E$ and the idling operator $\iota(E)$, where E is a $tSPA$ term. Moreover we extend the definition of the semantics of the SPA operators with the rules shown in figure 5.4.

$$\frac{}{tick.E \xrightarrow{tick} E} \qquad \frac{E \not\xrightarrow{tick}}{\iota(E) \xrightarrow{tick} \iota(E)}$$

$$\frac{E \xrightarrow{l} E'}{\iota(E) \xrightarrow{l} E'} l \notin \{tick\} \qquad \frac{E \xrightarrow{tick} E'}{\iota(E) \xrightarrow{tick} \iota(E')}$$

$$\frac{E_1 \xrightarrow{tick} E_1' \quad E_2 \xrightarrow{tick} E_2' \quad \forall a \in \mathcal{L} \quad \neg(E_1 \xrightarrow{a} \wedge E_2 \xrightarrow{\overline{a}})}{E_1 \| E_2 \xrightarrow{tick} E_1' \| E_2'}$$

$$\frac{E_1 \xrightarrow{tick} E_1' \quad E_2 \xrightarrow{tick} E_2'}{E_1 + E_2 \xrightarrow{tick} E_1' + E_2'} \qquad \frac{E \xrightarrow{h} E' \quad h \in H \cup \overline{H}}{E/H \xrightarrow{\tau} E'/H}$$

$$\frac{E \xrightarrow{l} E' \quad l \notin H \cup \overline{H}}{E/H \xrightarrow{l} E'/H} \qquad \frac{E \xrightarrow{tick} E' \quad \forall h \in H \cup \overline{H}\ E \not\xrightarrow{h}}{E/H \xrightarrow{tick} E'/H}$$

Figure 5.4: Operational semantics of new operators of *tSPA*.

Informally the ι (*idling*) operator allows the process E to wait indefinitely. At every instant of time if the process E can perform an action l then the whole system proceeds in this state, by dropping the idling behaviour. The third rule about $idling$ operator enforces the so called *time determinacy* that is a central property of timed process algebras, namely if $E \xrightarrow{tick} E'$ and $E \xrightarrow{tick} E''$ then $E' = E''$ (i.e. the associated LTS is *tick*-deterministic).

The core operator for management of time is the parallel one. As we see from the its semantics, both components must agree on performing a *tick* action. Roughly, they synchronize to let time pass. But there is a side condition, i.e. no communications are possible. This enforces the so called *maximal communication progress* assumption. Indeed, whenever a communication is ready to be performed then it must start immediately.

The hiding operator $/H$ is slightly modified with respect to the one of SPA. We avoid the possibility that the process E/H performs a *tick* action if at the same time it can perform an action in $H \cup \overline{H}$. (The rules for the *hiding* operator in figure 5.1 must be discharged).

In fact, the role of hiding operator in SPA is to represent a system as appears to low users when high activities are enabled. The same should be valid in the timed SPA context. Hence, the hiding of a system cannot let time pass in a state where the system can perform an high activity, otherwise it should not model correctly the *maximal communication progress* assumption. This is clarified by considering the process $E = \iota(h)$. Assume that $H = \{h\}$ and the standard rules for hiding, then we obtain $E/H \xrightarrow{\tau}$ and, at the same time, $E/H \xrightarrow{tick}$. Note that the process $(E \| Top_H^{tick}) \backslash H$ cannot perform a $tick$ action. Hence, $(E \| Top_H^{tick}) \backslash H \not\approx E/H$.

With our modified rules, we can formally prove that for every process E its composition with Top_H^{tick} (restricted on the high actions) is equivalent to E/H, i.e., that hiding corresponds to enabling every high level action.

Proposition 5.6 *For every process E we have:* $(E \| Top_H^{tick}) \setminus H \approx E/H$.

This technical modification permits us to recast part of the theoretical results about SPA for timed SPA.

First of all we state one of the peculiarity of timed SPA, that is common to several timed process algebras, namely *time determinacy*.

Lemma 5.4 *For every $tSPA$ process E we have:*

- *If $E \xrightarrow{tick} E'$ and $E \xrightarrow{tick} E''$ then $E' = E''$.*

Let us give the following definition of *weakly time alive* for processes.

Definition 5.7 *A process E is directly weakly time alive* iff $E \overset{tick}{\Longrightarrow}$. *A process E is weakly time alive* iff *for all $E' \in Der(E)$, we have E' is directly weakly time alive*

In the $tSPA$ model for defining NDC like properties, we do not consider all high processes for the interaction with the systems, we must restrict ourselves to *weakly time alive* processes that can perform only action in $ActH \cup \{\tau\}$. Let $\mathcal{E}_H^t$ the set of such processes.

The reason is the following: a process, that is not directly weakly time alive, may prevent time from elapsing in parallel with every system E. Hence, it can block the time flow. We want to avoid this unrealistic possibility.

As equivalence relation we still consider the observation equivalence. We use this equivalence since we can directly reuse the existing theory for the characteristic formula and the decidability procedures for $\mu-$calculi. Other finer equivalences and logics may be used for a more accurate study of the real-time aspects of the systems, nevertheless we believe that this framework is sufficient for many purposes.

We now restate the security properties proposed by Focardi and Gorrieri in this new setting. The most important property is timed Bisimulation Non Deducibility on Composition ($tBNDC$, for short).

Definition 5.8 $E \in tBNDC$ iff $\forall \Pi \in \mathcal{E}_H^t : (E \| \Pi) \backslash H \approx E \backslash H$.

As in the untimed case, we give the definition of two other properties, tBSNNI and tSBSNNI, which approximate tBNDC from above and below, respectively.

Definition 5.9 $E \in tBSNNI$ iff $E \backslash H \approx E/H$ *and* $(E \backslash H \xrightarrow{tick}$ iff $E/H \xrightarrow{tick})$.

Definition 5.10 $E \in tSBSNNI$ iff $\forall E' \in Der(E) : E' \in tBSNNI$.

The property $tBSNNI$ is slightly changed w.r.t. the original BSNNI. This is mainly due to the *maximal communication progress* property of our language. The $tBSNNI$ and $tSBSNNI$ properties are introduced for giving approximations of $tBNDC$ that are easily computable. Our final aim is to prove that a system is $tBNDC$ (actually, this is true for $tSBSNNI$). With our definition of $tBSNNI$ and hence of $tSBSNNI$ it is easy to see that $tSBSNNI \subseteq tBNDC$.

Roughly, E is $tBSNNI$ if the system $E \backslash H$, where no high level activity is allowed, behaves like system E that has all the high level actions *hidden* (i.e., transformed into internal τ actions). (The condition on $tick$ actions is merely technical and inserted only for proof purposes.)

Now, we are ready to state formally the relationships among the defined properties.

In particular we have that $tBSNNI$ is a weaker property than $tBNDC$.

Proposition 5.7 $tBNDC \subseteq tBSNNI$.

Furthermore, by following the proof technique of [35] we prove that $tSBSNNI$ is a stronger property than $tBNDC$.

Proposition 5.8 $tSBSNNI \subseteq tBNDC$.

Proof:

The proof is performed by showing that the following is a “timed” weak bisimulation, namely a weak bisimulation for processes which can perform a particular action $tick$.

$$\mathcal{R} = \{(E' \backslash H, E' \| \Pi \backslash H) \mid E' \in Der(E), \quad \Pi \in \mathcal{E}_H^t\}.$$

By inspection of possible cases, $(E' \backslash H, E' \| \Pi \backslash H) \in \mathcal{R}$ then:

- if $E' \backslash H \xrightarrow{a} E'' \backslash H, \quad a \neq tick$ then it follows that $E' \| \Pi \backslash H \xrightarrow{a} E'' \| \Pi \backslash H$, and $(E'' \backslash H, E'' \| \Pi \backslash H) \in \mathcal{R}$,

- if $E' \backslash H \xrightarrow{tick} E'' \backslash H$, then since $E' \in tSBSNNI$ and Π is weakly time alive, we have $\forall h \in H \cup \overline{H} \quad E' \not\xrightarrow{h}$ and so $E' \| \Pi \backslash H \overset{\tau}{\Longrightarrow} E' \| \Pi' \backslash H \xrightarrow{tick} E'' \| \Pi'' \backslash H$ and $(E'' \backslash H, E'' \| \Pi'' \backslash H) \in \mathcal{R}$,

- if $E' \| \Pi \backslash H \xrightarrow{a} E'' \| \Pi \backslash H, a \notin H \cup \overline{H}$ then we have $E' \backslash H \xrightarrow{a} E'' \backslash H$ and $(E'' \backslash H, E'' \| \Pi \backslash H) \in \mathcal{R}$.

- if $E' \| \Pi \backslash H \xrightarrow{\tau} E'' \| \Pi' \backslash H$, with $E' \xrightarrow{h} E''$ and $\Pi \xrightarrow{\overline{h}} \Pi'$, then $E'/H \xrightarrow{\tau} E''/H$. Since $E' \in tSBSNNI$ we have the following equivalence $E' \backslash H \approx E'/H$, hence there exists E_1 s.t. $E' \backslash H \overset{\tau}{\Longrightarrow} E_1 \backslash H$ and $E_1 \backslash H \approx E''/H$ and $E''/H \approx E'' \backslash H$. Hence we have up to weak bisimulation $(E'' \backslash H, E'' \| \Pi \backslash H) \in \mathcal{R}$.

- if $E' \| \Pi \backslash H \xrightarrow{a} E'' \| \Pi \backslash H$, then $E' \backslash H \xrightarrow{a} E'' \backslash H$, and $(E'' \backslash H, E'' \| \Pi \backslash H) \in \mathcal{R}$.

- if $E'||\Pi\backslash H \xrightarrow{tick} E''||\Pi'\backslash H, E' \xrightarrow{tick} E''$ and $\Pi \xrightarrow{tick} \Pi'$, then $E'\backslash H \xrightarrow{tick} E''\backslash H$ and $(E''\backslash H, E''||\Pi'\backslash H) \in \mathcal{R}$.

□

For $tSBSNNI$ property, we have not proven the compositionality, but for $tBNDC$ as for the corresponding property in the untimed language, we have the following negative result:

Proposition 5.9 *$tBNDC$ it is not compositional.*

Proof: By considering a counter-example similar to the one of proposition 5.1. □

5.5.1 Decidability results for $tBNDC$-like properties

In this section we show the flexibility of the methodology that we have proposed for the analysis of NDC like properties, by applying the same techniques to the analysis of $tBNDC$-like properties.

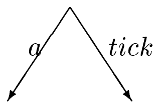
The steps we have to perform are the followings:

1. define a characteristic formula for the timed equivalence,
2. define partial evaluation techniques for the parallel operator,
3. study validity procedures when formulas are interpreted over the class of LTSs which are the semantics of $tSPA$ terms.

Point 1 can be solved by observing that our “timed equivalence” is the same as observational equivalence, and hence we can use the already defined characteristic formula for this equivalence. The crux is that we use $tick-$deterministic equational $\mu-$calculus.

Point 2 can be solved by applying our theory described in chapter 2.

Point 3 is more challenging. In fact, differently from SPA, it is not always the case that for every (finite) LTS there exists an $tSPA$ term whose semantics is this LTS. For example consider the following LTS:



This problem is due to our definition of the choice operator. We avoid this problem by considering the choice operator of high users as the standard CCS one. (This is obtained by dropping the third rule for choice in Figure 5.4) Then it is easy to see that for every finite-state LTS we can find a High user whose semantics is this LTS. Hence we consider properties that are stronger than $tBNDC$.[1]

[1]However, we feel that is possible to suitably change the validity procedure of [93] in order to consider only LTSs which are the semantics of some $tSPA$ term. We leave the proof of this conjecture as a future work.

Let us see the partial evaluation function $A//E$, where E is a $tSPA$ process, for the parallel operator of $tSPA$. It is shown in table 5.2 and can be inferred from the partial evaluation function we have studied in example 2.5.2, where we consider the same parallel operator, defined by means of GSOS rules. So we can state the following lemma, where E_1 is a finite state process and $||$ is the timed parallel operator.

Lemma 5.5 *Given a process $E_1||E_2$ and an equational specification $D{\downarrow}X$ we have:* $E_1||E_2 \models (D{\downarrow}\,X)$ iff $E_2 \models (D{\downarrow}\,X)//E_1$.

We call fs^* the set of finite-state processes, which are deterministic, weakly time alive and whose choice operator has the new semantics. (Actually, we still suppose to restrict ourselves to consider $tick$-deterministic processes, even though with this choice operator is possible to generate LTS which does not enjoy this property.)

We can prove the decidability of $tBNDC$-like properties when we only consider finite state processes fs^*.

Proposition 5.10 *For all finite state processes E the following property is decidable:*

$$\forall \Pi \in fs^* : \quad (E||\Pi) \setminus H \approx E \setminus H.$$

Proof: Similar to the proof of proposition 5.4. In particular since, we deal with $\{tick\}$–deterministic processes, we use deterministic $\mu-$calculus. Moreover, lemma 5.5 is used, and a simple condition on weakly time alive processes is needed. □

In this way we have extended the non interference analysis as proposed by Focardi and Gorrieri to deal with quantitative time aspects. Moreover, we have shown that the compositional analysis techniques studied in chapter 2 permit easily to solve the technical problems that may occur in this analysis.

5.6 An example

We rephrase the description of the Manager of a mutual exclusion variable (see Example 5.3) in $tSPA$, in this way we can take in account timing constraints. We implement the *no-write-down no-read-up* policy (see [36]). High users can only read the variable and low users can only write it. Hence, the information flow should only go from low level users to high level users. But, there could be some timing covert channels.

A first description of the manager could be the following:

$$\begin{array}{rcl} C & = & req_H.tick.C_1 + req_L.tick.write_L.tick.C + \tau.tick.tick.C \\ C_1 & = & tick.\tau.C + \iota(\overline{read_H}.C) \end{array}$$

The set H of high actions is $\{req_H, read_H\}$. Roughly, the system can receive a high request and then, after letting one unit of time pass, it can go in a state where it can serve the request for at most one unit of time or return in the initial state. Otherwise the system can receive a low request and let one unit of time pass and then let the low level user to

$$\begin{array}{rcl}
(D{\downarrow}X)//t & = & (D//t){\downarrow}X_t \\
\epsilon//t & = & \epsilon \\
(X =_\sigma AD)//t & = & ((X_s =_\sigma A//s)_{s\in Der(t)})(D)//t \\
X//t & = & X_t \\
\langle a\rangle A//s & = & \langle a\rangle(A//s) \vee \bigvee_{s\xrightarrow{a}s'} A//s', \quad \text{if } a \neq \tau, a \neq tick \\
\langle\tau\rangle A//s & = & \langle\tau\rangle(A//s) \vee \bigvee_{s\xrightarrow{\tau}s'} A//s' \vee \bigvee_{s\xrightarrow{a}s'}\langle\overline{a}\rangle(A//s') \\
\langle tick\rangle A//s & = & \begin{cases} \langle tick\rangle A//s' \wedge \wedge_{a:s\xrightarrow{a}}[\overline{a}]\mathbf{F} & s \xrightarrow{tick} s' \\ \mathbf{F} & otherwise \end{cases} \\
[a]A//s & = & [a](A//\text{s}) \wedge \bigwedge_{s\xrightarrow{a}s'} A//s', \quad \text{if } a \neq \tau, a \neq tick \\
[\tau]A//s & = & [\tau](A//\text{s}) \wedge \bigwedge_{s\xrightarrow{\tau}s'} A//s' \wedge \bigwedge_{s\xrightarrow{a}s'}[\overline{a}](A\;//s') \\
[tick]A//s & = & \begin{cases} [tick]A//s' \vee \vee_{a:s\xrightarrow{a}}\langle\overline{a}\rangle\mathbf{T} & s \xrightarrow{tick} s' \\ \mathbf{T} & otherwise \end{cases} \\
A_1 \wedge A_2//s & = & (A_1//s) \wedge (A_2//s) \\
A_1 \vee A_2//s & = & (A_1//s) \vee (A_2//s) \\
\mathbf{T}//s & = & \mathbf{T} \\
\mathbf{F}//s & = & \mathbf{F}
\end{array}$$

Table 5.2: Partial evaluation function for parallel operator || of timed SPA.

write and then let one unit of time pass and returns in its initial state. The last possibility is that the system lets two units of time pass and then returns in its initial state. This system is not $tBNDC$, in fact consider the following high process:

$$X = \iota(req_H.\iota(read_H.\iota(\mathbf{0})))$$

The manager in parallel with the high process X may have the following computation:

$$C||X\backslash H \overset{tick}{\Longrightarrow}\overset{req_L}{\Longrightarrow}$$

which cannot be performed by the process $C\backslash H$. So we refine our Manager and we let exactly one unit of time to the high level user to read the message. Please note, that after the req_H it has to wait one unit of time. So at the end, the high user takes two units of time from the request actions.

We can re-design our system, by considering a timer that can be activated by the manager, by issuing the action go and then it can be stopped by the action $stop$.

