Università degli Studi di Pisa
Dipartimento di Informatica

Dottorato di Ricerca in Informatica

Ph.D. Thesis: TD-4/06

Probabilistic Timed Automata for Security Analysis and Design

Angelo Troina

Supervisor
Prof. Andrea Maggiolo Schettini

Referee
Prof. Catuscia Palamidessi

Referee
Prof. Christel Baier

May 2006

Dipartimento di Informatica
Largo B. Pontecorvo 3, 56127 - Pisa

Acknowledgments

In questi tre anni di dottorato Andrea Maggiolo è stato una guida costantemente presente. I suoi incoraggiamenti, i suoi consigli e la sua amicizia sono sempre stati preziosi. Andrea, grazie per quanto hai fatto in questi anni, è veramente un piacere lavorare con te.

Il supporto e il contributo di Ruggero Lanotte sono stati fondamentali per la stesura di questa tesi. Senza i suoi consigli e "teoremi" probabilmente non sarebbe stata la stessa.

Ringrazio poi Roberto Barbuti e Paolo Milazzo per avere stimolato il mio interesse verso la biologia dei sistemi, e per avermi coivolto in una collaborazione che si è sempre rivelata estremamente piacevole.

Un pensiero va anche al Tea Club e a tutti i pomeriggi sfiziosi trascorsi insieme a Susanna, Eugenia, Francesca, Marco e Giorgio.

Difficilmente potrò mai ricompensare Franco e Rosa per quanto hanno fatto e continuano a fare per me; ai miei genitori esprimo tutta la mia riconoscenza.

Un abbraccio ai miei compagni di questi anni trascorsi a Pisa...
...Carlo, Sascha, non abbattete il muro senza di me!

Last but not least, I wish to thank Prof. Catuscia Palamidessi and Prof. Christel Bayer who reviewed this thesis and gave me many useful suggestions.

Abstract

The usefulness of formal methods for the description and verification of complex systems is nowadays widely accepted. While some system properties can be studied in a non-timed and non-probabilistic setting, others, such as quantitative security properties, system performance and reliability properties, require a timed and probabilistic description of the system. This thesis focuses on methods for the formal modeling of probabilistic timed systems, and on algorithms for the automated verification of their properties. The models considered describe the behavior of a system in terms of time and probability, and the formal description languages used are based on extensions of Timed Automata, Markov Decision Processes and combinations of them.

In multilevel systems it is important to avoid unwanted indirect information flow from higher levels to lower levels, namely the so called covert channels. Initial studies of information flow analysis were performed by abstracting away from time and probability. It is already known that systems which are considered to be secure may turn out to be insecure when time or probability are considered. Recently, work has been done in order to consider also aspects either of time or of probability, but not both. In this thesis, a general framework is proposed, which is based on Probabilistic Timed Automata, where both probabilistic and timing covert channels can be studied. A concept of weak bisimulation for Probabilistic Timed Automata is given, together with an algorithm to decide it. Such an equivalence relation is necessary to define information flow security properties. Thus, a Non Interference security property and a Non Deducibility on Composition security property are given. They allow expressing information flow in a timed and probabilistic setting, and they can be compared with analogous properties defined in settings where either time or probability or none of them are taken into account. This permits a classification of the properties depending on their discriminating power.

Some new aspects are then explored involving the introduction of parameters in Markov Decision Processes and the use of Timed Automata in the study of cryptographic protocols. Finally, some real-life applications are described through automata-based formalisms, and model checking is used to study security issues affected by aspects of time and probability.

Contents

Introduction

Complex systems may exhibit behaviour depending on time and probability. While some systems properties can be studied in non-timed and non-probabilistic settings, others require that system description methodologies and verification tools take these aspects into account. This is the case when studying quantitative security properties or performance and reliability properties.

System design aims at ensuring that some properties hold for system processes. Some systems can be designed in such a way that the wanted properties are guaranteed to hold, but this, in general, cannot be achieved for complex systems, either because it would be impractically expensive or because they include unreliable components. Hence, the need arises of languages for formal specifications and techniques to verify that the specified properties hold. Verification techniques based on testing and simulation can miss errors when the number of states of the system is very large. Theorem provers and proof checkers do not have this shortcoming, but they are time consuming and often require manual intervention. An alternative verification technique, called model checking, is based on search procedures which determine if the specification, expressed in a certain logic, is true of the representation of the behaviour of the system.

This thesis focuses on methods for the formal modeling and analysis of timed and probabilistic systems, and on algorithms for the automated verification of their properties. In particular, security properties must be precisely defined in order to prove that the system design fulfills the wanted requirements. In the more general usage, security means controlling the way information may flow among different entities, and security properties one wants to hold are information flow properties.

The formal description languages we consider are based on extensions of Markov decision processes [20, 71] and Timed Automata [9].

On the one hand, probabilistic models are nowadays widely used in the design and verification of complex systems in order to quantify unreliable or unpredictable behaviour in security, performance and reliability analysis [39, 14, 141, 5]. The analysis process consists in building a probabilistic model of the systems, typically a Markov Decision Process [20, 71] or a Continuous Time Markov Chain on which analytical, simulation based and numerical calculations can be performed to obtain the desired quantitative measures.

On the other hand, the importance of real time considerations in the functioning of systems has suggested to augment finite automata with timing constraints and to extend logics accordingly [9]. Within such a model, different properties have been investigated (e.g. expressiveness of the model) both with discrete and dense time assumptions (see also [67]). To describe more general situations and model concurrent systems, different extensions have been proposed (see [10, 47, 91]).

In 1963, Rabin introduced Probabilistic Automata [127], closely related to Markov Decision Processes. There are several extensions of Probabilistic Automata with time variables and constraints, or extensions of Timed Automata with probabilistic choices. Some models of Probabilistic Timed Automata can be found, e.g., in [8, 39, 85, 18, 96]. Temporal logics have been adapted in order to deal with quantitative verification, namely with the problem of determining the probability with which a system satisfies a specification.

In this thesis we introduce extensions of the model of Probabilistic Timed Automata in a context of security analysis by defining a notion of weak bisimulation for Probabilistic Timed Automata and developing an algorithm to decide it [94, 100]. Once given such a notion, it is easy to extend security analysis techniques based on the Non-Interference theory [54] within the model of Probabilistic Timed Automata (see, i.e. [96, 98]). Other well known information flow security properties may be formulated within our framework. We also investigate an extension of the Timed Automata model dealing with data structures in order to represent computation of cryptographic primitives and then analyze security protocols. Interesting applications may also derive by considering probabilities as parameters.

In conclusion, we aim at offering a general uniform model for probabilistic timed automata able to describe a large class of systems and to analyze properties of different kinds. In fact, automata based models offer a rigorous and precise modeling technique, and, at the same time, such descriptions can be immediately transformed as input for model checkers.

I.1 Concepts and Models of Security

Our society is becoming more and more dependent on computer networks. The enormous amount of data that is processed, transmitted and stored needs some form of protection. Cryptography has been envisaged as the main practical means to protect information transmitted on communication networks. A long tradition of studies on cryptographic algorithms validates the use of these techniques. However, nowadays cryptography is mainly used as a building block of many complex applications where the correctness of the cryptographic algorithm is not, *per se*, a guarantee of the correctness of the applications. Indeed, cryptographic-based procedures are largely used for authentication of messages, personal identification, digital signatures, electronic money transfer, credit card transactions and many other critical applications. Recent technological developments, with the introduction of mobile code, mobile agents and Java applets have raised a supplementary increasing demand for security in computer networks. The main issues, in this sense, are related to protection of the host resources from unwanted damage caused by imported, possibly malicious, code. Browser-based applications are critically dependent on the security of the applets they are using.

Surprisingly enough, such a pervasive phenomenon has not been accompanied by correspondingly widespread development and use of formal tools to help analysts and designers to describe faithfully, to analyze in detail, and to prove the correctness of such systems. This is mainly due to a lack of theoretical understanding of the phenomena. On the one hand, even when formal techniques are available, they are generally perceived as difficult to learn and apply: in other words, they are not seen as cost-effective. On the other hand, the increasing number of reports of security flaws in software show that ignoring the problem of rigorous correctness proofs is at our own risk. Remarkable examples range from academic cryptographic protocols, such as the Denning-Sacco key distribution protocol (with public key, 1981) [42], which was believed to be correct for several years until shown to be flawed by Abadi (1994) [2], to industrial applications, such as the programming language Java, whose type flaws were at the core of a number of security holes. Many of these flaws could conceivably have been prevented by the use of a careful formal design and analysis.

The detection and the prevention of flaws is indeed one of the main motivations for using formal methods: formal system specification is an essential prerequisite for analysis, that may help detect many design flaws. Furthermore, if the specification is given in an executable language, then it may also be exploited to simulate the execution of the system, helping the verification of properties (early prototyping). Typically, other motivations for the use of formal specifications include the need for expressing unambiguous user requirements, for producing a reference guide for the implementor of the real system during the various development phases. Eventually, systems will be certified to be free of flaws.

I.1.1 Information Flow Security

Distributed systems have resources and data shared among users located everywhere in the world. Security is a major issue. By security one means a number of requirements, such as *confidentiality*, *availability* and *integrity*, to be satisfied by the system.

The term *security model* is used in the literature (e.g. see [105]) to mean the definition of a mechanism for enforcing a security property. A security model imposes restrictions on a system interface (usually input/output relations) that are sufficient to ensure that any implementation satisfying these restrictions will enforce the property.

In this thesis we are interested in confidentiality. Intuitively, confidentiality states the complete absence of any unauthorized disclosure of information. A mechanism for confidentiality, called *access control*, was first formulated by Lampson [89] and later refined by Graham and Denning [56]. The structure of the model is that of a state machine where each state is a triple (S, O, M) with S a set of subjects (active agents, such as users), O a set of objects (data, such as files) and M is an access matrix which as one row for each subject, one column for each object and is such that the entry $M[s, o]$ contains the access right subject s has for object o.

Harrison, Ruzzo and Ullman [63] use Lampson's concept of an access control model to analyze the complexity of determining the effects of a particular access control policy they propose. It turns out that it is often hard to predict how access rights can propagate in a given access control model. A user is often unaware that executing certain programs (*Trojan Horses*) he will pass to another user some entirely unrelated set of rights he possesses. This has lead to the formulation of two distinct access control policies.

In classic *Discretionary Access Control* security (DAC), every subject decides the access properties of his objects. The fact that users decides the access rights on their files gives flexibility, but also facilitates security leakages.

In *Mandatory Access Control security* (MAC), some access rules are imposed by the system and Trojan Horse programs may have a limited effect. An example of MAC is *multilevel security* proposed by Bell and Lapadula [19]. In such a model, every subject is bound in a security level, and so is every object. Information may flow from a certain object to a certain subject only if the level of the subject is greater than the level of the object. A Trojan Horse action is, hence, restricted to a level and has no way of downgrading information. With MAC, however, it could still be possible to transmit information indirectly by means of system sides effect.

For example, consider a low level user and a high level user sharing a finite storage resource (e.g., a buffer). It may be possible for the high level user to transmit information indirectly by completely filling the shared buffer. If the high level user wants to transmit the bit 1 it simply fills the buffer; in this case, since the buffer is full, a low write action returns an error message and the low user deduces the transmission of bit 1. Otherwise, if the low write action is successful, the low level user deduces that the high level user is transmitting the bit 0. Such indirect transmission, which does not violate access rules, is called a *covert channel* [88].

The existence of covert channels has led to the more general approach of *information flow security*, which aims at controlling the way information may flow among different entities. The idea is to try to directly control the whole flow of information, rather than only the direct communication among agents. In [54] the notion of *Non Interference* is introduced, stating, intuitively, that low level agents should not be able to deduce anything about the activity of high level agents. By imposing some information flow rules, it is possible to control direct and indirect leakages, as both of them give rise to unwanted information flows.

Other properties have been introduced in the literature in order to capture different behaviour of systems that has to be considered not secure. One of the most interesting and intuitive security properties is the *Non Deducibility on Composition* (*NDC*) [142, 51], which states that what a low level user observes of the system in isolation is not altered when considering all the potential interaction of the system with any high level agent of the external environment.