So, the new manager is the following:

$$\begin{array}{lcl}
Timer & = & \iota(go.tick.\iota(\overline{stop}.Timer)) \\
C & = & req_H.tick.\overline{go}.(\iota(stop.C) + \iota(\overline{read_H}.\iota(stop.C))) \\
 & & +req_L.tick.write_L.tick.C + \tau.tick.tick.C \\
Manager & = & (Timer||C)\backslash\{go, stop\}
\end{array}$$

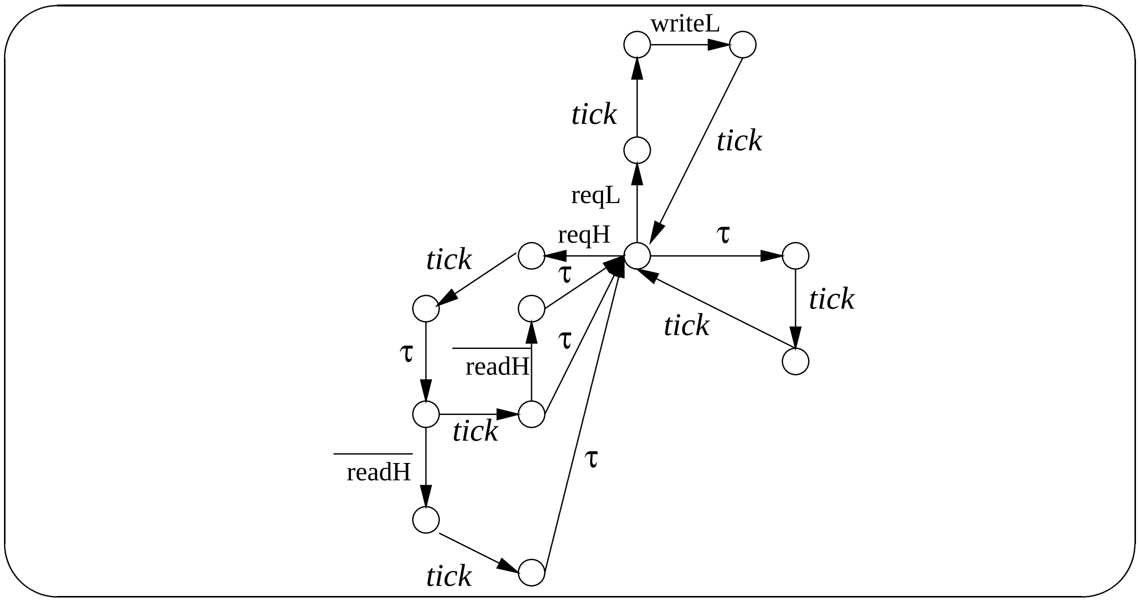


Figure 5.5: The behaviour of the process $Manager$, represented by a Labeled Transition System.

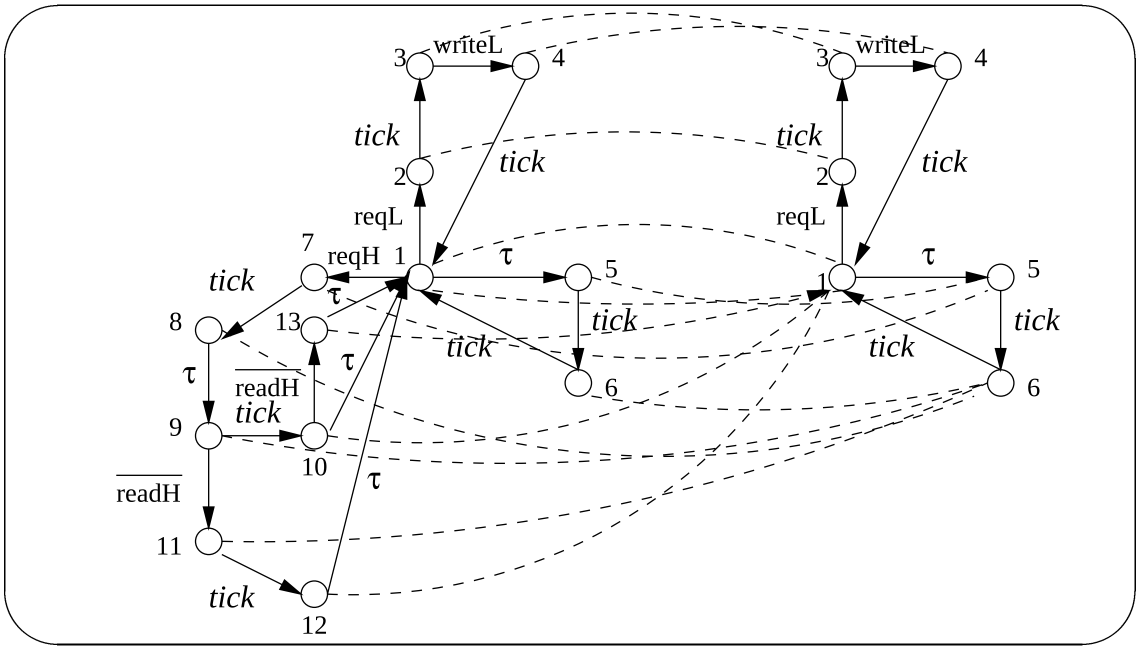


Figure 5.6: A graphical explanation of $tBNDC$ membership of $Manager$.

Now, high users are forced to wait exactly one time unit in the critical section, and then they must leave it.

It is worthwhile noticing that $Manager$ does not belong to $tSBSNNI$. In fact, there is a process E' in $Der(E)$ s.t. $E' \backslash H \xrightarrow{tick}$ and $E'/H \not\xrightarrow{tick}$. The process E' is the following:

$$(tick.\iota(\overline{stop}.Timer) \| (\iota(stop.C) + \iota(\overline{read_H}.\iota(stop.C)))) \backslash \{go, stop\}.$$

Remark 5.1 *We can directly prove that $Manager \in tBNDC$. The idea is to directly build a timed weak bisimulation that contains $(Manager \| \Pi \backslash H, Manager \backslash H)$, for every high user $\Pi \in \mathcal{E}_H^t$. This is one of the advantages of considering timed (weak) bisimulation as equivalence relation between our systems. In fact, this kind of equivalence has a nice proof technique:*

> *We can prove that two processes are equivalent by simply providing a bisimulation relation that contains them.*

In our specific case, this is very interesting since at the same time we are able to prove an infinite number of equivalences (like BNDC membership requires)!

Consider first the Figure 5.6. The dashed lines represent couples of a function f from derivatives of $Manager$ to derivatives of $Manager \backslash H$. For sake of simplicity, in Figure 5.6 we use numbers to represent derivatives instead of terms. Hence we have the following relation:

$$\mathcal{R} = \{(E' \| \Pi \backslash H, E'') \mid (E', E'') \in f, \Pi \in \mathcal{E}_H^t, E' \in Der(Manager)\}.$$

The proof that $\mathcal{R}$ is a timed weak bisimulation follows by inspection of the possible cases. As example, we show that if $(9 \| \Pi \backslash H, 6) \in \mathcal{R}$ we have the following possibilities to consider:

- *if $9 \| \Pi \backslash H \xrightarrow{tick} 10 \| \Pi' \backslash H$, hence $\Pi \xrightarrow{tick} \Pi'$ and $\Pi \not\xrightarrow{\overline{read_H}}$. In this case $6 \xrightarrow{tick} 1$ and $(10 \| \Pi' \backslash H, 1) \in \mathcal{R}$.*

- *if $9 \| \Pi \backslash H \xrightarrow{\tau} 11 \| \Pi' \backslash H$, then $\Pi \xrightarrow{read_H} \Pi'$, but since $(11 \| \Pi' \backslash H, 6) \in \mathcal{R}$ we have completed.*

- *if $9 \| \Pi \backslash H \xrightarrow{\tau} 9 \| \Pi' \backslash H$, then $\Pi \xrightarrow{\tau} \Pi'$ but also in this case we have $(9 \| \Pi' \backslash H, 6) \in \mathcal{R}$.*

- *if $6 \xrightarrow{tick} 1$, we have two possibilities, if $\Pi \xrightarrow{read_H} \Pi'$ then $9 \| \Pi \backslash H \xrightarrow{\tau} 11 \| \Pi' \backslash H$. Now, since Π' must be weakly alive, we have $11 \| \Pi' \backslash H \overset{\tau}{\Longrightarrow} 11 \| \Pi'_1 \xrightarrow{tick} 12 \| \Pi'' \backslash H$, and $(12 \| \Pi'' \backslash H, 1) \in \mathcal{R}$. The other possibility is that $\Pi \not\xrightarrow{read_H}$ and also in this case since Π must be weakly alive, we have $9 \| \Pi \backslash H \overset{\tau}{\Longrightarrow} 9 \| \Pi' \xrightarrow{tick} 10 \| \Pi'' \backslash H$ and $(10 \| \Pi'' \backslash H, 1) \in \mathcal{R}$.*

5.7 Conclusions and future research

In this chapter we have studied a notion of information flow security property, namely Non Deducibility on Composition, as defined by Focardi an Gorrieri (see [33, 34, 35]).

We have proposed a methodology for studying NDC like properties. This kind of properties may be defined by using several equivalences. Likely, characteristic formulas also may be defined for several equivalence relations over processes, e.g. ready-simulation and divergence bisimulation (see [4, 91, 92]). Hence, our approach can also be applied in the analysis of NDC properties defined by exploiting the aforementioned equivalences.

This method is based on the combination of partial model checking (a modular approach for model checking) and theorem proving. In the last years, model checking and theorem proving, have been used separately and successfully in the analysis of security protocols ([32, 37, 59, 78, 84]), in particular for detecting flaws in authentication protocols.

One of the main advantages of composing partial model checking and theorem proving is the possibility of analyzing the behaviour of a system against all the possible environments it can interact with. So *deadlock freedom* can be checked w.r.t. all *high* level users (these properties can not be analyzed with the tool presented in [35]). It is worthwhile noticing that this method can ensure the *non interference* property only w.r.t. the characteristics of the system that are modeled.

We have extended the theory for dealing with a quantitative notion of time. We have shown the flexibility of our methodology for the analysis of BNDC like properties, by showing that partial evaluation analysis can be fruitfully exploited to work in this framework. The time modeling is very simple, more challenging models may be studied. For example, a possible extension should be the integration of time and probabilities in the same framework for dealing also with a notion probabilistic non interference (see [38]). A possible language for description of systems could be the one presented in [9].

We believe that partial evaluation techniques provide a general framework where NDC like properties can be decided for a generic specification language.

Chapter 6

Analysis of cryptographic protocols

In this chapter we propose a methodology for the formal analysis of cryptographic protocols. We define a language for the description of protocols and a logical language for expressing security properties of protocols. An unspecified component is used to model a hostile environment in which the protocol runs. Then we design a suitable partial evaluation function for our language. We show a relationship among the verification problems of different security properties. Hence the theoretical aspects of an implementation of the theory proposed are given. The chapter concludes with a discussion about the use of compositional analysis concepts in the formal verification of security properties.

6.1 Introduction

The continuous growing of the amount of security-sensitive information, which flows in computer networks, has caused a lot of interest in research in formal methods for the definition and the analysis of security properties which systems must ensure.

A typical example of security properties is the necessity that only legitimate users can access some kind of information, or a particular service, or else that parties in a communication get assurance about the identity of their correspondents. Furthermore, computer networks may consist of thousand of geographically distributed computers, and the communication between a two of them may involve the exploiting of communication features of many others in the network. This raises the necessity of establishing secure communication channels, in such a way that the secrecy (or confidentiality) of exchanged data is kept during the steps of the communication.

Cryptographic systems ([83]) are used to try to solve these problems in communication protocols. These protocols are named cryptographic. Through encryption, messages are exchanged between parties over a possibly insecure medium in such a way that it should be possible to retrieve the actual meaning of the message only from users who know a certain piece of information, i.e. a key (this ideal situation is referred to as perfect encryption assumption). Unfortunately, cryptography can represent only a foundational tool for ensuring security properties, but alone it is not sufficient, as proved by many flaws found in cryptographic protocols (see [2, 35, 61, 72]), even by assuming the *perfect encryption*. Cryptographic protocols, even though at conceptual level involve only few communications between parties, are recognized to be prone to errors.

In the last years, various techniques for finding flaws in such protocols have been developed (see [32, 35, 59, 61, 63, 72, 82]). Some of them are essentially based on the analysis of finite state systems, and typically can ensure error freedom only for a finite amount of the behaviour of systems.

Another approach is based on proof techniques for authentication logics (see [2, 48, 78]) or for process algebras (see [1, 14]).

Our approach is novel in this area, and is based on partial evaluation techniques derived from the idea of partial model checking and compositional analysis. We have already shown the applicability of this approach for ensuring information flow security properties in the previous chapter. Here, we transpose these ideas for the verification of cryptographic protocols.

The intuitive idea is the following: verifying that a system S in conjunction with a generic process X enjoys a property expressed by a formula F of a logical language L, is equivalent to verify that X by itself satisfies a particular formula $F//S$, where the requirements are changed in order to respect the evaluation of the behaviour of S:

$$S\|X \models_L F \text{ iff } X \models_L F//S$$

We can exploit this idea by supposing that the process X can be seen as an intruder that tries to discover some information, which should remain enclosed in the system S. The unleashing of information may be expressed by the logical formula F.

The idea of modeling an intruder that acts in composition with the system to be analyzed is not new. But generally, only a particular intruder is considered, and his capabilities are somewhat questionable. We follow the natural idea to consider the intruder as an unspecified component. Then, the partial evaluation techniques seem to be a correct technical tool for analyzing this kind of situations, as we have advocated in chapters 3 and 5. In this way we have formally, and exactly which properties this possible intruder must have in order to perform a successful attack on the protocol (in our formal model).

6.2 An operational language for the description of protocols

In this section we present the language for the description of protocols. First of all we introduce the notion of types, and typed messages. The model consists of a set of sequential agents that can communicate by exchanging messages, in a synchronous way.

6.2.1 Types and typed messages

Let us suppose to have a finite set $\{\mathsf{T}^1,\ldots,\mathsf{T}^n\}$ of collections of values. Every set $\mathsf{T}^i, i \in \{1,\ldots,n\}$ can be partitioned in two disjoint sets, the basic values BT^i and the random values RT^i, in such a way that the former set is finite and the latter is infinite (but countable). Every collection T^i has associated a $symbol$ T^i that represents the set of values of that collection, i.e. its type. By using a finite set of function symbols $\{F^k, G^{k'}, \ldots\}$ and a special symbol $\times$ (product), we build the set of message types inductively as follows:

- every basic type $T^i, i \in \{1, \ldots, n\}$ is a message type,
- if T, T' are message types then $T \times T'$ is a message type,
- if $T_1, \ldots, T_j$ are message types and F^j is a function symbol of arity j then $F^j(T_1, \ldots, T_j)$ is a message type,
- nothing else is a message type.

Then we can define the set of typed messages inductively as follows:

- if x is a message variable and T is a message type then $x : T$ is a typed message of type T,
- if $m_i \in \mathsf{T}^i$ then $m_i : T^i$ is a typed message of type T^i,
- if $m_1 : T_1$ and $m_2 : T_2$ are typed messages then $(m_1, m_2) : T_1 \times T_2$ is a typed message of type $T_1 \times T_2$,
- if $m_1 : T_1, \ldots, m_n : T_n$ are typed messages and F^j is a function symbol of arity j then $F^j(m_1, \ldots, m_j) : F^j(T_1, \ldots, T_j)$ is a typed message of type $F^j(T_1, \ldots, T_j)$,
- nothing else is a typed message.

A typed message $m : T$ without variables is called a $pure$ typed message. Equality between typed message means syntactic equality. The set of typed messages of a type T, Msgs(T), is defined as the set $\{m : T | m : T \text{ is a pure typed message}\}$. We give below the definition of the set of typed submessages of a typed message $m : T$.

Definition 6.1 *Let $m : T$ be a pure typed message; the typed submessages of $m : T$ are $SubM(m : T)$, where $SubM$ is defined as:*

$$SubM(m : T^i) = \{m : T^i\}$$

$$SubM((m, m') : T \times T') = \{(m, m') : T \times T'\} \cup SubM(m : T) \cup SubM(m' : T')$$

$$SubM(F(m_1, \ldots, m_n) : F(T_1, \ldots, T_n)) = \{F(m_1, \ldots, m_n) : F(T_1, \ldots, T_n)\} \cup \bigcup_{i \in \{1, \ldots, n\}} SubM(m_i : T_i)$$

A message $m : T$ is *initial* iff $m : T$ is pure and every submessage $m' : T^i$ of $m : T$ of basic type T^i is member of BT^i. A function η from variables to pure messages or types is called a substitution. We write $(m : T)[\eta]$ for the message obtained replacing every variable x with $\eta(x)$.

Before introducing the syntax of composed systems let us see the inference system of sequential agents, that permits them to deduce new messages by starting from the set of messages that they have produced or received. This system consists of a set of inference schemata. An inference schema is a couple $IS = \langle (m_1 : T_1, \ldots, m_n : T_n), m_0 : T_0 \rangle$, usually written as:

$$\frac{m_1 : T_1, \ldots, m_n : T_n}{m_0 : T_0}$$

where $m_1 : T_1, \ldots, m_n : T_n$ is a set of premises (possibly empty) and $m_0 : T_0$ is the conclusion. We say that a pure typed message $m : T$ is inferred from a set of pure typed messages $m'_1 : T_1, \ldots, m'_n : T_n$ by using $IS = \langle (m_1 : T_1, \ldots, m_n : T_n), m_0 : T_0 \rangle$ (for short $m'_1 : T_1, \ldots, m'_n : T_n \vdash_{IS} m : T$), if a substitution η exists s.t. for every $1 \leq j \leq n$ $(m_j : T_j)[\eta] = m'_j : T_j$ and $(m_0 : T_0)[\eta] = m : T$. An axiom schema is a rule schema with no premises. As notation, we write $\langle\langle m_i : T_i \rangle\rangle_{i \in I}, I = \{1, \ldots, n\}$ for a sequence $m_1 : T_1 \ldots m_n : T_n$.

A *proof* for a typed message $m : T$ is a finite tree, rooted in $m : T$, whose leaves are instances of axiom schema and each node is built from its descendents by applying a rule schema. We write $\phi \vdash m : T$ if a proof exists for a typed message $m : T$, where every message $m' : T' \in \phi$ appears in the conclusion of an axiom. Given an inference system, we define a deduction function $\mathcal{D}(\phi) = \{m : T | \phi' \subseteq \phi \wedge \phi' \vdash m : T\}$.

6.2.2 Syntax

We briefly introduce the syntax of systems. A system (term) is generated by the following grammar:

Composed systems:

$$S \quad ::= \quad S \backslash L \mid S \backslash\backslash L \mid S_1 \| S_2 \mid (A)_\phi$$

Sequential agents:

$$\begin{array}{lll} A & ::= & Nil \mid a.A \mid A_1 + A_2 \mid [m : T = m' : T']A_1; A_2 \mid \\ & & [\langle\langle m_i : T_i \rangle\rangle_{i \in I} \vdash_{IS} x : T]A_1; A_2 \end{array}$$

$$a \quad ::= \quad c!m : T \mid c?(x) : T \mid \tau \mid \tau_{c,m:T} \mid \chi^x_{c,m:T} \mid gen^{x,(i,j)}_T$$

where $m : T, m' : T'$ are typed messages, C is a finite set of channels with $c \in C$, x is a message variable, ϕ is a finite set of pure typed messages, L is a subset of C and $i, j \in \mathbb{N}$ (the set of natural numbers).