The problem of formalizing the notion of confidentiality boils down to that of formalizing the equivalence of processes [51, 135]. The latter is a central and difficult question at the heart of computer science to which there is no unique answer. Which notion of equivalence is appropriate depends on the context and application. Consequently, we should not be surprised that the information security community has failed to come up with a consensus on what constitutes confidentiality. In [51] weak bisimulation equivalence has been shown to be one of the most reasonable equivalence relation. Fine enough to capture any unsecure behavior that can give rise to information flow, but not so strict to classify as unsecure behaviour which are instead correct. Hence, we also resort to weak bisimulation in order to define information flow security properties within our framework.

Definitions of information flow security properties have been formulated in various system description formalisms (see, e.g., [54, 104, 57, 51, 52, 50, 17, 6]).

The problem of composability of multi-level security properties is discussed in [104]. It is shown that some security properties do not compose and that it is possible to connect two systems, both of which are judged to be secure, such that the composite system is not secure. In [51] Focardi and Gorrieri promote the classification of a set of properties capturing the idea of information flow. The authors use a slight extension of Milner's CCS [111], called Security Process Algebra (SPA), to specify and analyze security properties of concurrent systems. In SPA, the set of actions is partitioned into high level actions and low level ones in order to specify multilevel systems. Security properties are classified according to the equivalence relations used for modeling indistinguishable observational behaviour.

Most of the properties considered are based on analysis of information flow that does not take into consideration aspects of time or of probability, and therefore these properties are not useful to check the existence of probabilistic or timing covert channels. To overcome this, in the recent years, a significant work has been done in order to extend the study by considering either probability (see, e.g., [57, 6, 123, 5]) or time (see, e.g., [52, 50, 17]).

In [57] Gray describes a general purpose, probabilistic state machine model which can be used to model a large class of computer systems. Information flow probabilistic properties are developed. In [6, 5] a probabilistic process algebra is used to capture security leakages which may arise by observing the frequencies of performance of certain actions. A notion of probabilistic observational equivalence is given when formalizing the probabilistic security properties. In [123] Di Pierro, Hankin and Wiklicky present a model in which the notion of Non Interference is approximated in the sense that it allows for some exactly quantified leakage of information.

In [52] a CCS-like timed process algebra is used to formalize time-dependent information flow security properties. In [50] Evans and Schneider detail an approach to verifying time dependent security properties. Time is introduced into the Communicating Sequential Processes (CSP) verification framework. In [17] Barbuti and Tesei propose the formalization of a Non Interference property in a timed setting within the model of Timed Automata.

By the best of our knowledge, there are no studies of information flow security where time and probability are considered together.

I.1.2 Security Protocols

Security protocols are a critical element of the infrastructures needed for secure communication and processing information. Most security protocols are extremely simple if only their length is considered. However, the properties they are supposed to ensure are extremely subtle, and therefore it is hard to get protocols correct just by informal reasoning. The history of security protocols (and also cryptography) has a lot of examples, where weaknesses of supposedly correct protocols or algorithms were discovered even years later. Thus, security protocols are excellent candidates for rigorous formal analysis. They are critical components of distributed security, are very easy to express and very difficult to evaluate by hand.

Many security properties may be defined in order to analyze cryptographic protocols (among them, *secrecy*, *authentication*, *integrity*, *non-repudiation*, *anonymity*, *fairness*, etc.).

Secrecy has a number of different possible meanings; the designer of an application must decide which one is appropriate. The strongest interpretation would be that an intruder is not able to learn anything about any communication between two participants of a system by observing or even tampering the communication lines. That is, he cannot deduce the contents of messages, sender and receivers, the message length, the time they were sent, and not even the fact that a message was sent in the first place. In theory, this "perfect" secrecy can be approximated quite close by exploiting todays cryptographic tools: encryption and digital signatures, dummy traffic to keep a constant network load, anonymizing proxies to "shuffle" packets across the nodes to make routing analysis infeasible, and a huge calculation overhead on the message recipients, since decryption and signature checking must be performed with each possible key (since there is no sender or receiver name attached). Hence, for most practical applications this would neither be efficient nor necessary. A very weak interpretation, which is used in todays encrypted email communication, confines secrecy to the actual content of messages. This may be the most important part, but especially for very small messages it must be taken into consideration that an attacker learns something about the exact (if a stream cipher is used) or approximate (with a block cipher) length of the message. Additionally, depending on the controlled network area attackers can log a partial or complete traffic analysis. But this approach can be performed with no useless overhead. These two extreme situations point out that the designer of a distributed system must analyze exactly which attributes of communication have to be concealed and which can go unprotected to allow a more efficient implementation.

A system provides strong authentication if the following property is satisfied: if a recipient R receives a message claiming to be from a specific sender S then S has sent exactly this message to R. For most applications this formulation must be weakened, since in most cases communication channels are subject to both technical errors and tampering by attackers. A system provides weak authentication if the following property is satisfied: if a recipient R receives a message claiming to be from a specific sender S then either S has sent exactly this message to R or R unconditionally notices this is not the case. Some authors make a distinction between the two aspects of authentication: a validated sender name is referred to as "authentication of origin" and the fact that the message has not been altered in any way is called integrity. But neither property alone increases the confidence to a message, so both must always be present. An additional property authentication systems can have is non-repudiation. This property states that a recipient is not only confident that the message is authentic (sent by S and unmodified) but can also prove this fact. In analogy to handwriting signatures these systems are called digital signature systems.

A system that is anonymous over a set of users has the following property: when a user sends a message, then any observer will be unable to identify the user, although he might be able to detect the fact that a message was sent. Examples are an open discussion board or the prevention of traffic analysis on web servers. Sometimes it is not intended that the messages can be sent completely independent from each other. In an electronic voting protocol, the property shall be constrained: every user can send exactly one message anonymously, but the protocol must ensure that double votes can not occur unnoticed. Either it is constructed if it is impossible to vote twice or the offending users identity is revealed in such a case.

An application where fairness properties get important is electronic contract signing. A protocol that achieves this must secure that the process of applying all contractor signatures is transactional. This means that the third contractor must not be able to halt the protocol if the first two signatures are valid. There are several protocols available which have to trade off the need of a trusted third party against a lot of communications for incremental commitment.

The use of formal methods for modeling and analyzing cryptographic operations is now well-established. Since the seminal paper by Dolev and Yao [46] introduced a simple and intuitive description for cryptographic protocols, many alternative definitions have been proposed on the basis of several approaches, ranging from modal logics to process algebras (see, e.g., [40, 109, 79, 57, 137, 119, 48, 136]).

Security protocols, like distributed programs in general, are sensitive to the passage of time; however, the role of time in the analysis of cryptographic protocols has only recently received some attention (see [55, 117, 33, 99]).

Also aspects of probability are taken into account when analyzing quantitative security properties (measuring, in this sense, the security level of the protocol) or when dealing with probabilistic protocols (see [5, 6, 7, 21, 97, 118, 112]).

I.2 Modeling Formalisms

A formal approach must offer languages to describe systems. Many description formalisms have been proposed. *Process algebras* [69, 110] are an algebraic framework with operators of choice and parallelism, in which process equality can be specified by algebraic laws. *Petri nets* [130] are transition systems whose dynamics are described by the passage of tokens. UML [134], which subsumes previous *Statecharts* [62] and *Message sequence charts*, is a syntactically rich graphical language. All these formalisms, describing, in general, nondeterministic behaviour, have been extended with aspects of probability, time and parameters.

Nondeterminism usually represents uncertainty about the step that will be taken from a state among a set of possible ones (e.g. unknown scheduling mechanisms). Probability allows the modeling of systems with unreliable or probabilistic behaviour, and becomes essential in the analysis of reliability and performance properties. The correct functioning of systems which interact with physical processes depends crucially upon *real-time* considerations. Parameters are used to describe a class of systems with the same structures but with different constants. They are useful to describe an abstract system in which some values are not available at the moment or to study for which values of parameters a system satisfies some properties.

Models based on the concept of states and transitions, also called *automata*, have turned out to be particularly intuitive. States of an automaton represent snapshots of the described system, while transitions represent state changes.

The notion of finite automaton was developed in the fifties with neuron nets and switching circuits in mind. Later, finite automata have served as useful tool in the design of lexical analyzers ([70]). The subject of finite automata on infinite sequences and infinite trees was established in the sixties by Büchi [25], Mc Naughton [106] and Rabin [128].

Markov Decision Processes were proposed by Bellman [20] and Howard [71] adding probability distributions to transitions. In 1963, Rabin introduced *Probabilistic Automata* [127], closely related to Markov Decision Processes.

Timed Automata were introduced in the nineties by Alur and Dill [9] as an extension of Büchi's ω-Automata to describe real-time systems. Timed Automata are equipped with variables measuring time, called *clocks*. Transitions are guarded by *clock constraints*, which compare the value of a clock with some constant, and by *reset updates*, which reset a clock to the initial value 0. In general, timed automata models have an infinite state space, however, this infinite state space can be mapped to an automaton with a finite number of equivalence classes (regions) as states.

Systems of communicating agents can be described by automata composed in parallel and sharing synchronization channels. Transitions labeled with a complementing channel name can be taken at the same moment and data transmission is typically modeled by a synchronization, where global variables are updated ([33]).

There are several extensions of Probabilistic Automata with time variables and constraints, or extensions of Timed Automata with probabilistic choices. Some models of Probabilistic Timed Automata can be found, e.g., in [8, 39, 85, 87, 18, 96].

I.3 Specification Languages and Model Checking

System design aims at ensuring that some properties hold for system processes. Some systems can be designed in such a way that the wanted properties are guaranteed to hold, but this, in general, cannot be achieved for complex systems, either because it would be impractically expensive or because they include unreliable components. Hence, the need arises of languages for formal specification of properties and of techniques to verify that the specified properties hold.

Temporal logics have proved to be useful for specifying concurrent systems, because they can describe the ordering of events in time without introducing time explicitly. They were originally developed by philosophers for investigating the way time is used in natural language arguments [72]. Pnueli [124] was the first to use temporal logic for reasoning about concurrency. His approach consisted in proving properties of the program under consideration from a set of axioms that described the behavior of the individual statements in the program. Since proofs were constructed by hand, the technique was often difficult to use in practice.

In general, in temporal logics, time is not mentioned explicitly; instead, a formula might specify that *eventually* some designed state is reached, or that an error state is *never* entered. Properties like *eventually* or *never* are specified using special *temporal operators*. These operators can also be combined with boolean connectives or nested arbitrarily. Temporal logics differ in the operators that they provide and in the semantics of these operators.

Among the many temporal logics proposed in the literature we mention the Computation Tree Logic CTL* (see [31]) with its two sublogics: a *branching-time* logic CTL and a *linear-time* logic LTL (see [49]).

Verification techniques based on testing and simulation can miss errors when the number of states of the system is very large. Theorem provers and proof checkers do not have this shortcoming, but are time consuming and often require manual intervention. In the 1980s, an alternative verification technique, called *temporal logic model checking* was developed independently by Clarke and Emerson [31] and by Queille and Sifakis [126]. In this approach, specifications are expressed in a propositional temporal logic, while circuit designs and protocols are modeled as state-transition systems. The verification technique is based on search procedures which determine if the specification is true of the transition system.

Model checking has several important advantages over mechanical theorem provers or proof checkers. The most important is that the procedure is completely automatic. Typically, the user provides a high level representation of the model and the specification to be checked. The model checking algorithm will either terminate with the answer *true*, indicating that the model satisfies the specification, or give a counterexample execution that shows why the formula is not satisfied. Counterexamples are particularly important in finding subtle errors in complex transition systems, and may therefore be used as a debugging instrument.

For a detailed survey on model checking see [32].

Model Checking with Probability and Time

Model checking has rapidly become a well-established method to analyze and debug complex systems. The first extension of model checking algorithms with probability was proposed in the eighties [64, 147], originally focusing on qualitative probabilistic temporal properties (i.e. those satisfied with probability 1 or 0), but later also introducing quantitative properties [34]. Probabilistic model checking combines probabilistic analysis and conventional model checking in a single tool. It provides an alternative to simulation (which is time consuming) and to analytical approaches (which do not represent systems at the desired level of detail), while, at the same time, offering the full benefits of temporal logic model checking. However, work on implementation and tools did not begin until recent years [60, 65], when the field of model checking matured. In the last few years, several probabilistic model checkers were developed. Among them we quote (PRISM [84, 125] and FHP-Murφ [121]).