The inference construct $[\langle\langle m_i : T_i\rangle\rangle_{i\in I} \vdash_{IS} x : T]A_1; A_2$ acts as a binding for the variable x in A_1, whereas the variable x must not appear in A_2. The prefix constructs $c?(x) : T.A$, $\chi^x_{c,m:T}.A$, $gen^{x,(i,j)}_T.A$ are bindings for the variable x in A. Hereafter we consider only sequential agents where the binding constructs bind the same variable almost once. A sequential agent is said to be closed if every message variable is bound. In the sequel we consider only closed agents.

We make some assumptions on the capacity of sequential agents to guess random values. Random generated messages (nonces) are used to witness the freshness of messages during executions of the protocols (runs). To model the characteristics of these messages, we have assumed that for every basic type T^i there is a subset of messages of this kind, i.e. RT^i. A particular action $gen^{x,(i,j)}_T$ permits to *guess* a random value of a basic type T. Random messages of composed types can be built by using basic random values as subcomponents. Since they must be guessed randomly, it should be impossible that two different agents can guess the same values. Formally, this is achieved by partitioning the set of basic values in as many sets as the number of sequential agents that compose a system. For every basic type T^k there is a particular function $\mathcal{R}^{T^k} : \mathbb{N} \times \mathbb{N} \mapsto \mathsf{RT}^k$ which gives the values that are *guessed* by agents. For every one of these functions, if $i \neq i'$or $j \neq j'$ then $\mathcal{R}^{T^k}(i,j) \neq \mathcal{R}^{T^k}(i',j')$. A sequential agent is correct if the set of actions $gen^{x,(i,j)}_T$, which he performs, has always the same j index and always a different i index.

A compound system is correct if its sequential agents in their starting configuration contain only messages that are in their initial knowledge, which must consist only of initial messages. We implicitly assume that the receiver of a message is able to recognize the structure of received messages, even if it is not able to *understand* the meaning of such messages.

The set Act of actions, which can be performed by a compound system, is defined as: $Act = \{c?m : T \mid c \in C, m : T \in Msgs(T)\} \cup \{c!m : T \mid c \in C, m : T \in Msgs(T)\} \cup \{\chi_{c,m:T} \mid c \in C, m : T \in Msgs(T)\} \cup \{\tau_{c,m:T} \mid m : T \in Msgs(T)\}\} \cup \{\tau_{g:T^i} \mid g \in \mathsf{RT}^i\} \cup \{\tau\}$. Below we give the definition of the function *channel*, that given an action returns a channel ($void$ if the channel is not specified), and *message* that given an action returns its message:

	$c!m : T$	$c?m : T$	τ	$\tau_{c,m:T}$	$\chi_{c,m:T}$	$\tau_{g:T}$
channel	c	c	$void$	c	c	$void$
message	$\{m : T\}$	$\{m : T\}$	$\emptyset$	$\{m : T\}$	$\{m : T\}$	$\{g : T\}$

To every sequential agent in a composed system, it is possible to assign a unique identifier, i.e. the path from the root to the sequential agent term in the parsing tree of the compound term. As notation we write $S \overset{\gamma}{\mapsto} S'$ if γ is a finite sequence of actions $\gamma_i, 1 \leq i \leq n$ s.t. $S = S_0 \xrightarrow{\gamma_1} \ldots \xrightarrow{\gamma_n} S_n = S'$. Given a sequence of actions we use the function $msgs$, defined as $msgs(\beta\gamma) = message(\beta) \cup msgs(\gamma)$ and $msgs(\epsilon) = \emptyset$, to get the set of communicated messages. Given a sequence of transitions $S \overset{\gamma}{\mapsto} S'$ of a compound term S, $(S \overset{\gamma}{\mapsto} S') \downarrow_X$ [1] is the sequence of actions of the agent identified from

[1]Here we use this notation with a different meaning of the one in chapters 1,2,3.

X in S, that have contributed to the transitions of the whole system. The set of channels, where an agent A possibly performs a communication, is defined as $Sort(A)$ (this set can be built syntactically).

6.2.3 Operational semantics

As usual we give semantics to our language through Labelled Transition Systems. The semantics of compound terms is given by the least set of action relations induced by the rules of figure 6.1.

Informally, the semantics of sequential agents is the following: Nil is the process that can do nothing; $a.A$ is the process that can perform an action according to the particular construct a and then behaves as A (in particular the action $\chi^x_{c,m:T}$ allows to listen the communications internal to other subcomponents of the system). Moreover the rule $(!)$ plays a central role in our calculus. It asserts that an agent can send a message only if he can deduce it by his knowledge. $A_1 + A_2$ is the process that nondeterministically chooses to behave as A_1 or A_2; $[m : T = m' : T']A_1; A_2$ is the matching construct: it permits to check the equality between typed messages.

The deduction construct $\langle\langle m_i : T_i \rangle\rangle_{i \in I} \vdash_{IS} x : T$ permits to deduce new messages by applying a particular inference schema. By using this construct a finite number of times, an agent can build the proof of every message in $\mathcal{D}(\phi)$. Typically, it may be used to decrypt messages by applying a rule such as 6 in section 6.4.

The compound system $S \| S_1$ performs an action a if one of the two subcomponents performs this action, and a synchronization action ($\tau_{c,m:T}$) if both the subcomponents perform complementary actions, i.e. send ($c!m : T$) or receive ($c?m : T$). It is worthwhile noticing that in contrast to CCS process algebra our synchronization actions carry information on the message exchanged. In this way we can model eaves–dropping. To model a synchronous communication over a set of channels L, the $\backslash L$ operator must be used in conjunction with $\|$, e.g. $(S \| S_1)\backslash L$. The latter process cannot perform actions whose channel is in the set L, except for synchronizations. The hiding construct $S \backslash\backslash L$ permits to hide the message exchanged during a synchronization on a channel in L. This operator can be used to model a subsystem where the communications are safe, and also it may be applied in a compositional design methodology. For notational convenience, we use $S \|_L X$ for $(S \| X)\backslash L$; besides we consider $S = Nil$ when $\forall a \in Act, S \not\xrightarrow{a}$.

6.3 A logical language for the description of protocol properties

We present a logical language ($\mathcal{L}_K$) for the specification of the functional and security properties of a compound system. We have extended a normal multimodal logic with operators that permit to know whether a message can be derived by an agent after a sequence γ performed by the whole system, starting from an initial knowledge.

$$(?)\,\frac{m:T \in Msgs(T)}{(c?(x):T.A)_\phi \xrightarrow{c?m:T} (A[m/x])_{\phi\cup\{m:T\}}}$$

$$(gen)\,\frac{g=\mathcal{R}^T(i,j)}{(gen_T^{x,(i,j)}.A)_\phi \xrightarrow{\tau_{g:T}} (A[g/x])_{\phi\cup\{g:T\}}}$$

$$(\chi)\,\frac{}{(\chi^x_{c,m:T}.A)_\phi \xrightarrow{\chi_{c,m:T}} (A[m/x])_{\phi\cup\{m:T\}}}$$

$$([]_1)\,\frac{m:T \neq m':T' \qquad (A_2)_\phi \xrightarrow{a} (A_2')_{\phi'}}{([m:T=m':T']A_1;A_2)_\phi \xrightarrow{a} (A_2')_{\phi'}} \qquad (!)\,\frac{m:T\in D(\phi)}{(c!m:T.A)_\phi \xrightarrow{c!m:T} (A)_\phi}$$

$$([]_2)\,\frac{m:T = m':T' \quad (A_1)_\phi \xrightarrow{a} (A_1')_{\phi'}}{([m:T=m':T']A_1;A_2)_\phi \xrightarrow{a} (A_1')_{\phi'}} \qquad (+_1)\,\frac{(A_1)_\phi \xrightarrow{a} (A_1')_{\phi'}}{(A_1+A_2)_\phi \xrightarrow{a} (A_1')_{\phi'}}$$

$$(\mathcal{D}_1)\,\frac{\langle\langle m_i:T_i\rangle\rangle_{i\in I}\vdash_{IS} m:T \;\; (A_1[m/x])_{\phi\cup\{m:T\}} \xrightarrow{a} (A_1')_{\phi'}}{([\langle\langle m_i:T_i\rangle\rangle_{i\in I}\vdash_{IS} x:T]A_1;A_2)_\phi \xrightarrow{a} (A_1')_{\phi'}}$$

$$(\mathcal{D}_2)\,\frac{\nexists(m:T)\langle\langle m_i:T_i\rangle\rangle_{i\in I}\vdash_{IS} m:T \;\; (A_2)_\phi \xrightarrow{a} (A_2')_{\phi'}}{([\langle\langle m_i:T_i\rangle\rangle_{i\in I}\vdash_{IS} x:T]A_1;A_2)_\phi \xrightarrow{a} (A_2')_{\phi'}}$$

$$(\backslash L_1)\,\frac{S \xrightarrow{c!m:T} S' \quad c\notin L}{S\backslash L \xrightarrow{c!m:T} S'\backslash L} \qquad (\backslash L_2)\,\frac{S \xrightarrow{c?m:T} S' \quad c\notin L}{S\backslash L \xrightarrow{c?m:T} S'\backslash L}$$

$$(\backslash L_3)\,\frac{S \xrightarrow{\tau_{c,m:T}} S'}{S\backslash L \xrightarrow{\tau_{c,m:T}} S'\backslash L} \qquad (\backslash L_4)\,\frac{S \xrightarrow{\tau} S'}{S\backslash L \xrightarrow{\tau} S'\backslash L}$$

$$(\|_1)\,\frac{S \xrightarrow{a} S'}{S\|S_1 \xrightarrow{a} S'\|S_1} \qquad (\|_2)\,\frac{S \xrightarrow{c?m:T} S' \quad S_1 \xrightarrow{c!m:T} S_1'}{S\|S_1 \xrightarrow{\tau_{c,m:T}} S'\|S_1'}$$

$$(\backslash\backslash L_1)\,\frac{S \xrightarrow{a} S' \quad channel(a)\notin L}{S\backslash\backslash L \xrightarrow{a} S'\backslash\backslash L} \qquad (\tau_1)\,\frac{S \xrightarrow{\tau_{c,m:T}} S' \quad c\in L}{S\backslash\backslash L \xrightarrow{\tau} S'\backslash\backslash L}$$

$$(\chi_1)\,\frac{S \xrightarrow{\tau_{c,m:T}} S' \quad S_1 \xrightarrow{\chi_{c,m:T}} S_1'}{S\|S_1 \xrightarrow{\tau_{c,m:T}} S'\|S_1'}$$

Figure 6.1: Operational semantics of the language, where there are symmetric rules for $+_1, \|_1, \|_2$ and χ_1. In the rules $\mathcal{D}_1$ and $\mathcal{D}_2$ messages $\langle\langle m_i : T_i\rangle\rangle_{i\in I}$ must be in ϕ.

The syntax of the logical language $\mathcal{L}_K$ is defined by the following grammar:

$$F ::= \mathbf{T} \mid \mathbf{F} \mid \langle a\rangle F \mid [a]F \mid \wedge_{i\in I} F_i \mid \vee_{i\in I} F_i \mid m{:}T \in K^{\phi}_{X,\gamma} \mid \exists\gamma : (m{:}T) \in K^{\phi}_{X,\gamma}$$

where $m : T$ is a pure typed message, X is an agent identifier, I is an index set and ϕ a finite set of pure typed messages. Informally, the $\langle a\rangle F$ modality expresses the *possibility* to perform an action a and then satisfy F. The $[a]F$ modality expresses the *necessity* that after performing an action a the system satisfies F.

A system S satisfies a formula $m : T \in K^{\phi}_{X,\gamma}$ if S can perform a sequence γ of actions and an agent of S, identified by X, can deduce $m : T$ starting from the set of messages ϕ and the messages he has known during the performing of the sequence γ. This formula plays a central role in the analysis of authentication protocols, that are often based on the sharing, between two parties, of a secret, i.e. a particular message that is assumed no one else knows. Hence, pieces of information are used to witness the identity of agents and the eventual disclosure of particular information can have dangerous consequences. If we want to know if a sequence γ exists s.t. an agent X_ϕ can deduce $m : T$, we can express this fact using the formula $\exists\gamma : (m : T) \in K^{\phi}_{X,\gamma}$. Hence, in the sequel we are mainly interested in the study of properties that can be formulated in the following way:

> Is there an agent (intruder) that in communication with the agents of the system, can retrieve a secret that should be shared only among some agents of S?

It can be formally restated in our model in with the following statement, where $m : T$ represent the secret (message).

$$S \| X_\phi \models \exists\gamma : (m : T) \in K^{\phi}_{X,\gamma}$$

Actually, in general we wonder if there exists an intruder with a particular initial knowledge.

The language without $m : T \in K^{\phi}_{X,\gamma}$ and $\exists\gamma : (m : T) \in K^{\phi}_{X,\gamma}$ (knowledge operators) is called $\mathcal{L}$.

6.3.1 Semantics

We suppose to have a deduction function D: $\mathcal{P}(Msg) \mapsto \mathcal{P}(Msg)$, which has to enjoy the properties listed below:

Assumption 6.1 *For every type T the set of messages in D$(\phi) \cap Msgs(T)$ is finite and constructible*[2]*, when ϕ is a finite set. We require this assumption since we want to be able to perform an automatic analysis.*

[2]Here, we mean that there is a procedure that terminates in a finite amount of time and returns an explicit enumeration of $\mathcal{D}(\phi) \cap Msgs(T)$.

Assumption 6.2 *If IS is an inference schema and δ a bijection between random values then:* $m_1 : T_1 \ldots m_n : T_n \vdash_{IS} m : T$ iff $\delta(m_1 : T_1) \ldots \delta(m_n : T_n) \vdash_{IS} \delta(m : T)$. *The idea under this assumption is to avoid deduction systems that are not general and depend on particular random values.*

Assumption 6.3 *If $m : T$ is a typed message and $g : T^i$ a random value, that does not appear as submessage neither in $m : T$ nor in any of the messages in ϕ, then* $m : T \in \mathcal{D}(\phi)$ iff $m : T \in \mathcal{D}(\phi \cup \{g : T^i\})$. *As above.*

Assumption 6.4 *If $m : T$ is a typed message and $m : T \in \mathcal{D}(\phi)$, then every basic value of $m : T$ must be a submessage of some message in ϕ. We want that messages of basic type cannot be forged.*

We can give the semantics of a formula F w.r.t. an LTS associated with a composed system S. We define the semantics of a formula $F \in \mathcal{L}_K$ inductively as follows:

$$\begin{array}{lll}
\text{For every } S \text{ we have } S \models \mathbf{T} & & \\
\text{For no } S \text{ we have } S \models \mathbf{F} & & \\
S \models \wedge_{i \in I} F_i & \text{iff} & \forall i \in I : S \models F_i \\
S \models \vee_{i \in I} F_i & \text{iff} & \exists i \in I : S \models F_i \\
S \models \langle a \rangle F & \text{iff} & \exists S' : S \xrightarrow{a} S' \text{ and } S' \models F \\
S \models [a] F & \text{iff} & \forall S' : S \xrightarrow{a} S' \text{ implies } S' \models F \\
S \models m : T \in K^{\phi}_{X,\gamma} & \text{iff} & \exists S' : (S \overset{\gamma}{\mapsto} S') \downarrow_X = \tilde{\gamma} \text{ and} \\
 & & m : T \in \mathcal{D}(\phi \cup msgs(\tilde{\gamma})) \\
S \models \exists \gamma : (m : T) \in K^{\phi}_{X,\gamma} & \text{iff} & \exists \gamma : S \models m : T \in K^{\phi}_{X,\gamma}
\end{array}$$

6.4 A suitable deduction function

In this section we show that our requirements on the deduction function are not too restrictive, because the inference system in figure 6.2 satisfies our assumptions and also is similar to the one used by many authors (see [59, 63, 82]). Given a set of messages ϕ then $m : T \in \mathcal{D}(\phi)$ if $m : T$ can be proved by using the following axioms (1) and rules (2-6), where E represents the encryption function, Key is a basic type of keys and $(-1)Key$ is the type of inverse keys, and there are some basic types such as atomic messages and agent identifiers:

The interesting rules are 5 and 6; the former permits the encryption of messages by using a key, and the latter permits one to know the encrypted message when knowing the inverse key. Please note that the system does not allow the deduction of keys (or inverse keys) that are not in ϕ. We can state:

Proposition 6.1 *The above deduction function enjoys assumptions 6.1,6.2,6.3 and 6.4.*

$$(1)\frac{m : T \in \phi}{m : T}$$

$$(2)\frac{m : T_1 \quad m' : T_2}{(m, m') : T_1 \times T_2} \qquad (3)\frac{(m, m') : T_1 \times T_2}{m : T_1} \qquad (4)\frac{(m, m') : T_1 \times T_2}{m' : T_2}$$

$$(5)\frac{m : T_1 \quad k : Key}{E(k, m) : E(Key, T_1)} \qquad (6)\frac{E(k, m) : E(Key, T) \quad k^{-1} : (-1)Key}{m : T}$$

Figure 6.2: A deduction system that enjoys our assumptions.

6.5 A simple protocol

In this section we show a simple example of protocol written in our formalism. First of all we describe it in the notation used in literature as follows:

$$A \mapsto B \quad : \quad \{m\}_{PK(B)}$$

With $A \mapsto B : \ \{m\}_{PK(B)}$ is intended that A sends a message $\{m\}_{PK(B)}$ to B, and B receives this message. The message $\{m\}_{PK(B)}$ is an encryption of a message m with the public key of B. It is assumed that only B knows the private key $PK(B)^{-1}$ (inverse key).