PRISM [125] is a probabilistic model checker that allows modeling and analyzing systems which exhibit a probabilistic behavior. Given a description of the system to be modeled, PRISM constructs a probabilistic model that can be either a *Discrete-Time Markov Chain* (DTMC), a *Markov Decision Process* (MDP), or a *Continuous-Time Markov Chain*(CTMC) [83]. On the constructed model PRISM can check properties specified by using probabilistic temporal-logics (PCTL for DTMCs and MDPs, and CSL for CTMCs).

FHP-Murφ (*Finite Horizon Probabilistic Murφ*) [121, 122, 27] is a modified version of the Murφ verifier [45, 116]. FHP-Murφ allows us to define *Finite State/Discrete Time Markov Chains* and to automatically verify that the probability of reaching a given error state in *at most k* steps is below a given *threshold*.

In general, timed automata models have an infinite state space. However, the region automaton construction (see [9]) shows that this infinite state space can be mapped to an automaton with a finite number of equivalence classes (regions) as states, and finite-state model checking techniques can be applied to the reduced, finite region automata. Among the model checkers for timed automata we quote Kronos ([149]) and UPPAAL ([12]).

I.4 Summary

The thesis consists of three main parts.

Part I

In the first part we deal with information flow security properties in frameworks where aspects of probability and time are taken into account.

We start with some preliminary definition and results already known in the literature. In Chapter 1 we formally recall the models of Labeled Transition Systems, Probabilistic Automata and Timed Automata, while in Chapter 2 we introduce the model of Probabilistic Timed Automata. In these chapters we also present weak bisimulation relations for the different models. Thus, goal of this chapters is to define the background and the basic notions needed for the formalization of the novel results presented in the consequent chapters.

In Chapter 3 we give an algorithm to decide weak bisimulation for Probabilistic Timed Automata. A preliminary version of this algorithm has been presented in [94].

In Chapter 4 we propose a general framework, based on Probabilistic Timed Automata, where both *probabilistic* and *timing covert channels* can be studied. We define a Non Interference security property and a Non Deducibility on Composition security property, which allow for expressing information flow in a timed and probabilistic setting. We compare these properties with analogous properties defined in settings where either time or probability or none of them are taken into account. This permits a classification of the properties depending on their discriminating power. As an application, we study a system with covert channels that we are able to discover by applying our techniques. Preliminary results of the study presented in this chapter can be found in [96, 98].

Part II

In the second part we enrich the models of Markov Processes and of Timed Automata with parameters and data structures in order to deal with different classes of problems.

In Chapter 5 we develop a model of Parametric Probabilistic Transition Systems, where probabilities associated with transitions may be parameters. We show techniques to find instances of parameters that satisfy a given property and instances that either maximize or minimize the probability of reaching a certain state. As an application, we model a probabilistic non repudiation protocol with a Parametric Probabilistic Transition System. The theory we develop, allows us to find instances that maximize the probability that the protocol ends in a fair state (no participant has an advantage over the others). Results presented in this chapter appeared in [95].

Systems of Data Management Timed Automata (SDMTAs) are networks of communicating timed automata with structures to store messages and functions to manipulate them. In Chapter 6 we prove the decidability of reachability for this model and we show how it can be used to specify cryptographic communication protocols. Properties of secrecy and authentication are formulated within this framework. As an application, we model and analyze the Yahalom protocol. A preliminary study about SDMTAs has been presented in [99].

Part III

In the third part we describe real-life applications with automata-based formalisms and we study security properties by using probabilistic model checking.

In Chapter 7 we define a probabilistic model for the analysis of a Non-Repudiation protocol that guarantees fairness, without resorting to a trusted third party, by means of a probabilistic algorithm. We describe the protocol with Probabilistic Timed Automata. Then, by using the PRISM model checker, we estimate the probability for a malicious user to break the non-repudiation property, depending on various parameters of the protocol. Work in this chapter appeared in [97].

The *NRL Pump* protocol defines a multilevel secure component whose goal is to minimize leaks of information from high level systems to lower level systems, without degrading average time performances. In Chapter 8 we define a probabilistic model for the NRL Pump and show how the FHP-Murφ probabilistic model checker can be used to estimate the capacity of a probabilistic covert channel in the NRL Pump. We are able to compute the probability of a security violation as a function of time for various configurations of the system parameters (e.g. buffer sizes, moving average size, etc). Because of the model complexity, our results cannot be obtained by using an analytical approach and, because of the low probabilities involved, it can be hard to obtain them using a simulator. Preliminary results have been presented in [92, 93].

Part I

Information Flow Analysis

Chapter 1

Basic Notions

The models used in this thesis are all based on *Labeled Transition Systems* (LTSs), also called *automata*. These have turned out to be an intuitive and powerful framework for the analysis of concurrent systems, and they have been extended with probabilities and time.

Different types of LTSs give rise to different notions of external behaviour. Informally, the *external behavior* of an LTS, also called *visible behavior*, is given by its sequences of external actions. A special invisible action τ is considered. The external behavior of a timed LTS (Timed Automaton) also considers the passage of time as an external action. In probabilistic LTSs (Probabilistic Automata), the probability of performing each action is taken into account, so that a probabilistic measure is associated to each sequence of actions.

We may show that the external behavior of an automaton is contained in the external behavior of another one by proving behavior inclusion. However, this is a rather complex task. In this cases, simulation and bisimulation relations can be extremely useful. These relations compare the stepwise behavior of systems and when two systems are shown to be bisimilar, then there is also behavior inclusion. Intuitively, the idea behind bisimilar states is that each step one of them can take, can be mimicked by the other.

Operators of *restriction*, preventing the execution of certain actions, and *hiding*, masking certain actions with the τ one, can be used for defining information flow security properties. Compositionality is a fundamental issue when dealing with concurrent systems. Therefore, we provide a concept of *parallel composition* for the various models we present in this chapter.

In Section 1.1 we briefly introduce nondeterministic systems. In Section 1.2 we set up the probabilistic context by formalizing Probabilistic Automata. Finally, in Section 1.3 we introduce the timed setting by considering the model of Timed Automata.

1.1 Possibilistic Model

We introduce the formalism for nondeterministic, also called possibilistic, systems, together with a notion of weak bisimulation and operators of restriction, hiding and parallel composition.

1.1.1 The Formalism

In this section we recall some basic notions of finite *Labeled Transition Systems*. In the following of this dissertation we use the notation *Nondeterministic System* (NS) to denote a LTS.

Definition 1.1 *A* Nondeterministic System *(NS) is a tuple* $S = (\Sigma, Q, q_0, \delta)$, *where* Σ *is a set of labels,* Q *is a finite set of states with* $q_0 \in Q$ *the initial one. The set of transitions is given by* $\delta \subseteq Q \times \Sigma \cup \{\tau\} \times Q$, *where* τ *represents an internal silent move.*

$e_1 = (q_0, b, q_1)$
$e_2 = (q_0, a, q_2)$
$e_3 = (q_0, a, q_3)$

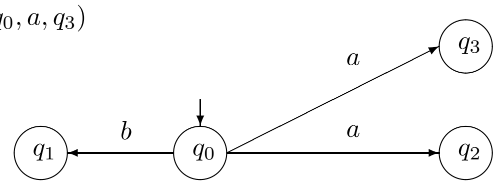

Figure 1.1: Example of NS.

Semantics

Given two states $q_i, q_j \in Q$ of $S = (\Sigma, Q, q_0, \delta)$, there is a step from q_i to q_j labeled with a (denoted $q_i \xrightarrow{a} q_j$) if $(q_i, a, q_j) \in \delta$. With S_T we denote the set of terminal states of S, namely $S_T = \{q \in Q \mid \forall q' \neq q \text{ and } \forall a \in \Sigma, (q, a, q') \notin \delta\}$.

An *execution fragment* of S is a finite sequence of steps $\sigma = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_k} q_k$, where $q_0, \dots, q_{k-1} \in Q$ and $a_i \in \Sigma \cup \{\tau\}$. With $ExecFrags_S$ we denote the set of execution fragments of S, and with $ExecFrags_S(q)$ we denote the set of execution fragments of S starting from q. We define $last(\sigma) = q_k$ and $|\sigma| = k$. The execution fragment σ is called *maximal* iff $last(\sigma) \in S_T$.

For any $j < |\sigma|$, with σ^j we define the sequence of steps $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_j} q_j$.

An *execution* of S is either a maximal execution fragment or an infinite sequence $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots$, where $q_0, q_1 \dots \in Q \setminus F$ and $a_1, a_2, \dots \in \Sigma$. We denote with $Execs_S$ the set of executions of S and with $Execs_S(q)$ the set of executions of S starting from q.

Example 1.1 *In Figure 1.1 we show an example of NS. Intuitively, from the initial state q_0, the NS may perform transition e_1 labeled with b reaching state q_1 or it can nondeterministically perform transitions e_2 and e_3, labeled with a and leading to states q_2 and q_3, respectively.*

An example of execution of the NS in Figure 1.1 is $\sigma = q_0 \xrightarrow{a} q_2$.

1.1.2 Behavioral Equivalence

As a relation of observational equivalence for NS, we now introduce the notion of weak bisimulation [111].

The *bisimulation* of a system by another system is based on the idea of mutual step-by-step simulation. Intuitively, two systems S and S' are bisimilar, if whenever one of the two systems executes a certain action and reaches a state q, the other system is able to simulate this single step by executing the same action and reaching a state q' which is again bisimilar to q. A *weak bisimulation* is a bisimulation which does not take into account τ (internal) moves. Hence, whenever a system simulates an action of the other system, it can also execute some internal τ actions before and after the execution of that action.

In order to abstract away from τ moves, Milner [111] introduced the notion of observable step, which consists of a single *visible* action a preceded and followed by an arbitrary number (including zero) of internal moves. Such moves are described by a *weak* transition relation $\Longrightarrow$, defined as $\stackrel{a}{\Longrightarrow} = (\xrightarrow{\tau})^* \xrightarrow{a} (\xrightarrow{\tau})^*$, where $\longrightarrow$ is the classical strong relation, and $\stackrel{\tau}{\Longrightarrow} = (\xrightarrow{\tau})^*$. It is worth noting that, with such a definition, a weak internal transition $\stackrel{\tau}{\Longrightarrow}$ is possible even without performing any internal action.

Definition 1.2 *Let $S = (\Sigma, Q, q_0, \delta)$ be a NS. A* weak bisimulation *on S is an equivalence relation $\mathcal{R} \subseteq \mathcal{Q} \times \mathcal{Q}$ such that for all $(p, q) \in \mathcal{R}$ it holds that $\forall a \in \Sigma$:*

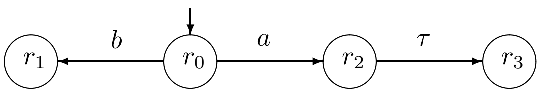

Figure 1.2: Example of weak bisimulation for NSs.

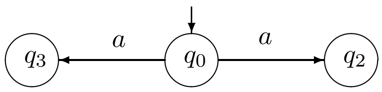

Figure 1.3: Example of restriction for NSs.

- *if* $p \xrightarrow{a} p'$*, then there exists* q' *such that* $q \stackrel{a}{\Longrightarrow} q'$ *and* $(p', q') \in \mathcal{R}$*;*
- *conversely, if* $q \xrightarrow{a} q'$*, then there exists* p' *such that* $p \stackrel{a}{\Longrightarrow} p'$ *and* $(p', q') \in \mathcal{R}$*.*

Two states p, q *are called* weakly bisimilar *on* S *(denoted* $p \approx_S q$*) iff* $(p, q) \in \mathcal{R}$ *for some weak bisimulation* $\mathcal{R}$*.*

Two NSs $S = (\Sigma, Q, q_0, \delta)$ *and* $S' = (\Sigma', Q', q'_0, \delta')$*, such that* $Q \cap Q' = \emptyset$*, are called weakly bisimilar (denoted by* $S \approx S'$*) if, given the NS* $\hat{S} = (\Sigma \cup \Sigma', Q \cup Q', q_0, \delta \cup \delta')$*, it holds* $q_0 \approx_{\hat{S}} q'_0$*.*

Note that it is always possible to obtain $Q \cap Q' = \emptyset$ by state renaming. Since we are just comparing two given states (i.e. q_0 and q'_0), the choice of the initial state of $\hat{S}$ does not affect the computation of the weak bisimulation equivalence classes [94]. We have chosen q_0, but we could choose q'_0 or any other state of $Q \cup Q'$ as well.