Actually, this notation is misleading, since it represents only a "correct" execution of the protocol. In fact, commonly there is no assurance about the identity of the parties in the communication. Hence, B can perform a receiving action but we can not assume that actually the other party is A. This must be deduced by the contents of messages. So the above protocol can only be used to send a message m to B in such a way that only B can read it. A non trivial example of authentication protocol is shown in figure 6.5.

Hence we model the behaviour of the two parties separately. We use as inference system the one in figure 6.2.

The sender $(A)_{\phi_A}$ is the following, where $\phi_A = \{m : T, PK(B) : Key\}$ is the initial knowledge of A.

$$([m : T \ PK(B) : Key \vdash_5 x : E(Key, T)](c_{AB}!x : E(Key, T).A_1); Nil)$$

The receiver $(B)_{\phi_B}$ with $\phi_B = \{PK(B) : (-1)Key\}$ is the following:

$$c_{AB}?(y) : E(Key, T).([y : E(Key, T) \ PK(B)^{-1} : (-1)Key \vdash_6 z : T]B_1; Nil)$$

where A_1 and B_1 are respectively the continuations of A and B.

The agent A builds the encrypted message and then sends it on the channel c_{AB}. The agent B receives an encrypted message from the channel c_{AB} and then tries to decrypt it by using his inverse (private) key $PK(B)^{-1}$. At the end of a correct run of the protocol the variable z of B must contain m.

It is worthwhile noticing that our description of the protocol is near to a possible implementation of the protocol, and make explicit the steps that the agents must perform. Actually, this positive aspect is common to other approaches based on process algebra theory (see [1, 35, 59, 84]).

6.6 Partial evaluation techniques

In this section we define the partial evaluation techniques tailored for our language. The crux is that, in contrast to previous work shown in this thesis, in this case we have to deal with terms that may have an infinite number of successors (e.g. this is the case for an agent (term) that can receive a typed message $c?(x) : T.S$). Since, we would like to design an automatic method for the analysis of protocols, we should avoid sources of infinite behaviour. In particular, we do not want to consider behaviours of intruders that are not useful for a successful attack of a protocol. This is the case of random generation actions.

Hence, it is convenient to assume a particular behaviour of agents with regard to the generation of random values. In particular, we want that if an agent performs a sequence of actions whose first action is the guessing of a random value $g : T$ then this value will be eventually sent as submessage of some message $m' : T'$ during this sequence and between these two events only guessing actions are performed.

Definition 6.2 *A sequential agent X_ϕ is immediately well behaved* iff *if $X_\phi \overset{\tau_{g_1:T_1}\gamma'}{\longmapsto} X^1_{\phi\cup\{g_1:T_1\}\cup msgs(\gamma')}$ then $\gamma' = \tau_{g_2:T_2}\ldots\tau_{g_n:T_n}c!m' : T\gamma''$ with*
$\phi' = \phi \cup \{g_1 : T_1, \ldots g_n : T_n\}$
$m' : T \in \mathcal{D}(\phi')$
$\{g_1 : T_1, \ldots, g_n : T_n\} \subseteq SubM(m' : T)$.
X_ϕ is well behaved iff *$\forall X'_{\phi'} \in \{X'_{\phi'} | X_\phi \longrightarrow_* X'_{\phi'}\}$ $X'_{\phi'}$ is immediately well behaved, where $\longrightarrow_*$ is the reflexive and transitive closure of the relation $\cup_{a\in Act} \xrightarrow{a}$.*

The following lemma tell us that if we are interested in the analysis of formulas like $\exists\gamma : m \in K^{\phi}_{X,\gamma}$, we can restrict ourselves to consider only this particular kind of sequential agents. This implies that only random values that are sent are influent.

Lemma 6.1 *If there exists a sequential agent X_ϕ s.t. $S\|_L X_\phi \models \exists\gamma : m \in K^{\phi}_{X,\gamma}$ then a well behaved agent X'_ϕ exists s.t. $S\|_L X'_\phi \models \exists\gamma : m \in K^{\phi}_{X',\gamma}$.*

We give the partial evaluation function for $||$ and $\mathcal{L}$ without knowledge operators in figure 6.3, and in figure 6.4 for $\|_L$ and $\exists\gamma : (m : T) \in K^{\phi'}_{X,\gamma}$, where:

$$\begin{array}{rcl} succ(S) & = & \{(c, m' : T, S') | S \overset{c?m':T}{\longrightarrow} S' \text{ and } m' : T \in \mathcal{D}(\phi)\} \\ Rsucc(S) & = & \{(c, m' : T, \langle g_1 : T_1, \ldots, g_n : T_n\rangle, S') | X_\phi \overset{\tau_{g_1:T_1},\ldots,\tau_{g_n:T_n}}{\longmapsto} X'_{\phi'} \wedge \\ & & S \overset{c?m':T}{\longrightarrow} S' \wedge m' : T \in \mathcal{D}(\phi') \wedge \{g_i : T_i\}_{i\in I} \subseteq SubM(m' : T)\} \end{array}$$

$$\begin{array}{rcl}
\mathbf{T}//S & \doteq & \mathbf{T} \\
\mathbf{F}//S & \doteq & \mathbf{F} \\
(\langle A\rangle F)//S & \doteq & \langle a\rangle(F//S) \vee \bigvee_{S \xrightarrow{a} S'} F//S' \qquad (a \neq \tau_{c,m}) \\
(\langle \tau_{c,m:T}\rangle F)//S & \doteq & \bigvee_{S \xrightarrow{c!m:T} S'} \langle c?m:T\rangle(F//S') \vee \bigvee_{S \xrightarrow{c?m:T} S'} \langle c!m:T\rangle(F//S') \\
 & & \bigvee_{S \xrightarrow{\tau_{c,m':T}} S'} \langle \chi_{c,m:T}\rangle F//S' \vee \bigvee_{S \xrightarrow{\tau_{c,m:T}} S'} F//S' \\
\vee_{i\in I} F_i//S & \doteq & \vee_{i\in I}(F_i//S) \\
\wedge_{i\in I} F_i//S & \doteq & \wedge_{i\in I}(F_i//S)
\end{array}$$

Figure 6.3: Partial evaluation function for $||$ and $\mathcal{L}$.

$$\begin{array}{lr}
\exists\gamma : (m{:}T) \in K^{\phi}_{X,\gamma}//S \doteq & \\
\bigvee_{(c,m':T',S')\in succ(S)} \langle c!m' : T'\rangle(\exists\gamma : (m{:}T) \in K^{\phi}_{X,\gamma}//S') & (sending) \vee \\
\bigvee_{(c,m':T',\langle\langle g_i:T_i\rangle\rangle_{i\in I},S')\in Rsucc(S)} \langle\langle \tau_{g_i:T_i}\rangle\rangle_{i\in I}\langle c!m' : T'\rangle & \\
\qquad (\exists\gamma : (m{:}T) \in K^{\phi\cup\langle\langle g_i:T_i\rangle\rangle_{i\in I}}_{X,\gamma}//S') & (guessing) \vee \\
\bigvee_{S \xrightarrow{c!m':T'} S'} \langle c?m' : T'\rangle(\exists\gamma : (m{:}T) \in K^{\phi\cup\{m':T'\}}_{X,\gamma}//S') & (receiving) \vee \\
\bigvee_{S \xrightarrow{\tau_{c,m'}} S'} \langle \chi_{c,m':T'}\rangle(\exists\gamma : (m{:}T) \in K^{\phi\cup\{m':T'\}}_{X,\gamma}//S') & (eaves-dropping) \vee \\
\bigvee_{S \xrightarrow{a} S'} \exists\gamma : (m{:}T) \in K^{\phi}_{\gamma}//S' & (idling) \vee \\
m : T \in K^{\phi}_{X,\epsilon}//S & (trivial)
\end{array}$$

$$m : T \in K^{\phi}_{X,\epsilon}//S = \exists\gamma : (m{:}T) \in K^{\phi}_{X,\gamma}//Nil = \begin{cases} \mathbf{T} & m : T \in \mathcal{D}(\phi) \\ \mathbf{F} & m : T \notin \mathcal{D}(\phi) \end{cases}$$

Figure 6.4: Partial evaluation function for $\|_L$ and $\exists\gamma : (m : T) \in K^{\phi}_{X,\gamma}$.

The set $succ(S)$ represents the sending actions (and relative successors of S) that can be performed by the intruder. The set $Rsucc(S)$ represents the sequences of guessing of random values followed by a sending of a message that can be performed by the intruder. By observing the compositional analysis proposed in [5, 55] it can be noted that it is somewhat semantic driven. Analogously, our partial evaluation can be derived by inspection of the operational semantics of the language. Between brackets we have put in evidence the corresponding behaviour of the intruder.

We will concentrate on the reduction for $\exists\gamma : (m : T) \in K^{\phi'}_{X,\gamma}$ since it is the most involved in analysis of security properties, while for $\mathcal{L}$ it is a generalization of the ideas of [5]. For the sake of simplicity, we avoid the problem of considering always a different index of the gen actions in the translated formulas (this problem can be easily solved by global counters). It is worthwhile noticing that $succ(S)$ is a finite set (by assumption 6.1).

Given a bijection σ between basic random values, that respects the type (i.e. if $g : T^i$

then $\sigma(g) : T^i$), this bijection naturally extends to a bijection between typed messages, sequential agents and composed systems in a straightforward way. If there is a σ bijection between two composed systems then we write $S \equiv_\sigma S'$.

Lemma 6.2 *If $S \equiv_\sigma S'$ then $S \xrightarrow{a} S_1 \Longrightarrow S' \xrightarrow{\sigma(a)} S'_1$ and $S' \equiv_\sigma S'_1$.*

The above lemma can be used to prove that $S \overset{\gamma}{\mapsto} S_1$ and $S \equiv_\sigma S'$ then $S' \overset{\sigma(\gamma)}{\mapsto} S'_1$.

The next proposition states the correctness of the partial reduction, where we assume that X is a well behaved process.

Proposition 6.2 *Given a system S, with $Sort(S) \cup Sort(X) \subseteq L$, a finite set of typed messages ϕ and an initial message $m : T$ then:*

$$S\|_L X_\phi \models \exists\gamma : (m : T) \in K^\phi_{X,\gamma} \text{ iff } X_\phi \models \exists\gamma : (m : T) \in K^\phi_{X,\gamma}//S.$$

Unfortunately, the formula $F = \exists\gamma : (m_1 : T') \in K^\phi_{X,\gamma}//S$ presents various infinitary disjunctions, which are due to the analysis of the generation of random values by the agent X. By our assumptions on the deduction function it is not fundamental which sequence of generation actions is performed, the essential thing is the correct kind of types that are generated. So we can prove:

Lemma 6.3 *If $m : T$ is an initial message, σ a bijection between $\{g_i : T_i\}_{i\in I}$ and $\{g'_i : T'_i\}_{i\in I}$ with $S'\|X' \equiv_\sigma S\|X$ and $SubM(\phi) \cap \{g_i : T_i\}_{i\in I} = \emptyset = SubM(\phi) \cap \{g'_i : T'_i\}_{i\in I}$ then:*

$$\begin{gathered} S\|_L X_{\phi\cup\{g_i:T_i\}_{i\in I}} \models \exists\gamma : (m : T) \in K^{\phi\cup\{g_i:T_i\}_{i\in I}}_{X,\gamma} \\ \text{iff} \\ S'\|_L X'_{\phi\cup\{g'_i:T'_i\}_{i\in I}} \models \exists\gamma : (m : T) \in K^{\phi\cup\{g'_i:T'_i\}_{i\in I}}_{X',\gamma}. \end{gathered}$$

Now we can give a translation from this formula to another one, s.t. the satisfiability is preserved, i.e. F is satisfiable iff $\tilde{F}$ is satisfiable. This translated formula $\tilde{F}$ presents only finitary disjunctions. This translation can be performed during the generation of F and keeps unchanged the finitary part of the formula. Let us consider the infinitary part of the formula to be reduced (if it is finitary then it is done), so let F be $\vee_{t\in Rsusc(S)} F_t$, where F_t with $t = (c, m' : T', \langle\langle g_i : T_i\rangle\rangle_{i\in I}, S')$ is:

$$\langle\langle g_i : T_i\rangle\rangle_{i\in I}\langle c!m' : T'\rangle(\exists\gamma : (m : T) \in K^{\phi\cup\langle\langle g_i:T_i\rangle\rangle_{i\in I}}_{X,\gamma}//S')$$

We can define an equivalence relation over tuples of $Rsucc(S)$ as below:

$$\begin{array}{ccc} (c, m : T, \langle\langle g_i : T_i\rangle\rangle_{i\in I}, S') & \equiv^e & (c', m' : T', \langle\langle g'_i : T'_i\rangle\rangle_{i\in I'}, S'_1) \\ & \text{iff} & \\ & c = c' \quad T = T' \quad S' \equiv_\sigma S'_1 & \\ \{\!|\langle\langle g_i : T_i\rangle\rangle_{i\in I}|\!\} & = & \{\!|\langle\langle g'_i : T'_i\rangle\rangle_{i\in I'}|\!\} \end{array}$$

where σ is a bijection between $\langle\langle g_i : T_i\rangle\rangle_{i\in I}$ and $\langle\langle g'_i : T'_i\rangle\rangle_{i\in I'}$ and $\{|\langle\langle g_i : T_i\rangle\rangle_{i\in I}|\}$ is the multiset of types which are present in the sequence. Now we can consider the quotient of $Rsucc(S)$ w.r.t. this equivalence relation. The key point is that $Rsucc(S)/\overset{e}{\equiv}$ is finite. In fact, up to renaming, only a finite number of successors of S can be chosen, since only a finite number of sequential agents of kind $c?(x) : T.A$ are present in S and S_1. Moreover the set of channels is finite and for a given type T only a finite number of different multisets are compatible. So we can choose for every class in $Rsucc(S)_{/\equiv^e}$ only a member, let M be the set of such tuples, hence we can define $\tilde{F}$ as $\vee_{t\in M}\tilde{F}_t$.

Lemma 6.4 *Under the hypothesis of proposition 6.2 we have:*

$$\exists X_\phi \models \exists\gamma : m_1 : T' \in K^{\phi}_{X,\gamma}//S \text{ iff } \exists Y_\phi \models \exists\gamma : m_1 : \tilde{T'} \in K^{\phi}_{Y,\gamma}//S.$$

We have reduced the verification of the existence of an agent X_ϕ s.t. $S\|_L X_\phi \models \exists\gamma : (m : T) \in K^{\phi}_{X,\gamma}$ to a satisfiability problem in a sublogic of $\mathcal{L}$. The main result of this chapter is the following:

Theorem 6.1 *Given a system S, with $Sort(S) \subseteq L$, a finite set of typed messages ϕ and an initial message $m : T$ then is decidable if $\exists X_\phi$ with $Sort(X) \subseteq L$ s.t. $S\|_L X_\phi \models \exists\gamma : (m : T) \in K^{\phi}_{X,\gamma}$.*

Moreover, we can build an agent (i.e. intruder) for a satisfiable formula $\tilde{F}$.

6.7 Authentication properties

The theory we have presented in the previous sections deals in particular with so called *secrecy* properties, i.e. that certain pieces of information remain enclosed in a particular context. Among other interesting properties are the authentication ones. The definition of authentication used in [59] can be restated as follows:

> Whenever a Receiver B completes a run of the protocol, apparently with Sender A, then A has recently been running a protocol, apparently with B.

We define two distinct actions *start*, *finish* to model the starting of Sender and the termination of Receiver respectively: when Sender starts it issues the action *start* and when Receiver terminates it issues the action *finish*. It is assumed that such actions cannot be performed by others than Sender and Receiver. In this setting we can formalize the authentication property as:

$$\Phi = \text{for any run } \gamma \quad (\mathit{finish} \in \gamma\downarrow_B \Longrightarrow \mathit{start} \in \gamma\downarrow_A)$$

Please note that, since the set of runs is prefix closed, this property also implies that *start* precedes *finish* in any γ. A system is correct if for every intruder X with $\mathrm{Sort}((A\|B)\|X)\setminus \{start, finish\} \subseteq L$ and $L \cap \{start, finish\} = \emptyset$ we have

$$((S\|B)\|X) \setminus L \ \text{ satisfies } (\mathbf{sat})\ \Phi$$

It should be clear that by adapting the compositional analysis techniques of previous section, this property could be easily checked; here we prefer to show a reduction of the verification of this property to a particular secrecy property, that can be directly handled by our theory (and so by our tool, see the following section). We believe that this is an interesting result of its own, since to the best of our knowledge this is the first attempt to perform a similar reduction.

We define an encoding $\mathcal{S}$ over systems as:

$$\begin{array}{rcl} \mathcal{S}(A\|B) & = & (A'\|B') \\ A' & = & A[start := c!start_v]\|c'?(y) : special.Nil \\ B' & = & B[finish := c'!finish_v]\|c?(x) : special.Nil \end{array}$$

where c, c' are channels not occurring in $A\|B$ and $start_v, finish_v$ are distinguished values. Moreover we assume:

- The intruder cannot interact over channel c, c'. This seems reasonable since these channels appear only for checking purpose; please note that this hypothesis matches the hypothesis that *start* and *finish* actions cannot be executed by the intruder.
- Values $start_v$ and $finish_v$ are basic values such that $start_v, finish_v \notin \phi_X$ and $start_v \in \phi_A \setminus \phi_B, finish_v \in \phi_B \setminus \phi_A$.