The following result holds.

Proposition 1.1 *It is decidable whether two NSs are weakly bisimilar.*

Example 1.2 *Let* S *be the NS in Figure 1.1 and* S' *be the NS in Figure 1.2. It holds that* $S \approx S'$*. Intuitively, from the initial states* q_0 *and* r_0 *the two NSs may either perform a step labeled with* a *or* b *and then reach a state where no other visible steps may be performed.*

1.1.3 Auxiliary Operators

We define operations of *restriction*, *hiding* and parallel composition on NSs.

We assume a NS $S = (\Sigma, Q, q_0, \delta)$ and a set $L \subseteq \Sigma$ of actions.

Definition 1.3 *The* restriction *of a NS* S *with respect to the set of actions* L *is* $S \backslash L = (\Sigma, Q, q_0, \delta')$*, where* $\delta' = \{(q, a, q') \in \delta \mid a \notin L\}$*.*

Example 1.3 *Let* S *be the NS in Figure 1.1 and* $L = \{b\}$*. In Figure 1.3 we show the NS* $S \setminus L$*, where every transition with label* b *is prevented.*

The *hiding* of a transition $e = (q, a, q')$ with respect to the set of actions L (written e/L) is defined as:

$$e/L = \begin{cases} e & if\ a \notin L \\ (q, \tau, q') & if\ a \in L \end{cases}$$

Definition 1.4 *The* hiding *of a NS* S *with respect to the set of actions* L *is given by* $S/L = (\Sigma, Q, q_0, \delta')$*, where* $\delta' = \{e/L \mid e \in \delta\}$*.*

Example 1.4 *Let* S *be the NS in Figure 1.1 and* $L = \{b\}$*. In Figure 1.4 we show the NS* S/L*, where label* b *is replaced with the label* τ *in every transition.*

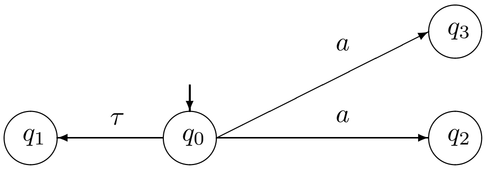

Figure 1.4: Example of hiding for NSs.

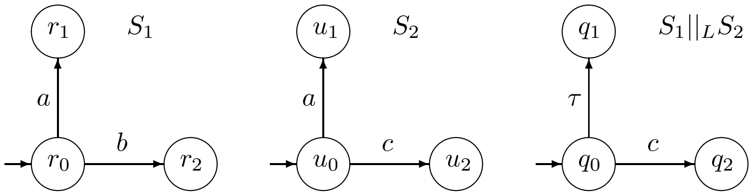

Figure 1.5: Example of parallel composition for NSs.

Proposition 1.2 *Given a NS S, $S \setminus L$ and S/L are NSs for all $L \subseteq \Sigma$.*

We assume two NSs $S_1 = (\Sigma, Q_1, r_0, \delta_1)$ and $S_2 = (\Sigma, Q_2, u_0, \delta_2)$ with $Q_1 \cap Q_2 = \emptyset$.

Definition 1.5 *The* parallel composition *of two NSs S_1 and S_2, with respect to the synchronization set L, is defined as $S_1||_L S_2 = (\Sigma, Q, (r_0, u_0), \delta)$, where $Q = Q_1 \times Q_2$ and, for any state $(r, u) \in Q$, the set of transitions δ is defined as follows:*

- *if system S_1 has a transition $e_1 = (r, a, r')$ with $a \notin L$, then $e = ((r, u), a, (r', u)) \in \delta$;*
- *if system S_2 has a transition $e_2 = (u, a, u')$ with $a \notin L$, then $e = ((r, u), a, (r, u')) \in \delta$;*
- *if system S_1 has a transition $e_1 = (r, a, r')$ with $a \in L$ and system S_2 has a transition $e_2 = (u, a, u')$, then S_1 and S_2 can synchronize and transition $e = ((r, u), \tau, (r', u')) \in \delta$.*

Note that, given such a definition of parallel composition, systems S_1 and S_2 are prevented from performing transitions with label in L without synchronizing. Also note that whenever S_1 and S_2 synchronize they give rise to an internal action τ.

Example 1.5 *In Figure 1.5 we show the NSs S_1, S_2 and $S_1||_L S_2$ with $L = \{a, b\}$. States q_0, q_1 and q_2 of $S_1||_L S_2$ correspond to the pairs (r_0, u_0), (r_1, u_1) and (r_0, u_2), respectively.*

Proposition 1.3 *Given the NSs S_1 and S_2, $S_1||_L S_2$ is a NS for all $L \subseteq \Sigma$.*

1.2 Probabilistic Model

We introduce the formalism for probabilistic systems together with a notion of weak bisimulation and operators of restriction, hiding and parallel composition. We also show how to remove probabilities in order to get a nondeterministic system, and that weak bisimulation is preserved when reducing to the possibilistic model. We shall use the same terminology for operators and bisimulation in the different models when this does not give rise to ambiguity.

1.2.1 The Formalism

In this section we introduce probabilities in the transitions of a NS. The model we obtain is a slight variant of the Markov Decision Processes and the Probabilistic Automata of [20, 71, 127].

Definition 1.6 *A* Probabilistic Automaton *(PA) is a tuple* $A = (\Sigma, Q, q_0, \delta, \Pi)$*, where:*

- Σ *is a finite alphabet of actions.*
- Q *is a finite set of states and* $q_0 \in Q$ *is the initial state.*
- $\delta \subseteq Q \times \Sigma \cup \{\tau\} \times Q$ *is a finite set of transitions. The symbol* τ *represents the silent or internal move. For a state* q*, we denote with* $start(q)$ *the set of transitions with* q *as source state, i.e. the set* $\{(q_1, a, q_2) \in \delta \mid q_1 = q\}$.
- $\Pi = \{\pi_1, \ldots, \pi_n\}$ *is a finite set of probability distributions as functions* $\pi_i : \delta \to [0, 1]$*, for any* $i = 1, \ldots, n$*, where* $\pi_i(e)$ *is the probability of performing transition* e *according to distribution* π_i*. We require that* $\sum_{e \in start(q)} \pi_i(e) \in \{0, 1\}$ *for any* i *and* q*. Moreover, we assume that for all* $\pi_i \in \Pi$ *there exists some* $e_j \in \delta$ *such that* $\pi_i(e_j) > 0$ *and, viceversa, for all* e_j *there exist some* π_i *such that* $\pi_i(e_j) > 0$.

Semantics

There is a *transition step* from a state q_i to a state q_j through action $(a, \pi) \in (\Sigma \cup \{\tau\}) \times \Pi$, written $q_i \xrightarrow{(a,\pi)} q_j$, if there is a transition $e = (q_i, a, q_j) \in \delta$ such that $\pi(e) > 0$. With S_T we denote the set of terminal states of S, namely $S_T = \{q \in Q \mid q \not\xrightarrow{(a,\pi)} q' \, \forall q' \in Q$ and $\forall (a, \pi) \in (\Sigma \cup \{\tau\}) \times \Pi\}$.

The probability of executing a transition step is chosen according to the values returned by the function π. Given a step $q_i \xrightarrow{(a,\pi)} q_j$, and $\alpha = (a, \pi)$, the probability $P(q_i, \alpha, q_j)$ of reaching state q_j from state q_i through a step labeled with α, is defined as $P(q_i, \alpha, q_j) = \pi((q_i, a, q_j))$.

Given a PA $A = (\Sigma, Q, q_0, \delta, \Pi)$, an *execution fragment* starting from q_0 is a finite sequence of transition steps $\sigma = q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} q_2 \xrightarrow{\alpha_3} \ldots \xrightarrow{\alpha_k} q_k$, where $q_0, q_1, \ldots, q_k \in Q$ and $\alpha_i \in (\Sigma \cup \{\tau\}) \times \Pi$. $ExecFrag_A$ is the set of execution fragments of A and $ExecFrag_A(q)$ is the set of execution fragments starting from q. We define $last(\sigma) = q_k$ and $|\sigma| = k$. The execution fragment σ is called *maximal* iff $last(\sigma) \in S_T$.

For any $j < k$, σ^j is the sequence of steps $q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_j} q_j$.

If $|\sigma| = 0$ we put $P_A(\sigma) = 1$, else, if $|\sigma| = k \geq 1$, we define $P_A(\sigma) = P_A(q_0, \alpha_1, q_1) \cdot \ldots \cdot P_A(q_{k-1}, \alpha_k, q_k)$.

An *execution* is either a maximal execution fragment or an infinite sequence $q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} q_2 \xrightarrow{\alpha_3} \ldots$, where $q_0, q_1 \ldots \in Q$ and $\alpha_1, \alpha_2, \ldots \in (\Sigma \cup \{\tau\}) \times \Pi$. We denote with $Exec_A$ the set of executions of A and with $Exec_A(q)$ the set of executions starting from q. Finally, let $\sigma \uparrow$ denote the set of executions σ' such that $\sigma \leq_{prefix} \sigma'$, where *prefix* is the usual prefix relation over sequences.

Executions and execution fragments of a PA arise by resolving both the nondeterministic and the probabilistic choices [85]. We need a notion of *scheduler* to resolve the nondeterminism that arises when choosing the distribution π within a set Π.

A *scheduler* for a Probabilistic Automaton A is a function F assigning to each finite sequence σ in $ExecFrag_A$ a distribution $\pi \in \Pi$. Namely, $F : ExecFrag_A \to \Pi$. With $\mathcal{F}_A$ we denote the set of schedulers for A.

Given a scheduler F and an execution fragment σ, we assume that F is defined for σ if and only if $\exists q \in Q$ and $a \in \Sigma \cup \{\tau\}$ such that $last(\sigma) \xrightarrow{(a,F(\sigma))} q$.

For a scheduler F we define $ExecFrag_A^F$ (resp. $Exec_A^F$) as the set of execution fragments (resp. executions) $\sigma = q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} q_2 \xrightarrow{\alpha_3} \ldots$ of A such that $F(\sigma^{i-1}) = \pi_i$, for any $0 < i < |\sigma|$ and $\alpha_i = (a_i, \pi_i)$.

$e_1 = (q_0, b, q_1) \qquad \pi(e_1) = \frac{1}{6}$
$e_2 = (q_0, a, q_2) \qquad \pi(e_2) = \frac{1}{3}$
$e_3 = (q_0, a, q_3) \qquad \pi(e_3) = \frac{1}{2}$

$\sum_{e \in \delta(q_0)} \pi(e) = 1$

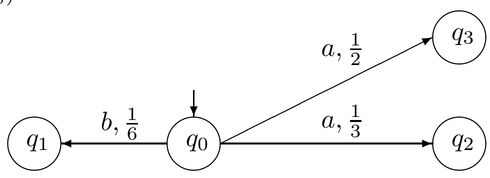

Figure 1.6: Example of PA.

Assuming the basic notions of probability theory (see e.g. [59]), we define the probability space on the executions starting from a given state $q \in Q$, as follows. Given a scheduler F, let $Exec_A^F(q)$ be the set of executions in $Exec_A^F$ starting from q, $ExecFrag_A^F(q)$ be the set of execution fragments in $ExecFrag_A^F$ starting from q, and $\Sigma\_Field_A^F(q)$ be the smallest sigma field on $Exec_A^F(q)$ that contains the basic cylinders $\sigma\uparrow$, where $\sigma \in ExecFrag_A^F(q)$. The probability measure $Prob_A^F$ is the unique measure on $\Sigma\_Field_A^F(q)$ such that $Prob_A^F(\sigma\uparrow) = P_A(\sigma)$.

Given a scheduler F, a state q and a set of states $Q' \subseteq Q$, with $Exec_A^F(q, Q')$ we denote the set of executions starting from q that cross a state in the set Q'. Namely, $Exec_A^F(q, Q') = \{\sigma \in Exec_A^F(q) \mid last(\sigma^i) \in Q', \text{ for some } i\}$.

We shall omit the index A when it is clear from the context which is the automaton we refer to.

In the following, if $\alpha = (a, \pi)$, with $\hat{\alpha}$ we denote a, and $\mathcal{C} \subseteq Q$. Given a scheduler F, $Exec^F(\tau^*\hat{\alpha}, \mathcal{C})$ is the set of executions that lead to a state in $\mathcal{C}$ via a sequence belonging to the set of sequences $\tau^*\hat{\alpha}$. We define $Exec^F(q, \tau^*\hat{\alpha}, \mathcal{C}) = Exec^F(\tau^*\hat{\alpha}, \mathcal{C}) \cap Exec^F(q)$. Finally, we define the probability $Prob^F(q, \tau^*\hat{\alpha}, \mathcal{C}) = Prob^F(Exec^F(q, \tau^*\hat{\alpha}, \mathcal{C}))$. The next proposition derives directly by this definition.

Proposition 1.4 *It holds that* $Prob^F(q, \tau^*\hat{\alpha}, \mathcal{C}) =$

$$\begin{cases} 1 & if\ \hat{\alpha} = \tau \wedge q \in \mathcal{C} \\ \sum_{r \in Q} Prob^F(q, \tau, r) \cdot Prob^F(r, \tau^*, \mathcal{C}) & if\ \hat{\alpha} = \tau \wedge q \notin \mathcal{C} \\ \sum_{r \in Q} Prob^F(q, \tau, r) \cdot Prob^F(r, \tau^*\alpha, \mathcal{C}) + Prob^F(q, \alpha, \mathcal{C}) & if\ \hat{\alpha} \neq \tau \end{cases}$$

If a PA does not allow nondeterministic choices it is said to be *fully probabilistic*.

Definition 1.7 *Given a PA* $A = (\Sigma, Q, q_0, \delta, \Pi)$*, we say that* A *is* fully probabilistic *if* $|\Pi| = 1$.

Example 1.6 *In Figure 1.6 we show an example of PA with* $\Pi = \{\pi\}$*. Intuitively, from the initial state* q_0*, the PA performs probabilistically transitions* e_1*,* e_2 *or* e_3 *with probabilities* $\frac{1}{6}$*,* $\frac{1}{3}$ *and* $\frac{1}{2}$*, respectively. Note that* $|\Pi| = 1$*, thus the PA is* fully probabilistic*. As a consequence, nondeterministic choices are not performed since every scheduler can return the only distribution* π*.*

Examples of executions of the PA in Figure 1.6 are $\sigma_1 = q_0 \xrightarrow{(a,\pi)} q_3$ *and* $\sigma_2 = q_0 \xrightarrow{(b,\pi)} q_1$ *with* $P(\sigma_1) = \frac{1}{2}$ *and* $P(\sigma_2) = \frac{1}{6}$*.*

The next proposition derives from results in [28].

Proposition 1.5 *Let* A_1 *and* A_2 *be two PAs and* Q_1 *and* Q_2 *be two subsets of states of* A_1 *and* A_2*, respectively. It is decidable in exponential time whether for any scheduler* F *of* A_1 *there exists* *a scheduler F' of A_2 such that $Prob^F_{A_1}(Exec^F_{A_1}(q_1, Q_1)) = Prob^{F'}_{A_2}(Exec^{F'}_{A_2}(q_2, Q_2))$, where q_1 and q_2 are states of A_1 and A_2, respectively. If for any $\sigma \in Exec^F_{A_1}(q_1, Q_1) \cup Exec^{F'}_{A_2}(q_2, Q_2)$ it holds that $\sigma^1 \in Q_1 \cup Q_2$, then the problem is decidable in polynomial time.*

1.2.2 Behavioral Equivalence

For the definition of weak bisimulation in the fully probabilistic setting, Baier and Hermanns [15] replace Milner's weak internal transitions $q \stackrel{\tau}{\Longrightarrow} q'$ by the probability $Prob(q, \tau^*, q')$ of reaching configuration q' from q via internal moves. Similarly, for visible actions α, Baier and Hermanns define $\stackrel{\alpha}{\Longrightarrow}$ by means of the probability $Prob(q, \tau^*\alpha, q')$. Given a scheduler F, the probabilistic model we have chosen for PAs reduces to the one of fully probabilistic systems. In such a model, as proved in [15], the two relations of weak bisimulation and branching bisimulation do coincide. Relying on this result, we use branching bisimulation in order to decide weak bisimulation. The following definition derives from the definition of Baier and Hermanns [15] and Aldini et al. [6].

Definition 1.8 *Let $A = (\Sigma, Q, q_0, \delta, \Pi)$ be a PA. A* weak bisimulation *on A is an equivalence relation $\mathcal{R}$ on Q such that, for all $(q, q') \in \mathcal{R}$, $\mathcal{C} \in Q/\mathcal{R}$ and schedulers F, there exists a scheduler F' such that*

$$Prob^F(q, \tau^*\alpha, \mathcal{C}) = Prob^{F'}(q', \tau^*\alpha, \mathcal{C}) \qquad \forall \alpha \in \Sigma \cup \{\tau\}$$

and vice versa.
Two states q, q' are called weakly bisimilar *on A (denoted $q \approx_A q'$) iff $(q, q') \in \mathcal{R}$ for some weak bisimulation $\mathcal{R}$.*

Definition 1.9 *Two PAs $A = (\Sigma, Q, q_0, \delta, \Pi)$ and $A' = (\Sigma', Q', q'_0, \delta', \Pi')$ such that $Q \cap Q' = \emptyset$ are called weakly bisimilar (denoted by $A \approx A'$) if, given the PA $\hat{A} = (\Sigma \cup \Sigma', Q \cup Q', q_0, \delta \cup \delta', \hat{\Pi})$, where, for each couple $(\pi, \pi') \in \Pi \times \Pi'$, $\hat{\pi} \in \hat{\Pi}$ such that*

$$\hat{\pi}(e) = \begin{cases} \pi(e) & if\ e \in \delta \\ \pi'(e) & if\ e \in \delta' \end{cases}$$

it holds $q_0 \approx_{\hat{A}} q'_0$.

Decidability results for probabilistic weak bisimulation have been proved by Baier and Hermanns (see [14, 15]).

Proposition 1.6 *It is decidable whether two PAs are weakly bisimilar.*

1.2.3 Auxiliary Operators

We define operations of *restriction*, *hiding* and *parallel composition* on PAs.

We assume a PA $A = (\Sigma, Q, q_0, \delta, \Pi)$ and a set $L \subseteq \Sigma$ of actions.

Definition 1.10 *The* restriction *of a PA A with respect to the set of actions L is $A \setminus L = (\Sigma, Q, q_0, \delta', \Pi')$, where:*

- *$\delta' = \{(q, a, q') \in \delta \mid a \notin L\}$.*
- *$\pi' \in \Pi'$ iff $\pi \in \Pi$ where, for all $e = (q, a, q') \in \delta'$, $\pi'(e) = \frac{\pi(e)}{\sum_{e' \in \delta' \cap start(q)} \pi(e')}$.*

The second condition is assumed in order to normalize the probability of each transition according to the ones remaining after the restriction. Thanks to this rule the condition $\sum_{e \in start(q)} \pi'(e) \in \{0, 1\}$ continues to be true for each state q of $A \setminus L$.

Example 1.7 *Let A be the PA in Figure 1.6 and $L = \{b\}$. In Figure 1.7 we show the PA $A \setminus L$, where every transition with label b is prevented and probabilities are redistributed.*

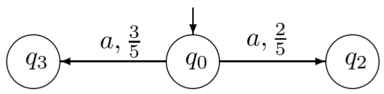

Figure 1.7: Example of restriction for PAs.

Again, we assume a PA $A = (\Sigma, Q, q_0, \delta, \Pi)$ and a set $L \subseteq \Sigma$ of actions.

The *hiding* of a transition $e = (q, a, q')$ with respect to the set of actions L (written e/L) is defined as:

$$e/L = \begin{cases} e & if\ a \notin L \\ (q, \tau, q') & if\ a \in L \end{cases}$$

Definition 1.11 *The hiding of a PA A with respect to the set of actions L is given by $A/L = (\Sigma, Q, q_0, \delta', \Pi')$, where $\delta' = \{e/L \mid e \in \delta\}$, and $\Pi' = \{\pi' | \exists \pi \in \Pi. \forall e' \in \delta'\ \pi'(e') = \sum_{e \in \delta : e/L = e'} \pi(e)\}$.*

Proposition 1.7 *Given a PA A, $A \setminus L$ and A/L are PAs for all $L \subseteq \Sigma$.*

Now, we introduce a parallel composition operator with a parameter p specifying the probability p for the left automaton to make the first step, and the probability $1-p$ for the right automaton to proceed first (provided no communication appears). However, other kinds of parallel composition operators can be defined without the use of parameterized operator (see [138] for a detailed discussion).

Assume PAs $A_1 = (\Sigma, Q_1, r_0, \delta_1, \Pi_1)$ and $A_2 = (\Sigma, Q_2, u_0, \delta_2, \Pi_2)$ with disjoint sets of states ($Q_1 \cap Q_2 = \emptyset$). We also assume a set $L \subseteq \Sigma$ of synchronization actions. Finally, for $i \in \{1, 2\}$, given a transition $e = (q, a, q') \in \delta_i$, and a probability distribution $\pi_i \in \Pi_i$ with $\pi_{i_a}(e)$ we denote the normalized probability of executing transition e with respect to all other transitions starting from q and labeled with a, i.e. $\pi_{i_a}(e) = \frac{\pi_i(e)}{\sum_{e' \in start_i^a(q)} \pi_i(e')}$, where $start_i^a(q)$ denotes the set of transitions in δ_i with q as source state and a as labeling action, i.e. the set $\{(q_1, a', q_2) \in \delta_i \,|\, q_1 = q \wedge a' = a\}$.

Definition 1.12 *The* parallel composition *of two PAs A_1 and A_2, with respect to the synchronization set L and the advancing speed parameter $p \in]0, 1[$, is defined as $A_1 ||_L^p A_2 = (\Sigma, Q, (r_0, u_0), \delta, \Pi)$. The set $Q = Q_1 \times Q_2$ of states of $A_1 ||_L^p A_2$ is given by the cartesian product of the states of the two automata A_1 and A_2. Given a state (r, u) of $A_1 ||_L^p A_2$ there is a probability distribution $\pi \in \Pi$ for any two probability distributions $\pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2$. In particular, $\delta = S_1 \cup S_2 \cup \bigcup_{a \in L} S_3^a$ where $S_1 = \{((r, u), b, (r', u)) \mid (r, b, r') \in \delta_1, u \in Q_2, b \notin L\}$, $S_2 = \{((r, u), b, (r, u')) \mid (u, b, u') \in \delta_2, r \in Q_1, b \notin L\}$ and, for any $a \in L$, $S_3^a = \{((r, u), \tau, (r', u')) \mid (r, a, r') \in \delta_1, (u, a, u') \in \delta_2\}$. Moreover, for any pair $\pi_1 \in \Pi_1$, $\pi_2 \in \Pi_2$, there exists $\pi \in \Pi$ such that, for all $e = (q, a, q') \in \delta$, it holds that $\pi(e) = \frac{f(e)}{\sum_{e' \in \delta \cap start(q)} f(e')}$ where*

$$f(e) = \begin{cases} \pi_1(e) \cdot p & if\ e \in S_1 \\ \pi_2(e) \cdot (1-p) & if\ e \in S_2 \\ \pi_1(e) \cdot p \cdot \pi_{2_a}(e) + \pi_2(e) \cdot (1-p) \cdot \pi_{1_a}(e) & if\ e \in \bigcup_{a \in L} S_3^a \end{cases}$$

Note that, given such a definition of parallel composition, automata A_1 and A_2 are prevented from performing transitions with label in L without synchronizing. Moreover, whenever A_1 and A_2 synchronize they give rise to an internal action τ. Also note that, chosen a transition e_1 (e_2) with label $a \in L$ of automaton A_1 (A_2) the transition e_2 (e_1) of A_2 (A_1) that synchronizes with e_1 (e_2) is chosen according to the probability $\pi_{2_a}(e_2)$ ($\pi_{1_a}(e_1)$) normalized with respect to all the other transitions labeled with a. Besides, according to Definition 1.6, given the parallel composition defined above, it holds that $\sum_{e \in start(q)} \pi(e) \in \{0, 1\}$ for each state q of $A_1 ||_L^p A_2$. This is done due to the last rule, that uses the auxiliary structure $f(e)$ in order to compute the normalized probabilities in π. In fact, transitions of the single automata A_1 and A_2 with label $a \in L$ are not

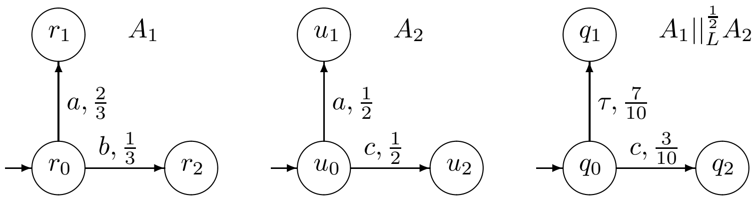

Figure 1.8: Example of parallel composition for PAs.

allowed to be performed without synchronization, and therefore they are lost in the compound system together with their probabilities (and therefore probabilities of the compound system must be redistributed). In the following of this thesis, when we omit the parameter p from the parallel composition operator of a probabilistic model, we assume it to be equal to $\frac{1}{2}$.