Over the system $\mathcal{S}(A\|B)$ we will consider the following property:

$$\Psi = \text{for any run } \gamma \quad (finish_v \in K_{A,\gamma} \Longrightarrow start_v \in K_{B,\gamma}).$$

where $K_{A,\gamma}$ ($K_{B,\gamma}$) represents the knowledge of the agent A (B) after the system has performed the sequence γ. Under the above assumptions we can state:

Proposition 6.3 *If* L *is such that* $\mathrm{Sort}((A\|B)\|X) \setminus \{start, finish\} \subseteq L$ *and* $L \cap \{start, finish\} = \emptyset$ *then:*

$$((A\|B)\|X) \setminus L \ \mathbf{sat}\ \Phi \quad \textit{iff} \quad (\mathcal{S}(A\|B)\|X) \setminus L \ \mathbf{sat}\ \Psi.$$

We can use our tool to check if a system satisfies Ψ by checking if it does not satisfy:

$$\exists\gamma : \mathit{finish}_v \in K_{A,\gamma} \wedge \mathit{start}_v \notin K_{B,\gamma}.$$

Moreover if we force the intruder to eavesdrop any message sent over channels c and c', we can check the property above simply by inspecting the knowledge of the intruder i.e. checking the following property:

$$\exists\gamma : \mathit{finish}_v \in K_{X,\gamma} \wedge \mathit{start}_v \notin K_{X,\gamma}.$$

The above property can be easily analyzed by a simple modification the partial evaluation function[3]. In particular, one checks if the intruder can deduce $finish_v$ but not $start_v$, in

[3]Actually our tool is able to manage more complex conditions on the knowledge of the intruder.

the base cases:

$$(finish_v \in K^{\phi}_{X,\epsilon} \wedge startv_v \notin K^{\phi}_{X,\epsilon})//S = \begin{cases} \mathbf{T} & finish_v \in \mathcal{D}(\phi) \wedge start_v \notin \mathcal{D}(\phi) \\ \mathbf{F} & otherwise \end{cases}$$

and

$$(\exists\gamma : finish_v \in K^{\phi}_{X,\gamma} \wedge start_v \notin K^{\phi}_{X,\gamma})//Nil = \begin{cases} \mathbf{T} & finish_v \in \mathcal{D}(\phi) \wedge start_v \notin \mathcal{D}(\phi) \\ \mathbf{F} & otherwise \end{cases}$$

6.8 Technical framework for the implementation

We have built a tool for performing the analysis of authentication protocols[4]. In this section we show the more interesting theoretical aspects of our implementation. In order to implement the partial evaluation function we have to specify how $succ$, $Rsucc$ and the membership of messages in $\mathcal{D}(\phi)$ can be computed. We define the *size* of a message $|m : T|$ as 1 if T is a basic type, and as $1 + \max\{|T_i|\}_{i\in\{1,\dots,j\}}$ if $T = F^j(T_1, \dots, T_j)$. Let $Msgs(|T|)$ be the set of messages whose type has a *size* equal or smaller than T.

We consider as deduction system the one presented in figure 6.2, which enjoys our assumptions. In the following we define a canonical representation of the knowledge of agents with the aim to compute easily $\mathcal{D}(\phi) \cap Msgs(T)$ and $m : T \in \mathcal{D}(\phi)$ (a similar representation, but for a different problem has been presented in [48]).

Definition 6.3 *ϕ is* downward closed *(DC)* iff $\forall m : T \in \mathcal{D}(\phi) \setminus \phi$ *we have* $m : T \in \mathcal{D}(\phi \cap Msgs(|T| - 1))$.

It is not difficult to prove that with a set of messages ϕ that is also DC we have $m : T \in \mathcal{D}(\phi)$ iff there is a proof of $m : T$ that uses only growing rules (namely rules in which the size of the conclusion is bigger than the sizes of the premises). Hence to decide if $m : T \in \mathcal{D}(\phi)$ it is enough to follow recursively the structure of the message, checking if submessages of $m : T$ belong to ϕ. Also to compute $\mathcal{D}(\phi) \cap Msgs(T)$ we simply follow the structure of the messages, getting for basic types T_b the list $\phi \cap Msgs(T_b)$, and then correctly reconstruct an appropriate list of messages.

Definition 6.4 *ϕ is* minimal iff $\forall m : T \in \phi$ *we have* $m : T \notin \mathcal{D}(\phi \cap Msgs(|T| - 1))$.

Definition 6.5 *ϕ is a* base *for Γ if $\mathcal{D}(\phi) = \Gamma$ and ϕ is* minimal *and* downward closed.

The property of *minimality* ensures that no unnecessary message belongs to the base. In fact, if $m : T$ can be deduced by a set ϕ that is DC, then by DC can it be deduced by other messages, say $m_1 : T_1, \dots, m_n : T_n$, with $|T_i| < |T|$ for $i \in \{1, \dots, n\}$ and so $m : T \in \mathcal{D}(\phi \setminus \{m : T\})$. Now we have that $\phi \setminus \{m : T\}$ is DC and moreover $\mathcal{D}(\phi) = \mathcal{D}(\phi \setminus \{m : T\})$.

Moreover this representation enjoys the following strong property:

Proposition 6.4 *Given $\mathcal{D}(\phi)$ with ϕ* base, *if ψ is a* base *for $\mathcal{D}(\phi)$ then we have $\psi = \phi$.*

[4]Joint work with Davide Marchignoli, see [62].

The last thing we have to specify is how a base ϕ can be updated to a base ϕ' in such a way to have $\mathcal{D}(\phi') = \mathcal{D}(\phi \cup \{m : T\})$. Given ϕ ***base*** for $\mathcal{D}(\phi)$ then we define $Add(m : T, \phi)$ such that $\mathcal{D}(Add(m : T, \phi)) = \mathcal{D}(\phi \cup \{m : T\})$:

```
$Add(m : T, \phi) =$
    $\{m_1 : T_1, \ldots, m_n : T_n\} = Decompose(m : T, \phi)$
    $\phi_0 = \phi \cup \{m{:}T\}$
    **for** i=1 **to** n **do** $\phi_i = Add(m_i : T_i, \phi_{i-1})$
    $Include(m : T, \phi_n \setminus \{m : T\})$
```

where:

- $Decompose(m : T, \phi)$ is the set of messages that can be derived from $m : T$; more precisely we compute $Decompose(m : T, \phi)$ as the set of messages that can be deduced starting from message $m : T$ and applying exactly one "destructor" rule (projection rule for pairs and decryption rule for encryptions). In this way we inductively consider smaller and smaller messages to be inserted in ϕ until undecomposable messages are reached;

- $Include(m : T, \phi)$ is the *minimal* $\phi' \subseteq (\phi \cup \{m : T\})$ such that $\mathcal{D}(\phi') = \mathcal{D}(\phi \cup \{m : T\})$. To obtain $Include(m : T, \phi)$ we take advantage of the fact that all the relevant submessages of $m : T$ have already been included in ϕ, so we simply need to remove from ϕ messages directly derivable from $m : T$. What we do is to remove from ϕ all the messages that can be deduced starting from $m : T$ and applying "constructor" rules (pair formation and encryption).

From a practical point of view, our work permits the so called *on the fly* analysis technique, i.e. if there are some errors, these can be found even without the explicit analysis of the whole system. In fact it is possible to build incrementally the reduced formula. Moreover since this formula is in disjunctive form, it is satisfiable iff one of the disjuncts is satisfiable. Hence one can analyze these disjuncts separately. If an attack exists, then it must exist a disjunct of the translated formula that is satisfiable. In order to verify that the reduced formula it is not satisfiable one has to analyze the whole system S.

6.9 Optimizations

In this section we try to highlight some further optimizations for our analysis of protocols. We have already seen that the formula produced by the partial evaluation function can be reduced to a finitary one, still preserving satisfiability. Here we present other reductions on the formulas, that can improve the efficiency of the verification method. We know that by definition of deduction function, these are *monotonic*, i.e. if $\phi \subseteq \phi'$ then $\mathcal{D}(\phi) \subseteq \mathcal{D}(\phi')$. This leads to the following fact, with $\phi \subseteq \phi'$:

$$S\|_L X_\phi \models \exists\gamma : (m : T) \in K^{\phi}_{X,\gamma} \Longrightarrow S\|_L X_{\phi'} \models \exists\gamma : (m : T) \in K^{\phi'}_{X,\gamma}.$$

The above implication follows from the fact that if $X_\phi \overset{\gamma}{\mapsto}$ then $X_{\phi'} \overset{\gamma}{\mapsto}$ too. So an intuitive approach could be to consider the behaviours of an intruder where his knowledge grows as possible. This idea has been exploited by many researchers (see [87]), in particular Shmatikov and Stern claim that they first prove the soundness of this approach. Their model differs from ours, since it is based on asynchronous communications. Here we transpose their idea in our formal context and we show also the soundness of our reduction, that can be stated easier due to our logical characterization. If we look at the partial evaluation function we note that there are two possible behaviours for an intruder w.r.t. a system that can perform a communication of a message $m : T$ on a channel c (i.e. a $\tau_{c,m:T}$ action), he can wait otherwise he can eavesdrop the communication. It is clear that by idling he looses the possibility to increase his knowledge, and if an intruder can derive $m'' : T''$ by starting from ϕ then he could derive it by starting from $\phi \cup \{m : T\}$. So we have that if:

$$(\exists\gamma : (m'' : T'') \in K^{\phi}_{X,\gamma}//S') \text{ is satisfiable}$$

then

$$\langle\tau_{c,m:T}\rangle(\exists\gamma : (m'' : T'') \in K^{\phi\cup\{m:T\}}_{X,\gamma}//S') \text{ is satisfiable.}$$

Since we consider disjunctive formulas, we can safely cut off the part of the partially evaluated formula, which is obtained through the analysis of idling behaviour of the intruder with respect to a communication action, since if this formula is satisfiable then the formula corresponding to the eaves-dropping of this communication is satisfiable. Another suggestion is not to permit the intruder to send a message if an honest participant in the protocol can do that. We can formally state this reduction in our formalism as an equivalence of the satisfiability problem between:

$$\langle c?m : T\rangle\langle c!m : T\rangle(\exists\gamma : (m'' : T'') \in K^{\phi}_{X,\gamma}//S')$$

and

$$(\exists\gamma : (m'' : T'') \in K^{\phi}_{X,\gamma}//S')$$

where $S \xrightarrow{\tau_{c,m:T}} S'$ and $m : T \in \phi$.

It is clear that if $(\exists\gamma : (m'' : T'') \in K^{\phi}_{X,\gamma}//S')$ is not satisfiable then so $\langle c?m : T\rangle\langle c!m : T\rangle(\exists\gamma : (m'' : T'') \in K^{\phi}_{X,\gamma}//S')$ is not satisfiable, otherwise it is like to say that $\langle a\rangle\langle b\rangle\mathbf{F}$ is satisfiable. The other side of the equivalence is similar.

The partial order reduction techniques can be applied too. In this way we exploit the independence between actions (i.e. performing of one of the two actions do not prevent the performing of the other), by example in the case that the system has two separate agents that both can perform a sending action. In our context we can prove that:

$$\begin{array}{l}\langle c?m : T\rangle\langle c'?m' : T'\rangle(\exists\gamma : (m'' : T'') \in K^{\phi}_{X,\gamma}//S') \quad \vee \\ \langle c'?m' : T'\rangle\langle c?m : T\rangle(\exists\gamma : (m'' : T'') \in K^{\phi}_{X,\gamma}//S')\end{array}$$

is equivalent (from the satisfiability point of view) to

$$\langle c?m : T\rangle\langle c'?m' : T'\rangle(\exists\gamma : (m'' : T'') \in K^{\phi}_{X,\gamma}//S').$$

6.10 Examples

We show two examples of analysis of security protocols with our tool.

6.10.1 Needham Schroeder Public Key protocol

In this subsection we show an example protocol that has became paradigmatic for testing tools for cryptographic protocol analysis. For a long time it has been considered correct, and also proved within a logical framework. It has a simple flaw, that arises when the system is considered in presence of another agent. In figure 6.5 we present the *intended* execution between a sender and a receiver, by using the notation used in literature. In the flawed version the sender A communicates to B a fresh nonce N_a and its name encrypted with the public key of B (so only B, who knows the private key, can decrypt this message). Then the receiver B communicates to A the nonce N_a that he has received before and a fresh nonce N_b encrypted with the public key of A. Finally the sender communicates to the receiver the nonce N_b. In the intention of the designer of the protocol, at the end of a run between a sender A and a receiver B, it must be that only A and B know N_a and N_b (these nonces can be used to establish a new communication with a new shared key that is function of these values).

$$\begin{array}{ccc} \begin{array}{rcl} A \mapsto B & : & \{N_a, A\}_{PK(B)} \\ B \mapsto A & : & \{N_a, N_b\}_{PK(A)} \\ A \mapsto B & : & \{N_b\}_{PK(B)} \\ \multicolumn{3}{c}{\text{flawed version}} \end{array} & \qquad & \begin{array}{rcl} A \mapsto B & : & \{N_a, A\}_{PK(B)} \\ B \mapsto A & : & \{N_a, N_b, B\}_{PK(A)} \\ A \mapsto B & : & \{N_b\}_{PK(B)} \\ \multicolumn{3}{c}{\text{corrected version}} \end{array} \end{array}$$

Figure 6.5: Needham Schroeder Public Key protocol.

Our specification it is based on the description of the behaviour of the two components separately. We have tested our specification and as expected we have found a flaw, even if a slight different w.r.t. the one presented in [59]. An intruder is able to know the nonce N_b. To perform the verification we have only specified the initial knowledge of a possible intruder, i.e. the public keys of A and B, the names of A and B, and his private and public key. We do not need to give the nonces to the agents, since contrary to other approaches, our framework allows the intruder to guess them autonomously.

The following is a behaviour of an intruder that causes N_b to be leaked (we use $X(A)$ to represent the intruder that takes part to a communication as the agent A):

$$\begin{array}{lcl} A \mapsto X & : & E(Xkey, (N_a, A)) \\ X(A) \mapsto B & : & E(Bkey, (N_a, A)) \\ B \mapsto X(A) & : & E(Akey, (N_a, N_b)) \\ X \mapsto A & : & E(Akey, (N_a, N_b)) \\ X(A) \mapsto B & : & E(Akey, N_a) \\ A \mapsto X & : & E(Xkey, N_b) \end{array}$$

The attack performed can be summarized as follows: the agent A starts a run of the protocol with the agent X; then the agent X can simulate A in a run of the protocol with the agent B. The agent B sends to $X(A)$ the message $E(Akey, (N_a, N_b))$, which contains the fresh nonce N_b, encrypted with A public key. Now the intruder is not able to decrypt directly the message, but he can send the message to the agent A. The agent A will correctly decrypt $E(Akey, (N_a, N_b))$ and then reply the nonce N_b to X, encrypted with X public key, since he thinks that is the second message of his run with X. Now X knows N_b! It is interesting to note that the above intruder is not very clever, since he sends to B, as last message, a non correct message (encrypted with a wrong key), by permitting B to understand that there are some problems. A clever intruder can wait to receive the correct message from A, and then resend N_b to B, correctly encrypted. Also this intruder can be found by our *secrecy* analysis. By performing *authentication* analysis only the latter intruder can be found. This is reasonable since a *secrecy* attack can be performed even without an authentication attack. In fact, we may be interested that the information exchanged during a session of a protocol is kept secret, even though the protocol does not terminate correctly.

Indeed, we have corrected the protocol similarly to [59] and we have verified that there are no flaws. The presented attack is found in few seconds by our tool, and the verification of the corrected version takes less than a minute on a Pentium PC, with Linux operating system.

It is interesting to note that we do not need to introduce a specification for an intruder.

6.10.2 ISO SC27 Protocol

In this subsection we analyze an authentication protocol, namely the ISO SC27 protocol (see [45]). It has been proposed as ISO standard but a flaw was found in [11].

Here we show the flexibility of our tool by analyzing this protocol, which is based on shared key encryption, namely the key and the inverse key are the same. The deduction function used in the tool does not model correctly all the aspects of this encryption schema[5]. Nevertheless, by resorting to a simple trick we can use the tool to find a subtle attack on the protocol. Thus we assume that both A and B know a public key and a private key. The encryption will be performed by using the public key and the decryption by using the private key, which in this case is shared between A and B.

$$\begin{array}{lcl} A \mapsto B & : & N_a \\ B \mapsto A & : & \{N_a\}_{S_{AB}}, \{N_b\}_{S_{AB}} \\ A \mapsto B & : & N_b \end{array}$$

Figure 6.6: ISO SC27 protocol.

[5]We are currently extending the tool's features to include a suitable deduction function for shared key encryption.

The protocol should ensure authentication among parties. In particular in figure 6.6 we give the intended execution. A sends a nonce N_a to B. B replays the encryption of N_a with the shared Key S_{AB} together with the encryption of another nonce N_b. Then A decrypts N_b and sends it to B.

We have started the analysis by supposing that the initial knowledge of the intruder is empty. We use an *authentication* analysis as proposed in section 6.7. The attack found by the tool is the following (see also [11]):

$$\begin{array}{rcl} A \mapsto X(B) & : & N_a \\ X(B) \mapsto A & : & N_a \\ A \mapsto X(B) & : & E(ABkey, N_a), E(ABkey, N_b) \\ X(B) \mapsto A & : & E(ABkey, N_a), E(ABkey, N_b) \\ A \mapsto X(B) & : & N_b \\ X(B) \mapsto A & : & N_b \end{array}$$

This attack is very subtle since it may be performed even without the presence of the receiver B. In fact, we have modeled a situation where the agent A can start two parallel sessions, one in which he plays the role of sender and the other in which plays the role of receiver. Simply the intruder uses A as an oracle for the decryption of messages.

This shows that the verification of authentication protocols is very difficult. Because there is the necessity to check the protocol in the actual setting where it runs. In fact if we check the protocol in a setting where the agent A only acts as sender then the previous attack cannot be performed and hence it is not reported by our analysis.

6.11 Conclusions and related work

We have proposed a new approach for checking security properties of cryptographic protocols. This approach is a transposition of the ideas we have proposed in chapter 5 (or see [67]) for the analysis of *non interference* properties ([32, 34, 35, 36]).

We believe that underspecification can be seen as a suitable method for specifying security properties. Hence, if one accepts the previous idea then partial evaluation techniques seem to provide a unique conceptual framework for the analysis of security properties. By looking at our results in the previous chapters on *non interference* and at the ones in this chapter we can note that are obtained by following the schema below:

1. design suitable languages for system description and property specification;

2. develop partial evaluation techniques;

3. develop satisfiability procedure for the logic.

Obviously, the technical frameworks are in general different, but we feel that the previous steps can be considered as general guidelines for the analysis of security properties, when these are defined through underspecification. In particular the concepts recalled in chapter 2, even though not always applicable directly (just like in the security analysis of this chapter), may provide useful hints in performing the second step of the previous schema.