Example 1.8 *In Figure 1.8 we show the PAs A_1, A_2 and $A_1||_L^{\frac{1}{2}}A_2$ with $L = \{a, b\}$. States q_0, q_1 and q_2 of $A_1||_L A_2$ correspond to the pairs (r_0, u_0), (r_1, u_1) and (r_0, u_2), respectively.*

Proposition 1.8 *Given the PAs A_1 and A_2, $A_1||_L^p A_2$ is a PA for all $p \in]0, 1[$ and $L \subseteq \Sigma$.*

1.2.4 From PAs to NSs

Given a PA A, we call *unprob*(A) the NS obtained from A by removing Π. This can be done since we assumed that for each transition of A there is at least a probability distribution which assigns to such a transition a probability greater than 0 (see Definition 1.6).

Definition 1.13 *Given a PA $A = (\Sigma, Q, q_0, \delta, \Pi)$, $unprob(A) = (\Sigma, Q, q_0, \delta)$.*

Example 1.9 *Let A be the PA in Figure 1.6. If we remove probabilities from A the NS unprob(A) can be found in Figure 1.1.*

Relations among weak bisimulations

Lemma 1.1 *Given PAs A and A', $A \approx A' \Rightarrow unprob(A) \approx unprob(A')$*

Proof. Let us assume $A = (\Sigma, Q, q_0, \delta, \Pi)$, $A' = (\Sigma', Q', q_0', \delta', \Pi')$ and $\hat{A}$ constructed as in Definition 1.6. Since $A \approx A'$ for a weak bisimulation $\mathcal{R}$, we have that for all $(q, r) \in \mathcal{R}$, $\mathcal{C} \in Q \cup Q'/\mathcal{R}$ and schedulers F, there exists a scheduler F' such that $Prob^F_{\hat{A}}(q, \tau^*\alpha, \mathcal{C}) = Prob^{F'}_{\hat{A}}(r, \tau^*\alpha, \mathcal{C})$ $\forall \alpha \in \Sigma \cup \{\tau\}$. Now, if $Prob^F_{\hat{A}}(q, \alpha, q') > 0$ for some $q' \in \mathcal{C}$ there exists a configuration r' and a scheduler F' such that $Prob^{F'}_{\hat{A}}(r, \tau^*\alpha, r') = Prob^F_{\hat{A}}(q, \alpha, q') > 0$. Therefore if $q \xrightarrow{\alpha} q'$, then there exists r' such that $r \stackrel{\alpha}{\Longrightarrow} r'$ and, since q' and r' are in the same equivalence class, there exists also a bisimulation $\mathcal{R}'$ on $\hat{Q}_{np}$ such that $(q', r') \in \mathcal{R}'$, where $\hat{Q}_{np}$ is the set of states of the NS constructed as in Definition 1.2 starting from *unprob*(A) and *unprob*(A'). The same holds if we exchange the roles of q and r. $\square$

1.3 Timed Model

We introduce the formalism for timed systems together with a notion of weak bisimulation and operators of restriction, hiding and parallel composition. We also show how to remove time in order to get a nondeterministic system, and that weak bisimulation is preserved when reducing to the possibilistic model.

1.3.1 The Formalism

We recall the definition of Timed Automata ([9]).

Let us assume a set X of positive real variables called *clocks*. A *valuation* over X is a mapping $v : X \to \mathbb{R}^{\geq 0}$ assigning real values to clocks. For a valuation v and a time value $t \in \mathbb{R}^{\geq 0}$, let $v + t$ denote the valuation such that $(v + t)(x) = v(x) + t$, for each clock $x \in X$.

The set of *constraints* over X, denoted $\Phi(X)$, is defined by the following grammar, where ϕ ranges over $\Phi(X)$, $x \in X$, $c \in \mathbb{Q}$, and $\sim \in \{<, \leq, =, \neq, >, \geq\}$:

$$\phi ::= true \,|\, x \sim c \,|\, \phi_1 \wedge \phi_2 \,|\, \phi_1 \vee \phi_2 \,|\, \neg\phi$$

We write $v \models \phi$ when *the valuation* v *satisfies the constraint* ϕ. Formally, it holds that $v \models true$, $v \models x \sim c$ iff $v(x) \sim c$, $v \models \phi_1 \wedge \phi_2$ iff $v \models \phi_1$ and $v \models \phi_2$, $v \models \phi_1 \vee \phi_2$ iff $v \models \phi_1$ or $v \models \phi_2$, and $v \models \neg\phi$ iff $v \not\models \phi$.

Let $B \subseteq X$; with $v[B]$ we denote the valuation resulting after resetting all clocks in B. More precisely, $v[B](x) = 0$ if $x \in B$, $v[B](x) = v(x)$, otherwise. Finally, with $\mathbf{0}$ we denote the valuation with all clocks reset to 0, namely $\mathbf{0}(x) = 0$ for all $x \in X$.

Definition 1.14 *A* Timed Automaton *(TA) is a tuple* $A = (\Sigma, X, Q, q_0, \delta, Inv)$*, where* Σ*,* X*,* Q*,* q_0*,* δ *and* Inv *where:*

- Σ *is a finite alphabet of actions.*
- X *is a finite set of positive real variables called clocks.*
- Q *is a finite set of states and* $q_0 \in Q$ *is the initial state.*
- $\delta \subseteq Q \times \Sigma \cup \{\tau\} \times \Phi(X) \times 2^X \times Q$ *is a finite set of transitions. The symbol* τ *represents the silent or internal move. For a state* q*, we denote with* $start(q)$ *the set of transitions with* q *as source state, i.e. the set* $\{(q_1, a, \phi, B, q_2) \in \delta \,|\, q_1 = q\}$.
- $Inv : Q \to \Phi(X)$ *is a function assigning a constraint* $\phi \in \Phi(X)$ *(called* state invariant*) to each state in* Q.

Semantics

A *configuration* of a TA A is a pair (q, v), where $q \in Q$ is a state of A, and v is a valuation over X. The initial configuration of A is represented by $s_0 = (q_0, \mathbf{0})$ and the set of all the configurations of A is denoted with $\mathcal{S}_A$.

There is a discrete *transition step* from a configuration $s_i = (q_i, v_i)$ to a configuration $s_j = (q_j, v_j)$ through action $a \in \Sigma \cup \{\tau\}$, written $s_i \xrightarrow{a} s_j$, if there is a transition $e = (q_i, a, \phi, B, q_j) \in \delta$ such that $v_i \models \phi \wedge Inv(q_i)$, $v_j = v_i[B]$ and $v_j \models Inv(q_j)$.

There is a continuous *time step* from a configuration $s_i = (q_i, v_i)$ to a configuration $s_j = (q_j, v_j)$ through time $t \in \mathbb{R}^{>0}$, written $s_i \xrightarrow{t} s_j$, if $q_j = q_i$, $v_j = (v_i + t)$ and $\forall t' \in [0, t]\ v_i + t' \models Inv(q_i)$.

An *execution fragment* of A is a finite sequence of steps $\sigma = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} s_k$, where $s_0, \ldots, s_{k-1} \in \mathcal{S}_A$, and $a_i \in \Sigma \cup \{\tau\}$. With $ExecFrag_A$ we denote the set of execution fragments of A, and with $ExecFrag_A(s)$ we denote the set of execution fragments of A starting from configuration s. We define $last(\sigma) = s_k$ and $|\sigma| = k$.

For any $j < |\sigma|$, with σ^j we define the sequence of steps $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_j} s_j$.

An *execution* of A is an infinite sequence of steps $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$, where $s_0, s_1 \ldots \in \mathcal{S}_A$ and $a_1, a_2, \ldots \in \Sigma \cup \{\tau\}$. We denote with $Exec_A$ the set of executions of A and with $Exec_A(s)$ the set of executions of A starting from s.

$e_1 = (q_0, b, 0 \leq x \leq 5, \emptyset, q_1)$
$e_2 = (q_0, a, true, \{x\}, q_2)$
$e_3 = (q_0, a, true, \emptyset, q_3)$

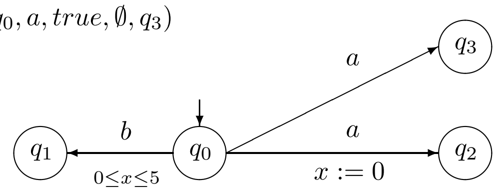

Figure 1.9: Example of TA.

Example 1.10 *In Figure 1.9 we show an example of TA with $Inv(q_i) = true$ for all $i \in [0, 3]$. In this example, and in the following ones, we omit the condition on a transition when the condition is true.*

Intuitively, from the initial state q_0, the TA may always perform some time step and update the value of clock x. Transition e_1 labeled with b can be performed by the TA if the value of clock x is less or equal to 5. Transitions e_2 and e_3, labeled with a and leading to states q_2 and q_3, respectively, may be performed at any time (their guard condition is true). Note, however, that if transition e_2 is performed, then the value of the clock x is reset to 0.

An example of execution fragment of the TA in Figure 1.9 is $(q_0, 0) \xrightarrow{9.7} (q_0, 9.7) \xrightarrow{a} (q_2, 0) \xrightarrow{3.2} (q_2, 3.2)$, where (q, t) represents the configuration composed by the state q and the valuation v such that $v(x) = t$.

1.3.2 Regions of Timed Automata

We recall the definitions of clock equivalence [9] and the theory of clock zones [24]. Clock equivalence is an equivalence relation of finite index permitting to group sets of evaluations and to have decidability results. Unfortunately, the number of equivalence classes is exponential w.r.t. the size of the TA. To avoid this, efficient symbolic representation by means of clock zones is introduced (see [68] and [24]).

Let A be a TA; with C_A we denote the greatest constant that appears in A.

Let us consider the equivalence relation $\sim$ over clock valuations containing precisely the pairs (v, v') such that:

- for each clock x, either $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, or both $v(x)$ and $v'(x)$ are greater than C_A;
- for each pair of clocks x and y with $v(x) \leq C_A$ and $v(y) \leq C_A$ it holds that $fract(v(x)) \leq fract(v(y))$ iff $fract(v'(x)) \leq fract(v'(y))$ (where $fract(\cdot)$ is the fractional part);
- for each clock x with $v(x) \leq C_A$, $fract(v(x)) = 0$ iff $fract(v'(x)) = 0$.

As proved in [9], $v \sim v'$ implies that, for any $\phi \in \Phi(X)$ with constants less or equal than C_A, $v \models \phi$ iff $v' \models \phi$. With $[v]$ we denote the equivalence class $\{v' \,|\, v' \sim v\}$. The set V of equivalence classes is finite, and with $|V|$ we denote its cardinality.

We now briefly recall the definition of *clock zone* and its properties. For more details see [24] and [68].

The set of clock zones on X (denoted with $\Psi(X)$) is the set of formulae ψ such that

$$\psi ::= true \,|\, false \,|\, x \sim c \,|\, x - y \sim c \,|\, \psi_1 \wedge \psi_2 \,|\, \psi_1 \vee \psi_2 \,|\, \neg\psi$$

where $\sim \{<, \leq, >, \geq, =\}$, $c \in \mathbb{Z}$ and $x, y \in X$.

With $\Psi_C(X)$ we denote the set of clock zones in $\Psi(X)$ that use integer constants in $[-C, C]$.

Known properties of clock zones are expressed by the following propositions.

Proposition 1.9 *Let ψ be a clock zone in $\Psi(X)$ and $x \in X$. A clock zone ψ' in $\Psi(X \setminus x)$ such that $\psi' \equiv \exists x.\psi$ is computable in polynomial time.*

The following proposition gives an upper bound to the number of clock zones.

Proposition 1.10 *There exists $\Psi' \subset \Psi_C(X)$ with exponential cardinality w.r.t. C and $|X|$ such that each clock zone in $\Psi_C(X)$ is equivalent to a clock zone in Ψ'.*

Definition 1.15 *Let A be a TA with states in Q and clocks in X; a* region *of A is a pair (q, ψ) where $q \in Q$ and $\psi \in \Psi(X)$.*

The following proposition states that the set of configurations reachable by performing either a discrete or a continuous time step starting from a set of configurations expressed by a region, is a region. In order to improve readability, without loss of generality, we omit state invariant conditions.

Proposition 1.11 *Given a region (q, ψ) and a transition $e = (q', \phi, a, B, q)$, the set of configurations $\{(q', v) \mid v \models \phi$ and $v[B] \models \psi\}$ from which it is possible to reach a configuration within the region (q, ψ) by a discrete step triggered by e, is equal to the region $(q', \phi \wedge \exists B.\psi \wedge B = 0)$.*

The set of configurations $\{(q, v) \mid v + t \models \psi$ and $t \in \mathbb{R}^{>0}\}$ from which it is possible to reach a configuration expressed by (q, ψ) by a continuous time step, is equal to the region $(q, \exists t.t > 0 \wedge \psi[X := X + t])$. Moreover, if $\psi \in \Psi_C(X)$, then $\exists t.t > 0 \wedge \psi[X := X + t] \in \Psi_C(X)$.

Since the set of regions is not finite, we need an approximation. If ψ is a clock zone of A, we denote with $Ap_A(\psi)$ the set $\{[v] \mid [v] \cap \psi \neq false\}$. The following proposition, proved in [24], states that Ap_A returns a clock zone.

Proposition 1.12 (Approximation) $Ap_A(\psi) \in \Psi_{C_A}(X)$.

The following theorem, proved in [24], states the correctness of the operator Ap.

Theorem 1.1 *The sequence of steps $(q_0, v_0) \xrightarrow{\alpha_1} (q_1, v_1) \xrightarrow{\alpha_2} \ldots$ is an execution of A iff there exists a sequence of regions $(q_0, \psi_0), (q_1, \psi_1) \ldots$ such that, for all i, $v_i \models \psi_i$ and, if $\alpha_i \in \Sigma \cup \{\tau\}$, then $\psi_i = Ap_A(\phi \wedge \exists B.\psi_{i+1} \wedge B = 0)$ for some transition $(q_i, \alpha_i, \phi, B, q_{i+1})$, and, otherwise, $\psi_i = \exists t.t > 0 \wedge \psi_{i+1}[X := X + t]$ and $q_{i+1} = q_i$. Moreover, each ψ_i is computable in polynomial time w.r.t. C_A and $|X|$.*

1.3.3 Behavioral Equivalence

The definition of weak bisimulation introduced for NSs (see Definition 1.2) can be naturally adapted for TAs.

Definition 1.16 *Let $A = (\Sigma, X, Q, q_0, \delta, Inv)$ be a TA. A* weak bisimulation *on A is an equivalence relation $\mathcal{R} \subseteq \mathcal{S}_\mathcal{A} \times \mathcal{S}_\mathcal{A}$ such that for all $(s, r) \in \mathcal{R}$ it holds that $\forall \alpha \in \Sigma \cup \{\tau\} \cup \mathbb{R}^{>0}$:*

- *if $s \xrightarrow{\alpha} s'$, then there exists r' such that $r \stackrel{\alpha}{\Longrightarrow} r'$ and $(s', r') \in \mathcal{R}$;*
- *conversely, if $r \xrightarrow{\alpha} r'$, then there exists s' such that $s \stackrel{\alpha}{\Longrightarrow} s'$ and $(s', r') \in \mathcal{R}$.*

Two configurations s, r are called weakly bisimilar *on A (denoted $s \approx_A r$) iff $(s, r) \in \mathcal{R}$ for some weak bisimulation $\mathcal{R}$.*
Two TAs $A = (\Sigma, X, Q, q_0, \delta, Inv)$ and $A' = (\Sigma', X', Q', q_0', \delta', Inv')$ such that $Q \cap Q' = \emptyset$ and $X \cap X' = \emptyset$ are called weakly bisimilar (denoted by $A \approx A'$) if, given the TA $\hat{A} = (\Sigma \cup \Sigma', X \cup X', Q \cup Q', q_0, \delta \cup \delta', \hat{Inv})$, it holds $(q_0, \mathbf{0}) \approx_{\hat{A}} (q_0', \mathbf{0})$, where:

$$\hat{Inv}(q) = \begin{cases} Inv(q) & if\ q \in Q \\ Inv'(q) & if\ q \in Q' \end{cases}$$

The following result is proved in [101] and [29].

Proposition 1.13 *It is decidable whether two TAs are weakly bisimilar.*

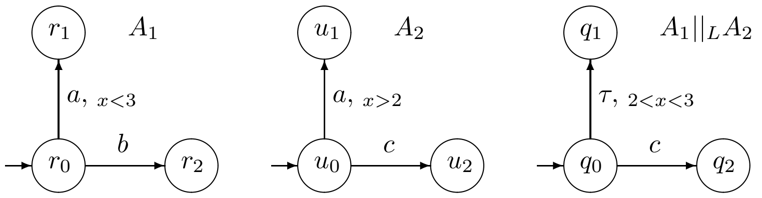

Figure 1.10: Example of parallel composition for TAs.

1.3.4 Auxiliary Operators

We define operations of *restriction*, *hiding* and parallel composition on TAs.

We assume a TA $A = (\Sigma, X, Q, q_0, \delta, Inv)$ and a set $L \subseteq \Sigma$ of actions.

Definition 1.17 *The* restriction *of a TA A with respect to the set of actions L is $A \setminus L = (\Sigma, X, Q, q_0, \delta', Inv)$, where $\delta' = \{(q, a, \phi, B, q') \in \delta \mid a \notin L\}$.*

Again, the *hiding* of a transition $e = (q, a, \phi, B, q')$ with respect to the set of actions L (written e/L) is defined as:

$$e/L = \begin{cases} e & if\ a \notin L \\ (q, \tau, \phi, B, q') & if\ a \in L \end{cases}$$

Definition 1.18 *The* hiding *of a TA A with respect to the set of actions L is given by $A/L = (\Sigma, X, Q, q_0, \delta', Inv)$, where $\delta' = \{e/L \mid e \in \delta\}$.*

Proposition 1.14 *Given a TA A, $A \setminus L$ and A/L are TAs for all $L \subseteq \Sigma$.*

We assume two TAs $A_1 = (\Sigma, X_1, Q_1, r_0, \delta_1, Inv_1)$ and $A_2 = (\Sigma, X_2, Q_2, u_0, \delta_2, Inv_2)$ with $Q_1 \cap Q_2 = \emptyset$ and $X_1 \cap X_2 = \emptyset$.

Definition 1.19 *The* parallel composition *of two TAs A_1 and A_2, with respect to the synchronization set L, is defined as $A_1||_L A_2 = (\Sigma, X, Q, (r_0, u_0), \delta, Inv)$, where $Q = Q_1 \times Q_2$ and $X = X_1 \cup X_2$. State invariants are defined as $Inv : Q_1 \times Q_2 \to \Phi(X_1 \cup X_2)$, where, for any $r \in Q_1$ and $u \in Q_2$ such that $Inv_1(r) = \phi_1$ and $Inv_2(u) = \phi_2$ we have that $Inv(r, u) = \phi_1 \wedge \phi_2$. For any state $(r, u) \in Q$, the set of transitions δ is defined as follows:*

- *if from state r the automaton A_1 has a transition $e_1 = (r, a, \phi, B, r')$ with $a \notin L$, then $e = ((r, u), a, \phi, B, (r', u)) \in \delta$;*
- *if from state u the automaton A_2 has a transition $e_2 = (u, a, \phi, B, u')$ with $a \notin L$, then $e = ((r, u), a, \phi, B, (r, u')) \in \delta$;*
- *if from state r the automaton A_1 has a transition $e_1 = (r, a, \phi_1, B_1, r')$ with $a \in L$ and from state u the automaton A_2 has a transition $e_2 = (u, a, \phi_2, B_2, u')$, then A_1 and A_2 can synchronize and transition $e = ((r, u), \tau, \phi_1 \wedge \phi_2, B_1 \cup B_2, (r', u')) \in \delta$.*

Note that, given such a definition of parallel composition, automata A_1 and A_2 are prevented of performing transitions with label in L without synchronizing. Also note that whenever S_1 and S_2 synchronize they give rise to an internal action τ.

Example 1.11 *In Figure 1.10 we show the TAs A_1, A_2 and $A_1||_L A_2$ with $L = \{a, b\}$. We assume invariants to be true for all states. States q_0, q_1 and q_2 of $A_1||_L A_2$ correspond to the pairs (r_0, u_0), (r_1, u_1) and (r_0, u_2), respectively.*

Proposition 1.15 *Given the TAs A_1 and A_2, $A_1||_L A_2$ is a TA for all $L \subseteq \Sigma$.*

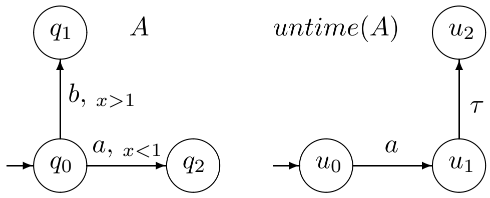

Figure 1.11: Example of *untime*(A).

1.3.5 From TAs to NSs

Given a TA A, we call $untime(A)$ the NS obtained as the *region automaton* of A. Intuitively, the region automaton (see [9]) is obtained by considering timed regions as states. Note that in the region automaton there is a step between regions R and R' with symbol a if and only if there is an admissible run $s \xrightarrow{t} s'' \xrightarrow{a} s'$ of the TA such that $t \in \mathbb{R}^{>0}$ and where $s \in R$ and $s' \in R'$. We consider the special symbol λ to label all the transitions of the NS $untime(A)$ arising from time steps of the TA A. Intuitively, since time steps are no more visible in the untimed setting, the invisible action τ is then used to hide all the λ steps.

Definition 1.20 *Given a TA $A = (\Sigma, X, Q, q_0, \delta, Inv)$, we define as $untime(A) = (\Sigma \cup \{\lambda\}, Q \times [V], (q_0, [v_0]), \delta')/\{\lambda\}$ the NS where:*

- $((q, [v]), \lambda, (q, [v'])) \in \delta'$ *iff* $v' = v + t$ *for some time* $t \in \mathbb{R}^{>0}$ *and* $v + t' \models Inv(q)\ \forall t' \in [0, t]$*;*
- $((q, [v]), a, (q', [v'])) \in \delta'$ *iff* $(q, a, \phi, B, q') \in \delta$*,* $v \models \phi \wedge Inv(q)$*,* $v' = v[B]$ *and* $v' \models Inv(q')$*.*

Example 1.12 *In Figure 1.11 we show the TA A, and its untimed version, the NS $untime(A)$. We assume $Inv(q_0) = x < 1$ and $Inv(q_1) = Inv(q_2) = true$. States u_0, u_1 and u_2 correspond, respectively, to the pairs $(q_0, [v_0])$, $(q_2, [v_0])$ and $(q_2, [v_1])$, where $[v_0] = \{v \mid v(x) < 1\}$ and $[v_1] = \{v \mid v(x) \geq 1\}$. In the figure we omitted self-loop transitions (u_i, τ, u_i) for any $i \in [0, 2]$. Note that, since $Inv(q_0) = x < 1$, transition labeled with b form q_0 to q_2 cannot be executed (it has constraint $x > 1$). Such a transition is lost in the NS $untime(A)$.*

Given an execution $\sigma = (q_0, v_0) \to \ldots \to (q_n, v_n)$ of A, with $[\sigma]$ we denote the execution $(q_0, [v_0]) \to \ldots \to (q_n, [v_n])$ of $untime(A)$.