The bases for a systematic study of security aspects of distributed systems are provided by the great amount of work performed in concurrency theory for the definition of concurrent languages for the description of several settings and by the sophisticated decision procedures techniques defined in mathematical logic.

In this line of research we plan to extend our approach for the analysis of cryptographic protocols, by considering the possibility that the intruder has a probability to guess the correct key for an encrypted message (see [57] for an attempt). We hope to borrow some concepts from [53], where partial evaluation techniques are developed for a process language which can express probabilistic aspects of concurrent systems.

We believe that the method for security analysis exposed in this thesis is very flexible and may have a wide range of applications. Until now we have used partial evaluation techniques for the analysis of *non interference, timed non interference, secrecy* and *authentication*.

We think that other verification problems may be faced with our analysis. For example we feel that *non repudiation* properties of cryptographic protocols may be correctly defined through underspecification and solved by means of partial evaluation techniques (see [85]).

Related work

The literature on security properties analysis and on verification of cryptographic protocols is wide. For an impressive overview of cryptographic protocols and cryptographic systems see [83]. Here we briefly recall some approaches related to process algebra theory and process logics.

To our knowledge, the only previous attempt to analyze *non interference* and cryptographic protocols within the same conceptual framework, has been proposed by Focardi and Gorrieri ([32, 35]). In their work an explicit description for a particular, actually the most general, intruder is requested.

The same limitation is present in the seminal work of Lowe (see [59]), who applies generic tools for verification of process algebra terms for the analysis of cryptographic protocols. In the aforementioned paper, Lowe shows how starting from the results of the analysis on a finite number of runs, one can deduce the correctness of the whole behaviour of the protocol.

Perhaps, a work more similar to ours is the one of Marrero *et al.* (see [63]), where a model with sequential agents is used and the explicit description of an intruder is not needed since an axiomatic behaviour of the intruder is supposed. But the work is more limited in its scopes, since they do not permit the intruder to guess nonces and uses two different methodologies for secrecy and authentication, and seems not to be directly generalizable (they make reasoning with a fixed theory), while generality and flexibility are major topics of our work. Actually, our approach is from the opposite direction, the behaviour of an intruder is automatically considered when one applies the point of view of compositional analysis.

Other approaches are based on proof theoretic methods (see [1, 2, 37, 48, 78, 84]). Some of them use temporal and modal logic concepts and permit to prove that a system, even though with a non finite behavior, enjoys security properties. In general these methods are not fully automated and need non trivial human efforts to analyze systems, moreover counterexamples are not directly feasible.

An interesting exception is the work of Kindred and Wing in [48], where the authors propose a fully automated original approach for checking that a protocol enjoys some properties expressed in a logical language L. Roughly, the method is based on a finite representation of a theory generated by the set of initial assumptions of the protocol, the protocol itself (expressed as a set of logical formulas), and the axioms and rules of the logical language L (this language has to verify some requirements, but some authentication logics reported in literature may be used).

Some authors have found the π calculus (see [71]) as a suitable language for describing security protocols. This is mainly due to the management of names (that can be seen as secrets) of this process language. In [1], Abadi and Gordon have proposed an approach based on proof theoretic tools for a variant of the $\pi-$calculus. The idea is to model the intruders by using testing equivalence theory for $\pi-$calculus (see [15]). They analyze protocols based on shared key encryption, but since their language is very powerful it seems difficult to automatize their approach. Another approach relies on control flow analysis techniques of the $\pi-$calculus (see [14]). By controlling how the information is exchanged along channels, it is possible to study confinement properties, namely whether information is never sent along a particular channel and remains enclosed in a system.

Bibliography

[1] M. Abadi and A. D. Gordon. Reasoning about cryptographic protocols in the spi calculus. In *CONCUR '97: Concurrency Theory, 8th International Conference*, volume 1243 of *Lecture Notes in Computer Science*, pages 59–73, 1997.

[2] M. Abadi and M. R. Tuttle. A semantics for a logic of authentication. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216. ACM Press, 1991.

[3] L. Aceto, B. Bloom, and F. Vaandrager. Turning SOS rules into equations. *Information and Computation*, 111(1):1–52, 1994.

[4] H. R. Andersen. *Verification of Temporal Properties of Concurrent Systems*. PhD thesis, Department of Computer Science, Aarhus University, Denmark, 1993.

[5] H. R. Andersen. Partial model checking (extended abstract). In *Proceedings of 10th Annual IEEE Symposium on Logic in Computer Science*, pages 398–407. IEEE Computer Society Press, 1995.

[6] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, IX(2):127–168, 1986.

[7] H. Bekič. Definable operations in general algebras, and the theory of automata and flow charts. In C. Jones, editor, *Hans Bekič: Programming Languages and Their Definition*, volume 177 of *Lecture Notes in Computer Science*, pages 30–55. Springer-Verlag, 1984.

[8] D. E. Bell and L. J. La Padula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-301, MITRE MTR-2997, March 1976.

[9] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202(1–2):1–54, 1998.

[10] G. Bhat and R. Cleaveland. Efficient model checking via the equational μ-calculus. In *Proceedings, 11*$^{\text{th}}$ *Annual IEEE Symposium on Logic in Computer Science*, pages 304–312, New Brunswick, New Jersey, 1996. IEEE Computer Society Press.

[11] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of a Family of Attack Resistant Authentication Protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, 1993.

[12] B. Bloom. Structured operational semantics as a specification language. In *Conference Record of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95)*, pages 107–117, San Francisco, California, 1995. ACM Press.

[13] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.

[14] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Control flow analysis for the pi-calculus. In *Proceedings of 9th International Conference on Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 84–98. 1998.

[15] M. Boreale and R. De Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120(2):279–303, 1995.

[16] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–439. IEEE Computer Society Press, 1990.

[17] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, DEC Systems Research CENTER, 1989.

[18] J. Clark and J. Jacob. A survey of authentication protocol literature. November 1997.

[19] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[20] E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[21] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal $\mu-$calculus. August 1993.

[22] M. Dam. CTL* and ECTL* as fragments of modal $\mu-$calculus. *Theoretical Computer Science*, 126:77–96, 1994.

[23] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24(2):211–237, 1987.

[24] R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes (concurrent programming). *Theoretical Computer Science*, 34(1-2):83–133, 1984.

[25] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *Journal of the ACM*, 42(2):458–487, 1995.

[26] Degano and Priami. Enhanced operational semantics. In *Computing Surveys*, volume 28. 1996.

[27] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 996–1072, Amsterdam, 1990. Elsevier Science Publishers.

[28] E. A. Emerson and E. M. Clarke. Design and synthesis of synchronization skeletons using branching time logic. *Science of Computer Programming*, 2:241–266, 1982.

[29] E. A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *IEEE Symposium on Foundations of Computer Science (FOCS'88)*, 1988.

[30] R. Focardi. Comparing two information flow security properties. In *Proceedings of The 9th Computer Security Foundations Workshop*, pages 116–122. IEEE Computer Society Press, 1996.

[31] R. Focardi. *Analysis and Automatic Detection of Information Flows in Systems and Networks*. PhD thesis, Department of Computer Science, University of Bologna, 1998.

[32] R. Focardi, A. Ghelli, and R. Gorrieri. Using non interference for the analysis of security protocols. In H. Orman and C. Meadows, editors, *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*. DIMACS Center, CoRE Building, Rutgers University, September 1997.

[33] R. Focardi and R. Gorrieri. An information flow security property for CCS. In *Second North American Process Algebra Workshop (NAPAW '93)*, 1993. TR 93-1369, Cornell (Ithaca).

[34] R. Focardi and R. Gorrieri. A classification of security properties. *Journal of Computer Security*, 3(1):5–33, 1995.

[35] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 27:550–571, 1997.

[36] J. A. Goguen and J. Meseguer. Security policy and security models. In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.

[37] J. W. Gray, III and J. McLean. Using temporal logic to specify and verify cryptographic protocols. In *Proceedings of The 8th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1995.

[38] J. W. Gray, III and P. F. Syverson. A logical approach to multilevel security of probabilistic systems. In *Proceedings of the 1992 IEEE Computer Society Symposium on Security and Privacy*, pages 164–176. IEEE Computer Society Press, 1992.

[39] J. F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[40] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994.

[41] E. Haghverdi and H. Ural. Submodule construction using derivatives. Technical Report TR-95-13, Computer Science Department, University of Ottawa, 1995.

[42] D. Harel and A. Pnueli. On the development of reactive systems. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of NATO *Advanced Summer Institute*, pages 477–498, 1985.

[43] M. Hennessy. *Algebraic Theory of Processes*. The MIT Press, Cambridge, Mass., 1988.

[44] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

[45] ISOSC27. Entity Authentication Using Symmetric Techniques. Report ISO/IEC JTC1.27.02.2 (20.03.1.2), June 1990.

[46] D. Janin and I. Walukiewicz. Automata for the μ-calculus and related results. In J. Wiedermann and P. Hájek, editors, *Proceedings 20th Intl. Symp. on Mathematical Foundations of Computer Science, MFCS'95*, volume 969 of *Lecture Notes in Computer Science*, 1995.

[47] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[48] D. Kindred and J. M. Wing. Fast, automatic checking of security protocols. In *Second USENIX Workshop on Electronic Commerce*, pages 41–52, Oakland, California, 1996.

[49] D. Kozen. Results on the propositional $\mu-$calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[50] O. Kupferman and M. Y. Vardi. Module checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 75–86. Springer Verlag, 1996.

[51] O. Kupferman and M. Y. Vardi. Module checking revisited. In *Proceedings of the Ninth International Conference on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 36–47. Springer-Verlag, 1997.

[52] O. Kupferman, M. Y. Vardi, and P. Wolper. Module checking. Technical Report TR98-302, Rice University, 1998.

[53] K. G. Larsen and A. Skou. Compositional verification of probabilistic processes. *Lecture Notes in Computer Science*, 630:456–467, 1992.

[54] K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 185–215. IEEE Computer Society Press, 1990.

[55] K. G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. *Journal of Logic and Computation*, 1(6):761–795, 1991.

[56] F. Levi. *Verification of temporal and Real-Time Properties of Statecharts*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1997.

[57] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *ACM Symposium in Computer and Communication Security*, 1998.

[58] J. Lind-Nielsen. Mudiv: A program performing partial model checking. Master's thesis, Department of Information Technology, Technical University of Denmark, September 1996.

[59] G. Lowe. Breaking and fixing the needham schroeder public-key protocol using FDR. In *Proceedings of Tools and Algorithms for the Construction and the Analisys of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996.

[60] G. Lowe. Some new attacks upon security protocols. In *Proceedings of The 9th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[61] G. Lowe and B. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, 1997.

[62] D. Marchignoli and F. Martinelli. Automatic verification of cryptographic protocols through compositional analysis techniques. *To appear in Proceedings of the International Conference on Tools and Algorithms for the Construction and the Analysis of Systems*, 1999.

[63] W. Marrero, E. Clarke, and S. Jha. A model checker for authentication protocols. In H. Orman and C. Meadows, editors, *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*. DIMACS Center, Rutgers University, September 1997.

[64] F. Martinelli. An uniform approach for the analysis of open systems. Submitted for publication.

[65] F. Martinelli. An improvement of algorithms for solving interface equations. *Information Processing Letters*, 67(4):185–190, 1998.

[66] F. Martinelli. Languages for description and analysis of authentication protocols. In *Proceedings of 6th Italian Conference on Theoretical Computer Science*, pages 304–315. World Scientific, 1998.

[67] F. Martinelli. Partial model checking and theorem proving for ensuring security properties. In *Proceedings of 11th Computer Security Foundations Workshop*, pages 44–52. IEEE Computer Society Press, 1998.

[68] P. Merlin and G. V. Bochmann. On the construction of submodule specification and communication protocols. *ACM Transactions on Programming Languages and Systems*, 5:1–25, 1983.

[69] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[70] R. Milner. Operational and algebraic semantics of concurrent processes. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 19, pages 1201–1242. The MIT Press, New York, N.Y., 1990.

[71] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, 1992.

[72] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi. In *Proceedings of the Symposium on Security and Privacy*, pages 141–153. IEEE Computer Society Press, 1997.

[73] R. M. Needham and M. D. Schroder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[74] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Proceedings of Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 526–548, 1992.

[75] D. Niwinski and I. Walukiewicz. Games for the mu -calculus. *Theoretical Computer Science*, 163(1-2):99–116, 1996.

[76] D. Park. Concurrency and automata on infinite sequences. In *Proceedings 5th GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, 1981.

[77] J. Parrow. Submodule construction as equation solving in CCS. *Theoretical Computer Science*, 68(2):175–202, 1989.

[78] L. C. Paulson. Proving properties of security protocols by induction. In *Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[79] G. D. Plotkin. A Structural Approach to Operational Semantics. Tech. Rep. FN-19, DAIMI, Univ. of Aarhus, Denmark, Sept. 1981.

[80] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57. IEEE Computer Society Press, 1977.

[81] H. Qin and P. Lewis. Factorisation of finite state machines under strong and observational equivalences. *Formal Aspects of Computing*, 3(3):284–307, 1991.

[82] A. W. Roscoe and M. H. Goldsmith. The perfect spy for model-checking crypto-protocols. In H. Orman and C. Meadows, editors, *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*. DIMACS Center, Rutgers University, September 1997.

[83] F. B. Schneider. *Applied Cryptography*. J. Wiley & sons, Inc., 1996.

[84] S. Schneider. Verifying authentication protocols with CSP. In *Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[85] S. Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of The 11th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1998.

[86] M. W. Shields. Implicit system specification and the interface equation. *The Computer Journal*, 32(5):399–412, 1989.

[87] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *Proceedings of 11th Computer Security Foundations Workshop*, pages 105–116. IEEE Computer Society Press, 1998.

[88] D. P. Sidhu and J. Aristizabal. Constructing submodule specifications and network protocols. *IEEE Transactions on Software Engineering*, 14:1565–1577, 1988.

[89] R. D. Simone. Hiher-level synchronizing devices in meije-sccs. *Theoretical Computer Science*, 37, 1985.

[90] A. K. Simpson. Compositionality via cut-elimination: Hennessy-Milner logic for an arbitrary GSOS. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 420–430. IEEE Computer Society Press, 1995.

[91] B. Steffen and A. Ingólfsdóttir. Characteristic formulae for processes with divergence. *Information and Computation*, 110(1):149–163, 1994.

[92] C. Stirling. Modal and temporal logics for processes. In *Logics for Concurrency: Structures versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 149–237, 1996.

[93] R. S. Street and E. A. Emerson. The propositional $\mu-$calculus is elementary. In *Eleventh International Colloquium in Automata, Languages and Programming*, volume 172 of *LNCS*, pages 465–472, 1984.

[94] R. S. Street and E. A. Emerson. An automata theoretic procedure for the propositional $\mu-$calculus. *Information and Computation*, 81(3):249–264, 1989.

[95] R. S. Streett. Fixpoints and program looping: Reductions from the propositional $\mu-$calculus into propositional dynamic logics of looping. In R. Parikh, editor, *Proceedings of the Conference on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 359–372, Brooklyn, NY, 1985. Springer.

[96] P. Syverson. On unifying some cryptographic protocol logics. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 14–28. IEEE Computer Society Press, 1994.

[97] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[98] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier Science Publishers B. V., 1990.

[99] I. Ulidowski and S. Yuen. Extending process languages with time. In *Proceedings of the 6th International Conference on Algebraic Methodology and Software*

Technology, volume 1349 of *Lecture Notes in Computer Science*, pages 524–538, 1997.

[100] R. J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90: Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297, Amsterdam, The Netherlands, 1990. Springer-Verlag.

[101] M. Y. Vardi. Verification of open systems. In *Proceedings of the 17th Conference on Foundations of Software Tecnology and Theoretical Computer Science*, Dec. 1997.

[102] M. Y. Vardi and P. Wolper. Automata theoretic techniques for modal logics of programs. In *ACM Symposium on Theory of Computing (STOC '84)*, pages 446–456, Baltimore, USA, 1984. ACM Press.

[103] I. Walukiewicz. On completeness of the μ-calculus. In *Proceedings 8th Annual IEEE Symp. on Logic in Computer Science (LICS'93)*, pages 136–146, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[104] I. Walukiewicz. *A Complete Deductive System for the mu-Calculus*. PhD thesis, Warsaw University, 1994.

[105] I. Walukiewicz. A complete deductive system for the μ-calculus. Technical Report RS-95-6, BRICS, Department of Computer Science, University of Aarhus, Denmark, 1995.

[106] G. Winskel. On the compositional checking of validity. In *Proceedings of CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*, pages 481–501, 1990.

[107] G. Winskel. *The Formal Semantic of Programming Languages*. MIT Press, Cambridge, Moss, 1993.

Appendix A

Proofs of chapter 6

This appendix contains the proofs of chapter 6. A first technical lemma is the following:

Lemma A.1 *Suppose that there exists a sequential agent X_ϕ s.t. $S\|_L X_\phi \xrightarrow{\tilde{\gamma_1}} \ldots \xrightarrow{\tilde{\gamma_n}}$ and $(S\|_L (X)_\phi \xrightarrow{\tilde{\gamma_1}} \ldots \xrightarrow{\tilde{\gamma_n}}) \downarrow_X = \gamma = (\gamma_1 \ldots \gamma_j)$, and $m : T$ is an initial message of S, with $m : T \in \mathcal{D}(\phi \cup msgs(\gamma))$ then :*

- *there exists X' s.t. $S\|_L X'_\phi \xrightarrow{\tilde{\gamma'_1}} \ldots \xrightarrow{\tilde{\gamma'_{n'}}}$ and $(S\|_L X'_\phi \xrightarrow{\tilde{\gamma'_1}} \ldots \xrightarrow{\tilde{\gamma'_{n'}}}) \downarrow_{X'} = \gamma' = (\gamma'_1 \ldots \gamma'_{j'})$ with $m : T \in \mathcal{D}(\phi \cup msgs(\gamma'))$, and moreover if $\gamma'_k, k \in \{1, \ldots, j'\}$ is the sending of a message $m' : T'$ then there exists a minimal h such that every $l, h \leq l < k$ we have that γ'_l is the generation of a random value g, which is a submessage of $m' : T'$, and every random submessage of $m' : T'$ that does not belong to $\mathcal{D}(\phi \cup msgs(\{\gamma'_1 \ldots \gamma'_h\}))$ is generated in this sequence. Moreover $\tilde{\gamma'}$ preserve the adjacency of every sequence of generation actions performed by X.*

Proof: First, we show that if in γ there is $\gamma_i = \tau_{g_i:T_i}$ and $g_i : T_i$ is never sent, as submessage, during the following $\gamma_{j'}$ with $i \leq j' \leq j$ then we have (we assume that i is the maximal index of of such action):

$$m : T \in \mathcal{D}(\phi \cup \{\gamma_k\}_{k \in \{1,\ldots,j\} \setminus \{i\}})$$

This follows from the fact that γ_i does not appear as submessage of $\phi \cup \{\gamma_1, \ldots, \gamma_{i-1}\}$ since it is newly created; moreover since it is never sent as submessage then it cannot be received as submessage in every action in $\gamma_{j'}$ with $i \leq j' \leq j$, since $g_i : T_i$ is a basic random value and it cannot be guessed from no-one else than X, and by assumption 6.4 this message cannot be deduced by any other agent in S. So, by observing that the possible actions in γ can be only generation, sending, receiving or eaves-dropping we can argue that $g_i : T_i \notin SubM(\phi \cup \{\gamma_k\}_{k \in \{1,\ldots,j\} \setminus \{i\}})$. By applying assumption 6.3 we have $m : T \in \mathcal{D}(\phi \cup \{\gamma_k\}_{k \in \{1,\ldots,j\} \setminus \{i\}})$. So we can build a process that after action γ_{i-1} goes in a process X'_{i+1}, and by the assumption on the deduction function we have $X'_{i+1} \overset{\gamma_{i+1} \ldots \gamma_j}{\mapsto} X'_j$, since the lack of knowledge of $g_i : T_i$ does not prevent X_{i+1} to deduce everything he could deduce with it. This result is based on the fact that (it can be proven by induction on the length of γ) if a process X with initial knowledge ϕ can perform γ then it is possible to build another term X' that does not use neither equality nor choice and produces this sequence. So if the value $g_i : T_i$ is never sent then it cannot be substituted in a sending action and hence the result follows. By repeating this step a finite number of times we get a process that does not generate a random value that he will not send as submessage.

Let us see how we can reorder the sequence in such a way that every sequence of generations of random values is followed by a sending of those values as submessages. This fact implies that a minimal h, which satisfies our requirements, must exist in γ.

Let us suppose to have a generation action in the sequence, there are various cases:

- before the sending of $g_i : T_i$ another sending is performed, where $g_i : T_i$ does not appear as submessage, then X can perform $\tau_{g_i:T_i}$ after the sending action, since $g_i : T_i$ is not influent for the performing of this kind of action how it can be proved by structural induction on X.

- before the sending of $g_i : T_i$ there is a receiving, then he can perform the generation action after the receiving action.

- before the sending of $g_i : T_i$ there is a generation action, good.

- before the sending of $g_i : T_i$ there is an eaves-dropping action, then he can perform the generation action after this action.

By applying a finite number of times this reduction we can build a process where every generation action of a random value $g_i : T_i$ is followed by a sequence of generation actions and finally by a sending of a message that contains $g_i : T_i$ as submessages. □

This result can be used to prove the following lemma:

Lemma 6.1 *If there exists a sequential agent X_ϕ s.t. $S\|_L X_\phi \models \exists\gamma : (m : T) \in K^{\phi}_{X,\gamma}$ then a well behaved agent X'_ϕ exists s.t. $S\|_L X'_\phi \models \exists\gamma : (m : T) \in K^{\phi}_{X',\gamma}$.*

We can observe that from the semantics of $\|_L$ it follows:

Lemma A.2 *Suppose $Sort(S) \cup Sort(X) \subseteq L$ then:*

$$\exists\gamma = \beta\gamma' : S\|_L X_\phi \models (m : T) \in K^{\phi}_{X,\gamma}$$
$$\text{iff}$$
$$\exists\beta : S\|_L X \xrightarrow{\beta} S'\|_L X'_{\phi'} \wedge S'\|_L X' \models \exists\gamma' : (m : T) \in K^{\phi'}_{X,\gamma'} \quad \vee$$
$$\exists\beta : S\|_L X \xrightarrow{\beta} S'\|_L X_{\phi} \wedge S'\|_L X \models \exists\gamma' : (m : T) \in K^{\phi}_{X,\gamma'} \quad \vee$$
$$\exists\beta : S\|_L X \xrightarrow{\beta} S\|_L X'_{\phi'} \wedge S\|_L X' \models \exists\gamma' : (m : T) \in K^{\phi'}_{X,\gamma'}.$$

Moreover, due to our assumptions on the deduction function we can state:

Lemma 6.2 *If $S \equiv_\sigma S'$ then $S \xrightarrow{a} S_1 \Longrightarrow S' \xrightarrow{\sigma(a)} S'_1$ and $S' \equiv_\sigma S'_1$.*

Proof: By structural induction on S and by inspection on the rules used to infer the transition of S. □

The above lemma can be used to prove that if $S \overset{\gamma}{\mapsto} S_1$ and $S \equiv_\sigma S'$ then $S' \overset{\sigma(\gamma)}{\mapsto} S'_1$.

The correctness of the partial evaluation is stated by the following proposition, where we assume to deal with well behaved processes.

Proposition 6.2 *If $m : T$ is an initial message and $Sort(S) \cup Sort(X) \subseteq L$ then $S\|_L X_\phi \models \exists\gamma : m : T \in K^\phi_{X,\gamma}$ iff $X_\phi \models \exists\gamma : m : T \in K^\phi_{X,\gamma}//S$*

Proof: By induction on the max depth of the derivations of S.
Base case:

$S = Nil$ then the result follows by observing that X can only deduce initial messages during its future behaviour that it could deduce in its initial position. So the result follows.
Inductive step:

$$\begin{array}{ll}
S\|_L X_\phi \models \exists\gamma : (m : T) \in K^\phi_{X,\gamma} & \text{iff} \\
\exists\gamma \text{ well behaved} : S\|_L X_\phi \models (m : T) \in K^\phi_{X,\gamma} & \text{iff} \\
S\|_L X_\phi \models (m : T) \in K^\phi_{X,\epsilon} \qquad \vee & \\
(\exists\gamma \text{ well behaved}, \gamma = \beta\gamma' \wedge S\|_L X_\phi \models (m : T) \in K^\phi_{X,\gamma} & \text{iff}_{(lemmaA.2)} \\
X_\phi \models (m : T) \in K^\phi_{X,\epsilon} \qquad \vee & \\
(1)\ \exists\beta : S\|_L X \xrightarrow{\beta} S'\|_L X'_{\phi'} \wedge S'\|_L X' \models \exists\gamma' : (m : T) \in K^{\phi'}_{X,\gamma'} \quad \vee & \\
(2)\ \exists\beta : S\|_L X \xrightarrow{\beta} S'\|_L X_\phi \wedge S'\|_L X \models \exists\gamma' : (m : T) \in K^\phi_{X,\gamma'} \quad \vee & \\
(3)\ \exists\beta : S\|_L X \xrightarrow{\beta} S\|_L X'_{\phi'} \wedge S\|_L X' \models \exists\gamma' : (m : T) \in K^{\phi'}_{X,\gamma'} &
\end{array}$$

Condition (1) takes in account possible synchronizations between S and X; these are (from the point of view of X agent):

- receiving a typed message on a channel c. Since only a finite set of messages can be sent from S on every channel and the number of channels is finite, we can consider only a finite number of actions β. So we have that $X_\phi \xrightarrow{c?m':T'} X'_{\phi\cup\{m':T'\}}$ and $S'\|_L X'_{\phi\cup\{m':T'\}} \models \exists\gamma' : (m : T) \in K^{\phi\cup\{m':T'\}}_{X,\gamma'}$. By inductive hypothesis we have that $X'_\phi \models \exists\gamma : (m : T) \in K^\phi_{X,\gamma}//S'$. So considering all the possible β actions of this kind we obtain:

$$\bigvee_{S \xrightarrow{c!m':T'} S'} \langle c?m' : T'\rangle(\exists\gamma : (m : T) \in K^{\phi\cup\{m':T'\}}_{X,\gamma}//S')$$

- sending of a typed message, the set of types that the system S can receive is finite and for every type every agent can deduce only a finite number of messages by assumption 6.1, hence by inductive hypothesis, the following disjunction takes in account these cases:

$$\bigvee_{(c,m':T',S')\in succ(S)} \langle c!m' : T'\rangle(\exists\gamma : (m : T) \in K^{\phi}_{X,\gamma}//S')$$

- eaves-dropping of a communication internal to the system S, such communications can be in a finite number, and hence by inductive hypothesis these cases can be treated as:
$$\bigvee_{S\overset{\tau_{c,m':T'}}{\longrightarrow}S'} \langle \chi_{c,m':T'}\rangle(\exists\gamma : (m : T) \in K^{\phi\cup\{m':T'\}}_{X,\gamma}//S')$$

Condition (2) takes in account actions performed by the system S without interaction with the agent X, by inductive hypothesis these actions can be taken in account using the following formula:

$$\bigvee_{S\overset{a}{\longrightarrow}S'} \exists\gamma : m : T \in K^{\phi}_{X,\gamma}//S'$$

The last condition (3) is more difficult to translate than the previous ones, since in this formulation it does not directly permit to use the induction hypothesis on the structure of S. The restriction to the analysis of well behaved processes and our requirements on the $Sort$ of S and X help us, in fact it follows that the only possibility is that $\beta = \tau_{g_1:T_1}$. Since γ is well behaved then if $S\|_L X \models m : T \in K^{\phi}_{X,\gamma}$ and γ can be rewritten as $\beta\gamma'\tau_{c,m':T'}\gamma''$, where $\beta\gamma' = \tau_{g_1:T_1}\dots\tau_{g_n:T_n}$ are generation actions. Moreover the random values, guessed in this sequence, appear as submessages in $m' : T'$ (see lemma A.1); and $\tau_{c,m':T'}$ is derived from the synchronization of a sending action $c!m' : T'$ by X and a receiving action performed by the system S. By definition of well behaved process, we have that $X_\phi \overset{\beta\gamma' c!m':T'}{\mapsto} X'_{\phi\cup msgs(\beta\gamma')}$ and $S\|_L X_\phi \overset{\beta\gamma'\tau_{c,m':T'}}{\mapsto} S'\|_L X'_{\phi\cup msgs(\beta\gamma')}$ and $S'\|_L X'_{\phi\cup msgs(\beta\gamma')} \models \exists\gamma : (m : T) \in K^{\phi\cup msgs(\beta\gamma')}_{X,\gamma}$.

We can now apply inductive hypothesis and so we obtain $X'_{\phi\cup msgs(\beta\gamma')} \models (\exists\gamma : m : T \in K^{\phi\cup msgs(\beta\gamma')}_{X,\gamma}//S')$. Hence, $X_\phi \models \langle\tau_{g_1:T_1}\rangle\dots\langle\tau_{g_n:T_n}\rangle\langle c!m' : T'\rangle(\exists\gamma : (m : T) \in K^{\phi\cup msgs(\beta\gamma')}_{X,\gamma}//S')$.

By the other hand, if

$$X_\phi \models \langle\tau_{g_1:T_1}\rangle\dots\langle\tau_{g_n:T_n}\rangle\langle c!m' : T'\rangle(\exists\gamma : (m : T) \in K^{\phi\cup msgs(\beta\gamma')}_{X,\gamma}//S')$$

then it can be proven that a well behaved γ exists s.t. $S\|_L X_\phi \models m : T \in K^{\phi}_{X,\gamma}$. So considering all possible initial sequences of random generation actions we have:

$$\bigvee_{(c,m':T',\langle\langle g_i:T_i\rangle\rangle_{i\in I},S')\in Rsucc(S)} \langle\langle\tau_{g_i:T_i}\rangle\rangle_{i\in I}\langle c!m' : T'\rangle(\exists\gamma : (m{:}T) \in K^{\phi'}_{X,\gamma}//S')$$

that is an infinitary disjunction, where $\phi' = \phi \cup \langle\langle g_i : T_i\rangle\rangle_{i\in I}$.

□

The following lemma states the key idea for the reduction of infinitary disjunctions to finitary ones.

Lemma 6.3 *If $m : T$ is an initial message, σ a bijection between $\{g_i : T_i\}_{i\in I}$ and $\{g'_i : T'_i\}_{i\in I}$ with $S'\|X' \equiv_\sigma S\|X$ and $SubM(\phi) \cap \{g_i : T_i\}_{i\in I} = \emptyset = SubM(\phi) \cap \{g'_i : T'_i\}_{i\in I}$ then:*

$$\begin{array}{c} S\|_L X_{\phi\cup\{g_i:T_i\}_{i\in I}} \models \exists\gamma : (m : T) \in K^{\phi\cup\{g_i:T_i\}_{i\in I}}_{X,\gamma} \\ \text{iff} \\ S'\|_L X'_{\phi\cup\{g'_i:T'_i\}_{i\in I}} \models \exists\gamma : (m : T) \in K^{\phi\cup\{g'_i:T'_i\}_{i\in I}}_{X',\gamma}. \end{array}$$

Proof: if $S\|_L X_{\phi\cup\{g_i:T_i\}_{i\in I}} \models \exists\gamma : (m : T) \in K^{\phi\cup\{g_i:T_i\}_{i\in I}}_{X,\gamma}$ then there exists γ such that

$$(S\|_L X_{\phi\cup\{g_i:T_i\}_{i\in I}} \overset{\gamma}{\mapsto}) \downarrow_X = \tilde{\gamma}$$

with $m : T \in \mathcal{D}(\phi \cup \{g_i : T_i\}_{i\in I} \cup msgs(\tilde{\gamma}))$. We are in the hypothesis of lemma 6.2 so we have $S'\|_L X'_{\phi\cup\{g'_i:T'_i\}_{i\in I}} \overset{\gamma'}{\mapsto}$ with $\gamma' = \sigma(\gamma)$. Hence we can see that $(S'\|_L X'_{\phi\cup\{g'_i:T'_i\}_{i\in I}} \overset{\gamma'}{\mapsto}) \downarrow_{X'} = \tilde{\gamma'} = \sigma(\tilde{\gamma})$. So we have:

$$\begin{array}{ll} m : T \in \mathcal{D}(\phi \cup \{g_i : T_i\}_{i\in I} \cup msgs(\tilde{\gamma})) & \text{iff} \\ \sigma(m : T) \in \mathcal{D}(\sigma(\phi \cup \{g_i : T_i\}_{i\in I} \cup msgs(\tilde{\gamma}))) & \text{iff} \\ \sigma(m : T) \in \mathcal{D}(\sigma(\phi) \cup \sigma(\{g_i : T_i\}_{i\in I}) \cup \sigma(msgs(\tilde{\gamma})))) & \text{iff} \\ m : T \in \mathcal{D}(\phi \cup \{g'_i : T'_i\}_{i\in I} \cup msgs(\tilde{\gamma'})) & \end{array}$$

Finally, we have $S'\|_L X'_{\phi\cup\{g'_i:T'_i\}_{i\in I}} \models \exists\gamma : (m : T) \in K^{\phi\cup\{g'_i:T'_i\}_{i\in I}}_{X',\gamma}$. The other direction can be proved by using a symmetric reasoning. □

The next lemma states the correctness of our translation from infinitary partially evaluated formulas, to finitary ones.

Lemma 6.4 *Given a system S and $m_1 : T'$ initial in S then:*

$$\exists X_\phi \models (\exists\gamma : m_1 : T' \in K^\phi_{X,\gamma}//S) \text{ iff } \exists Y_\phi \models (\exists\gamma : m_1 : \tilde{T'} \in K^\phi_{Y,\gamma}//S).$$

Proof:

By induction on the depth of the nesting infinitary disjunctions in $F = \exists\gamma : m_1 : T' \in K^\phi_{X,\gamma}//S$

($\Longrightarrow$)

So if $\exists X_\phi \models F$ then either X models a formula in the finite part of F and in this case also of $\tilde{F}$, or exists $t = (c, m : T, \langle\langle g_i : T_i\rangle\rangle_{i\in I}, S') \in Rsucc(S) : X_\phi \models F_t$, where F_t is

$$\langle\langle \tau_{g_i:T_i}\rangle\rangle_{i\in I}\langle c!m : T\rangle(\exists\gamma : (m_1 : T') \in K^{\phi\cup\langle\langle g_i:T_i\rangle\rangle_{i\in I}}_{X,\gamma}//S')$$

The last fact implies:

$$X_\phi \overset{\tau_{g_1:T_1},\ldots,\tau_{g_n:T_n}}{\longmapsto} X'_{\phi'} \xrightarrow{c!m:T} X''_{\phi'} \wedge X''_{\phi'} \models (\exists\gamma : m_1 : T' \in K^{\phi'}_{X,\gamma}//S_1)$$

with $S \xrightarrow{c?m:T} S_1, \phi' = \phi \cup \{g_1 : T_1, \ldots, g_n : T_n\}, m : T \in \mathcal{D}(\phi')$ and $\{g_i : T_i\}_{i\in I} \subseteq SubM(m : T)$.

There must be the case for some $t' = (c', m' : T', \langle\langle g'_i : T'_i\rangle\rangle_{i\in I'}, S'_1) \in [t]_{\equiv^e}$ that $F_{t'}$ appears as a disjunction in $\tilde{F}$, let us see that under this hypothesis there exists X^1_ϕ s.t. $X^1_\phi \models F_{t'}$.

Let σ be a bijection between the random values of $\langle\langle g_i : T_i\rangle\rangle_{i\in I}$ and $\langle\langle g'_i : T'_i\rangle\rangle_{i\in I'}$, (a bijection must exist by construction, since the two sequences have the same multiset of types). Hence it is possible to construct a process X^1_ϕ such that:

$$X^1_\phi \overset{\tau_{g'_1:T'_1},\ldots,\tau_{g'_n:T'_n}}{\longmapsto} X'^1_{\phi'_1}$$

and $X'^1_{\phi'_1} \equiv_\sigma X'_{\phi'}$. To achieve this aim, let us reorder the second sequence $\langle\langle g'_i : T'_i\rangle\rangle_{i\in I'}$, in such a way that $\sigma(g_j) = g'_j$ with $1 \leq j \leq n$. At this point if $X_\phi \xrightarrow{\tau_{g_1:T_1}} X'_{\phi\cup\{g_1:T_1\}}$, then X must be of the form $gen^{v,(i,j)}_{T_1}.Z$ and $X' = Z[g_1 : T_1/v]$. So we define $X^1 = (gen^{v,(i',j')}_{T_1}.Z)$, where $\mathcal{R}^{T^k}(i', j') = g'_1$. By iterating this n times give us the desired process.

Then for some $X''^1_{\phi'_1}$ it must be $X'^1_{\phi'_1} \xrightarrow{c!m':T} X''^1_{\phi'_1}$. We have $\sigma(S') = S'_1, \sigma(m : T) = (m' : T)$ and $\sigma(\langle\langle g_i : T_i\rangle\rangle_{i\in I}) = \langle\langle g'_i : T'_i\rangle\rangle_{i\in I'}$, so we are in the hypothesis of lemma 6.3, hence $S_1 \| X''_{\phi'} \models \exists\gamma : m_1 : T' \in K^{\phi'}_{X,\gamma}$ and so $S'_1 \|_L X''^1_{\phi'_1} \models \exists\gamma : m_1 : T' \in K^{\phi'_1}_{X,\gamma}$ and by proposition 6.2 this leads to $X''^1_{\phi'_1} \models \exists\gamma : m_1 : T' \in K^{\phi'_1}_{X,\gamma}//S'_1$. The depth of the nesting of infinitary disjunctions in the latter formula is less than in the formula $\exists\gamma : (m_1 : T') \in K^{\phi}_{X,\gamma}//S$, hence by applying the inductive hypothesis $\exists Y'_{\phi'_1} \models \exists\gamma : \widetilde{(m_1 : T')} \in K^{\phi'_1}_{Y',\gamma}//S'_1$, hence the thesis follows by considering $X''_{\phi'_1} = Y'_{\phi'_1}$ and $Y_\phi = X^1_\phi$.

The other direction is trivial.

$\square$

The following is the proof of the main theorem of the chapter.

Theorem 6.1 *Given a system S, with $Sort(S) \subseteq L$, a finite set of typed messages ϕ and an initial message $m : T$ then is decidable if $\exists X_\phi$ with $Sort(X) \subseteq L$ s.t. $S \|_L X_\phi \models \exists\gamma : (m : T) \in K^{\phi}_{X,\gamma}$.*

Proof:

By using proposition 6.2 and lemma 6.4 we can reduce the decidability of the existence of such X_ϕ to a satisfiability problem of the formula $\exists\gamma : \widetilde{(m : T)} \in K^{\phi}_{X,\gamma}//S$. We have only to prove that satisfiability for this kind of formulas is a decidable problem.

By induction in on the max depth of the derivations of S, we prove that the satisfiability $\tilde{F} = \exists\gamma : \widetilde{(m : T)} \in K^{\phi}_{X,\gamma}//S$ is decidable and moreover we can build a model for this formula (if it is satisfiable).

S=Nil then $\exists\gamma : \widetilde{(m : T)} \in K^{\phi}_{X,\gamma}//S$ is equal to **T** or **F**. So in the first case every process is a model in the latter no process is a model.

Inductive step then let us suppose then $\tilde{F}$ is satisfiable. Then there must be the case that either $(m : T) \in K^{\phi}_{X,\epsilon} = \mathbf{T}$ or for some successor S' of S that $\tilde{F'} = \exists\gamma : \widetilde{(m : T)} \in K^{\phi'}_{X,\gamma}//S'$ is satisfiable. In the former case every process is a model. Let us see the latter one. By induction, we can build $X'_{\phi'}$ that satisfies $\tilde{F'}$. By construction $\tilde{F'}$ must appear in some reduction of $\tilde{F}$ (see figure 6.4). Let us analyze the various possibilities:

sending: then $\phi' = \phi$ and by construction we have that $m' : T' \in \mathcal{D}(\phi)$ so let X be $c!m' : T'.X'$. We have that $X_\phi \models \tilde{F}$.

guessing: then exists a tuple $(c, m' : T', \langle\langle g_i : T_i\rangle\rangle_{i\in I}, S')$ such that $\phi' = \phi \cup \{g_i : T_i\}_{i\in I}$. For every $g_i : T_i$ we can choose a guessing action that produces such message so we have $X = gen^{x_1,(i_1,j)}_{T_1} \ldots gen^{x_n,(i_n,j)}_{T_n}.c!m' : T'.X'$, where x_i are variables that do not appear in X'. We have that $X_\phi \models \tilde{F}$.

receiving: then $S \overset{c!m':T'}{\longrightarrow} S'$ and $\phi' = \phi \cup \{m' : T'\}$. Hence let $X = c?(x) : T'.X'$ and x is a variable that does not appear in X' and the result follows.

eaves-dropping: then $S \overset{\tau_{c,m':T'}}{\longrightarrow} S'$ and $\phi' = \phi \cup \{m' : T'\}$. Hence let $X = \chi^{x}_{c,m':T'}.X'$ and x is a variable that does not appear in X' and the result follows.

idling: then $\phi' = \phi$ and let X be X'.

On the other hand, if $m : T \in K^{\phi}_{X,\epsilon} = \mathbf{F}$ and it does not exists a $\tilde{F'}$ that is satisfiable for some S', successor of S we have that $\tilde{F}$ cannot be satisfiable.

$\square$

To prove proposition 6.1, first we have to prove the following technical lemma:

Lemma A.3 *Given a finite set of typed messages ϕ. Then:*

α *If $m : T_1 \times T_2 \notin SubM(\phi)$ then if there exists a proof of $m : T_1 \times T_2$ of depth n then there exists a proof with a depth less or equal n where last rule applied is (2),*

β *If $E(k, m) : E(Key, T) \notin SubM(\phi)$ then if there exists a proof of $E(k, m) : E(Key, T)$ of depth n then there exists a proof with a depth less or equal n where last rule applied is (5).*

Proof: By induction on the depth n of the proof:
[$(n = 1)$]

α There is no proof of this depth since the only possibility is to use axiom (1), and $m : T_1 \times T_2 \notin SubM(\phi)$

β There is no proof of this depth since the only possibility is to use axiom (1), and $E(k, m) : E(Key, T) \notin SubM(\phi)$

[$n = 1 + n'$]

α By inspection of the last rule used in the proof:

rule 2 done,

rule 3 then, there must exist a proof for some message $m, m_2 : (T_1 \times T_2) \times T$. We can apply inductive hypothesis on this message, whose proof must have a depth less n and it cannot appear as submessage of some message in ϕ (otherwise $m : T_1 \times T_2$ should be a submessage of this message). Hence, there is a proof with a depth less then n where the last rule used is (2), but in this case $m : T_1 \times T_2$ must be proved with a proof of depth less then n, and we can apply inductive hypothesis again and this leads to the result,

rule 4 similar to above proof,

rule 5 It is no possible,

rule 6 then, there must exist a proof for some message $E(k, m) : E(Key, T_1 \times T_2)$. We can apply inductive hypothesis on this message, whose proof must have a depth less n and it cannot appear as submessage of some message in ϕ (otherwise $m : T_1 \times T_2$ should be a submessage of this message). Hence, there is a proof with a depth less then n where the last rule used is (5), but in this case $m : T_1 \times T_2$ must be proved with a proof of depth less then n, and we can apply inductive hypothesis again and this leads to the result,

β By inspection of the last rule used in the proof:

rule 2 It is no possible,

rule 3 then, there must exist a proof for a message $m_2, E(k, m) : T \times (E(Key, T))$. We can apply inductive hypothesis on this message, whose proof must have a depth less n and it cannot appear as submessage of some message in ϕ (otherwise $E(k, m) : E(Key, T)$ should be a submessage of this message). Hence, there is a proof with a depth less then n where the last rule used is (2), but in this case $E(k, m) : E(Key, T)$ must be proved with a proof of depth less then n, and we can apply inductive hypothesis again and this leads to the result, rule 4 similar to above proof,

rule 5 done,

rule 6 then, there exist a proof for a message $E(k', E(k, m)) : E(Key, E(Key, T))$. We can apply inductive hypothesis on this message, whose proof must have a depth less n and it cannot appear as submessage of some message in ϕ (otherwise $E(k, m) : E(Key, T)$ should be a submessage of this message). Hence, there is a proof with a depth less then n where the last rule used is (5), but in this case $E(k, m) : E(Key, T)$ must be proved with a proof of depth less then n, and we can apply inductive hypothesis again and this leads to the result.

□

Finally, we can state:

Proposition 6.1 *The deduction function of section 6.4 enjoys assumptions 6.1,6.2,6.3 and 6.4.*

Proof:

By structural induction on the type T and by inspection on the rules of the inference system we prove that function $\mathcal{D}$ enjoys assumption 6.1:

- If T is a basic type then since the submessages of messages in ϕ of type T are finite and the rules do not introduce new message of a basic type that are not submessage of the premises, we have the thesis.

- If $T = T_1 \times T_2$, the possible submessages of type T in ϕ can be in a finite number. If $m : T$ is not a submessage then for lemma A.3 α there is a proof where last rule used is (2), but by inductive hypothesis only a finite number of T_1 and T_2 messages can be deduced and so the thesis follows.

- If $T = E(Key, T_2)$, then the possible submessages of type T in ϕ can be in a finite number. Otherwise for lemma A.3 α there is a proof where last rule used is (5), but by induction hypothesis only a finite number of submessages of type key or T_2 can be deduced and hence the thesis follows.

By lemma A.3 and the fact that ϕ is finite, and so its submessages, we can build a function ψ s.t. the cardinality of $\mathcal{D}(\phi) \cap Msgs(T)$ is bounded from $\psi(\phi)$ and this leads to constructability, in our sense, of $\mathcal{D}(\phi) \cap Msgs(T)$.

Now we prove that $\mathcal{D}$ enjoys assumption 6.3, by induction on the max depth of the branches of the proof of $m : T \in \mathcal{D}(\phi \cup \{g : T'\})$. If the proof has a depth of 1 then is an axiom, and the result trivially follows. Let us analyze the inductive step. By inspection on the last rule used in the derivation:

Rule 2 then $m : T = (m_1, m_2) : T_1 \times T_2$ and we can apply inductive hypothesis on these two messages and we have that $m_i : T_i \in \mathcal{D}(\phi), i = 1, 2$ and so by apllying rule (2) $m : T \in \mathcal{D}(\phi)$,

Rule 3 then exists $m, m_1 : T \times T_1 \in \mathcal{D}(\phi \cup \{g : T'\})$. If $m, m_1 : T \times T_1$ is not in $\phi \cup \{g : T'\}$ as submessage then for lemma A.3 α there must be a shorter or equal proof of $m, m_1 : T \times T_1$ with last rule used is 2, applying inductive hypothesis the result follows. Otherwise we can apply directly inductive hypothesis since $g_i : T^i$ does not appear as submessage in $m, m_1 : T \times T_1$, and then apply rule 3.

Rule 4 analogous to the previous one,

Rule 5 then $m : T = E(k, m_1) : E(Key, T_1)$ and we can apply inductive hypothesis on $m_1 : T_1$ and $k : Key$ and so we have that $m_1 : T_1 \in \mathcal{D}(\phi)$ and $k : Key \in \mathcal{D}(\phi)$ hence by applying rule (5) we have $m : T \in \mathcal{D}(\phi)$,

Rule 6 then exists $E(k, m) : E(key, T)$ that is provable from $\mathcal{D}(\phi \cup \{g : T'\})$. If $E(k, m) : E(key, T)$ is not in $\phi \cup \{g : T'\}$ as submessage then for lemma A.3 β there must be a shorter or equal proof of $E(k, m) : E(key, T)$ where last rule uses is 5 so we can apply inductive hypothesis and we have the thesis. Otherwise we can apply directly inductive hypothesis since $g : T'$ does not appear as submessage in $E(k, m) : E(key, T)$, and then apply rule 6, since $k^{-1} : (-1)Key$ must be in $\mathcal{D}(\phi)$.

The proof for assumption 6.2 is done noticing that the rule schemata are based on metavariables for messages, so they are invariant for renaming or bijection between random messages.

The proof for assumption 6.4 can be done by noticing that the rules of the inference system do not introduce in the consequence random values that are not in the premises, hence the result follows.

□

Proposition A.1 *Assumption 6.4 implies assumption 6.3*

Proof: We prove $m : T \in \mathcal{D}(\phi \cup \{g : T^i\}) \Longrightarrow m : T \in \mathcal{D}(\phi)$ by induction on the max depth of the branches of the proof of $m : T \in \mathcal{D}(\phi \cup \{g : T^i\})$.

If the depth is 1 then the rule used is an axiom and since it follows that $m : T \in \phi$ so $m : T \in \mathcal{D}(\phi)$. The inductive case is analyzed by considering the fact that the premises must have a proof shorter then $m : T$ so we can apply on these the inductive hypothesis and so also these premises can be proved by starting from ϕ and the result follows.

□

Proposition 6.4 *Given $\mathcal{D}(\phi)$ with ϕ* base, *if ψ is a* base *for $\mathcal{D}(\phi)$ then we have $\psi = \phi$.*

Proof: We show $\psi \subseteq \phi$. By contradiction consider a message $m : T \in \psi$ and $m : T \notin \phi$, whose type has the smallest size among the messages in ψ that are not in ϕ.

Now $m : T \in \psi$, so $m : T$ is in $\mathcal{D}(\psi) = \mathcal{D}(\phi)$. Since ϕ is downward closed, there must exist a set of typed messages $\{m_1 : T_1, \ldots, m_n : T_n\} = \phi \cap Msg(|T| - 1)$ s.t. $m : T \in \mathcal{D}(\{m_1 : T_1, \ldots, m_n : T_n\})$. For every $i \in \{1, \ldots n\}$ the size we have $m_i : T_i \in \mathcal{D}(\phi) = \mathcal{D}(\psi)$.

Thus we can prove that for every $i \in \{1, \ldots n\}$ the message $m_i : T_i$ is in ψ. Suppose the contrary, and let $m_j : T_j, j \in \{1, \ldots n\}$ be a message whose type has the smallest size w.r.t. the ones that are not in ψ. Then since ψ is DC then there exists a set of messages in ψ, whose size is smaller than the size of T_j, from which $m_j : T_j$ is deducible. This means that these messages are in ϕ and hence are among the messages $\{m_1 : T_1, \ldots, m_n : T_n\}$. But this leads to a contradiction since in this case we have that ϕ is not minimal!. Hence messages $\{m_1 : T_1, \ldots, m_n : T_n\}$ are in ψ, but in this case also ψ is not minimal, in contrast with the hypothesis that ψ is base.

The other inclusion can be proved by using a symmetric reasoning.

□

Appendix B

A verification session

In this appendix we present a simple verification session for the example of subsection 6.10.1, with the tool we have developed for cryptographic protocols analysis.

The code for a sender agent is the following:

```
Send(ab, Encrypt((na:Nonce, a:Aid), bkey:EKey)).
Recv(ab, XA : Enc(Nonce*Nonce)).
If Deduce (YA = Decrypt(XA, akey:DKey)) Then
  If Deduce (NAa = Fst(YA)) Then
    If (NAa = na : Nonce) Then
      If Deduce (NAb = Snd(YA)) Then
        Send(ab,Encrypt(NAb,bkey:EKey)).0
      End Deduce
    End If
  End Deduce
End Deduce
```

The code for a receiver agent is the following:

```
Recv(ab, Z : Enc(Nonce*Aid)).
If Deduce (X = Decrypt(Z, bkey : DKey)) Then
  If Deduce (A = Snd(X)) Then
    If (A = a : Aid)  Then
      If Deduce (Na = Fst(X)) Then
        Send(ab, Encrypt((Na,nb : Nonce), akey : EKey)).
        Recv(ab, V:Enc(Nonce)).
         If Deduce (Vb = Decrypt(V, bkey : DKey)) Then
          If (Vb = nb : Nonce) Then
           0
          End If
         End Deduce
      End Deduce
      End If
    End Deduce
  End Deduce
```

Analisys of the flawed version:

```
# load_spec "nspk.spec";;
Spec Loaded- : unit = ()
# set_secret "nb : Nonce";;
- : unit = ()
# set_base [" akey : EKey" ; "bkey : EKey"; "xkey : EKey";
          "xkey : DKey" ; "a : Aid"; "b : Aid"; "x :   Aid"];;
- : unit = ()
# run_spec();;
- : bool = true
# show_intruder ();;
- : Terms.action list =
[Terms.Recv ("ax", "E[xkey]((na,a)) : Enc[(Nonce*Aid)]",
             Enc[(Nonce*Aid)]);
 Terms.Send ("ab", E[bkey]((na,a)) : Enc[(Nonce*Aid)]);
 Terms.Recv ("ab", "E[akey]((na,nb)) : Enc[(Nonce*Nonce)]",
             Enc[(Nonce*Nonce)]);
 Terms.Send ("ax", E[akey]((na,nb)) : Enc[(Nonce*Nonce)]);
 Terms.Send ("ab", E[akey](na) : Enc[Nonce]);
 Terms.Recv ("ax", "E[xkey](nb) : Enc[Nonce]", Enc[Nonce])]
```

Analysis of the corrected version:

```
 load_spec "nspk_ok.spec";;
Spec Loaded- : unit = ()
# set_secret "nb : Nonce";;
- : unit = ()
# set_base [" akey : EKey" ; "bkey : EKey"; "xkey : EKey";
          "xkey : DKey" ; "a : Aid"; "b : Aid"; "x :   Aid"];;
- : unit = ()
# run_spec();;
- : bool = false
```