As a consequence of Lemma 4.13 in [9] we have the following result.

Lemma 1.2 *Given a TA A, if σ is an execution fragment of A, then $[\sigma]$ is an execution fragment of $untime(A)$. Viceversa, if $[\sigma]$ is an execution fragment of $untime(A)$, then there exists $\sigma' \in [\sigma]$ such that σ' is an execution fragment of A.*

Relations among weak bisimulations

Lemma 1.3 *Given TAs A and A', $A \approx A' \Rightarrow untime(A) \approx untime(A')$.*

Proof. The implication holds by the construction of the region automaton and by Lemma 1.2. Actually, for each sequence of steps of a TA, there exists an analogous sequence for the NS obtained with $untime(A)$. Weak bisimulation is, therefore, preserved. □

Chapter 2

Probabilistic Timed Automata

The framework of Probabilistic Timed Automata (PTAs) allows the description of timed systems showing a probabilistic behavior, in an intuitive and succinct way. Therefore, within the framework of PTAs, where time and probabilities are taken into consideration, the modeler can describe, on a single model, different aspects of a system, and analyze real-time properties, performance and reliability properties (by using classical model checking techniques), and information flow security properties useful to detect both probabilistic and timing covert channels.

In Section 2.1 we introduce the model of PTAs. In Section 2.2 we define a notion of weak bisimulation for PTAs. In Section 2.3 we define operators of restriction, hiding and parallel composition for PTAs. In Section 2.4 we show how to remove probability and time from a PTA in order to obtain a PA or a TA.

2.1 The Formalism

We give a definition of Probabilistic Timed Automata inspired by definitions in [18, 85, 8, 94].

As for the model of TAs (see Section 1.3.1), let us assume again a set X of *clocks*. We denote with $v, v', \ldots$ valuations over X and with $\Phi(X)$ the set of constraints over X.

Definition 2.1 *A Probabilistic Timed Automaton (PTA) is a tuple* $A = (\Sigma, X, Q, q_0, \delta, Inv, \Pi)$, *where:*

- Σ *is a finite alphabet of actions.*
- X *is a finite set of positive real variables called clocks.*
- Q *is a finite set of states and* $q_0 \in Q$ *is the initial state.*
- $\delta \subseteq Q \times \Sigma \cup \{\tau\} \times \Phi(X) \times 2^X \times Q$ *is a finite set of transitions. The symbol* τ *represents the silent or internal move. For a state* q*, we denote with* $start(q)$ *the set of transitions with* q *as source state, i.e. the set* $\{(q_1, a, \phi, B, q_2) \in \delta \mid q_1 = q\}$.
- $Inv : Q \to \Phi(X)$ *is a function assigning a constraint* $\phi \in \Phi(X)$ *(called* state invariant*) to each state in* Q.
- $\Pi = \{\pi_1, \ldots, \pi_n\}$ *is a finite set of probability distributions as functions* $\pi_i : \delta \to [0, 1]$*, for any* $i = 1, \ldots, n$*, where* $\pi_i(e)$ *is the probability of performing transition* e *according to distribution* π_i*. We require that* $\sum_{e \in start(q)} \pi_i(e) \in \{0, 1\}$ *for any* i *and* q*. Moreover, we assume that for all* $\pi_i \in \Pi$ *there exists some* $e_j \in \delta$ *such that* $\pi_i(e_j) > 0$ *and, viceversa, for all* e_j *there exist some* π_i *such that* $\pi_i(e_j) > 0$.

2.1.1 Semantics of PTAs

A *configuration* of A is a pair (q, v), where $q \in Q$ is a state of A, and v is a valuation over X. The initial configuration of A is represented by $(q_0, \mathbf{0})$ and the set of all the configurations of A is denoted with $\mathcal{S}_A$.

There is a discrete *transition step* from a configuration $s_i = (q_i, v_i)$ to a configuration $s_j = (q_j, v_j)$ through action $(a, \pi) \in (\Sigma \cup \{\tau\}) \times \Pi$, written $s_i \xrightarrow{(a,\pi)} s_j$, if there is a transition $e = (q_i, a, \phi, B, q_j) \in \delta$ such that $\pi(e) > 0$, $v_i \models \phi \wedge Inv(q_i)$, $v_j = v_i[B]$ and $v_j \models Inv(q_j)$.

There is a continuous *time step* from a configuration $s_i = (q_i, v_i)$ to a configuration $s_j = (q_j, v_j)$ through time $t \in \mathbb{R}^{>0}$, written $s_i \xrightarrow{t} s_j$, if $q_j = q_i$, $v_j = (v_i + t)$ and $\forall t' \in [0, t]\ v_i + t' \models Inv(q_i)$.

Given a configuration $s = (q_i, v_i)$, $Adm(s) = \{(q_i, a, \phi, B, q) \in \delta \mid v_i \models \phi \wedge Inv(q_i) \wedge v_i[B] \models Inv(q)\}$ is the set of transitions executable by an automaton from configuration s; a transition in $Adm(s)$ is said to be *enabled* in s. Given two configurations $s_i = (q_i, v_i)$, $s_j = (q_j, v_j)$ and $(a, \pi) \in (\Sigma \cup \{\tau\}) \times \Pi$, $Adm(s_i, (a, \pi), s_j) = \{e = (q_i, a, \phi, B, q_j) \in \delta \mid \pi(e) > 0,\ v_i \models \phi \wedge Inv(q_i),\ v_j = v_i[B],\ v_j \models Inv(q_j)\}$ is the set of transitions leading from configuration s_i to configuration s_j through a transition step labeled with (a, π). A configuration $s = (q_i, v_i)$ is *terminal* iff $Adm(s') = \emptyset$ for all $s' = (q_i, v_i + t)$ where $t \in \mathbb{R}^{\geq 0}$; S_T denotes the set of terminal configurations.

For configurations s_i, s_j, and $\alpha \in ((\Sigma \cup \{\tau\}) \times \Pi) \cup \mathbb{R}^{>0}$, the probability $P(s_i, \alpha, s_j)$ of reaching configuration s_j from configuration s_i through a step labeled with α, is defined as

$$P(s_i, \alpha, s_j) = \begin{cases} \dfrac{\sum_{e \in Adm(s_i, \alpha, s_j)} \pi(e)}{\sum_{e \in Adm(s_i)} \pi(e)} & \text{if } s_i \xrightarrow{\alpha} s_j,\ \alpha = (a, \pi) \in (\Sigma \cup \{\tau\}) \times \Pi \\ 1 & \text{if } s_i \xrightarrow{\alpha} s_j,\ \alpha \in \mathbb{R}^{>0} \\ 0 & \text{if } s_i \not\xrightarrow{\alpha} s_j \end{cases}$$

The probability of executing a transition step from a configuration s is chosen according to the values returned by the function π among all the transitions enabled in s, while the probability of executing a time step labeled with $t \in \mathbb{R}^{>0}$ is set to the value 1. Intuitively, an automaton chooses nondeterministically a distribution π for executing a transition step (which is then selected probabilistically, according to π, among all the transitions enabled in s), or to let time pass performing a time step.

An *execution fragment* starting from s_0 is a finite sequence of timed and transition steps $\sigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots \xrightarrow{\alpha_k} s_k$, where $s_0, s_1, \ldots, s_k \in \mathcal{S}_A$ and $\alpha_i \in ((\Sigma \cup \{\tau\}) \times \Pi) \cup \mathbb{R}^{>0}$. $ExecFrag$ is the set of execution fragments and $ExecFrag(s)$ is the set of execution fragments starting from s. We define $last(\sigma) = s_k$ and $|\sigma| = k$. The execution fragment σ is called *maximal* iff $last(\sigma) \in S_T$.

For any $j < k$, σ^j is the sequence of steps $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_j} s_j$.

If $|\sigma| = 0$ we put $P(\sigma) = 1$, else, if $|\sigma| = k \geq 1$, we define $P(\sigma) = P(s_0, \alpha_1, s_1) \cdot \ldots \cdot P(s_{k-1}, \alpha_k, s_k)$.

An *execution* is either a maximal execution fragment or an infinite sequence $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$, where $s_0, s_1 \ldots \in \mathcal{S}_A$ and $\alpha_1, \alpha_2, \ldots \in ((\Sigma \cup \{\tau\}) \times \Pi) \cup \mathbb{R}^{>0}$. We denote with $Exec$ the set of executions and with $Exec(s)$ the set of executions starting from s. Finally, let $\sigma \uparrow$ denote the set of executions σ' such that $\sigma \leq_{prefix} \sigma'$, where *prefix* is the usual prefix relation over sequences.

Executions and execution fragments of a PTA arise by resolving both the nondeterministic and the probabilistic choices [85]. To resolve the nondeterministic choices of a PTA, we introduce *schedulers* of PTAs.

A *scheduler* of a PTA $A = (\Sigma, X, Q, q_0, \delta, Inv, \Pi)$ is a function F from $ExecFrag$ to $\Pi \cup \mathbb{R}^{>0}$. With $\mathcal{F}_A$ we denote the set of schedulers of A. Given a scheduler $F \in \mathcal{F}_A$ and an execution fragment σ, we assume that F is defined for σ iff $\exists s \in \mathcal{S}_A$ and $a \in \Sigma \cup \{\tau\}$ such that $last(\sigma) \xrightarrow{F(\sigma)} s$ if $F(\sigma) \in \mathbb{R}^{>0}$ or $last(\sigma) \xrightarrow{(a,F(\sigma))} s$ if $F(\sigma) \in \Pi$.

For a scheduler $F \in \mathcal{F}_A$ we define $ExecFrag^F$ (resp. $Exec^F$) as the set of execution fragments (resp. executions) $\sigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$ of A such that, for any $0 < i < |\sigma|$:

- if $\alpha_i \in \mathbb{R}^{>0}$, then $F(\sigma^{i-1}) = \alpha_i$;
- if $\alpha_i = (a, \pi)$, then $F(\sigma^{i-1}) = \pi$.

A scheduler should also respect the *nonZeno* condition of divergent times. Formally we have that for any infinite sequence $\sigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots$ in $Exec^F$ the sum $\sum_{\alpha_i \in \mathbb{R}^{>0}} \alpha_i$ diverges.

Assuming the basic notions of probability theory (see e.g. [59]) we define the probability space on the executions starting in a given configuration $s \in \mathcal{S}_A$ as follows. Given a scheduler F, let $Exec^F(s)$ be the set of executions starting in s, $ExecFrag^F(s)$ be the set of execution fragments starting in s, and $\Sigma^F_{Field}(s)$ be the smallest sigma field on $Exec^F(s)$ that contains the basic cylinders $\sigma\uparrow$, where $\sigma \in ExecFrag^F(s)$. The probability measure $Prob^F$ is the unique measure on $\Sigma^F_{Field}(s)$ such that $Prob^F(\sigma\uparrow) = P(\sigma)$.

Let A be a PTA, $F \in \mathcal{F}_A$, $\hat{\alpha}$ stand for a if $\alpha = (a, \pi)$ or $\alpha = a \in \mathbb{R}^{>0}$, $s \in \mathcal{S}_A$ and $\mathcal{C} \subseteq \mathcal{S}_A$. Given the set $Exec^F(\tau^*\hat{\alpha}, \mathcal{C})$ of executions that lead to a configuration in $\mathcal{C}$ via a sequence belonging to the set of sequences $\tau^*\hat{\alpha}$, we define $Exec^F(s, \tau^*\hat{\alpha}, \mathcal{C}) = Exec^F(\tau^*\hat{\alpha}, \mathcal{C}) \cap Exec^F(s)$. Given a scheduler F, we define the probability $Prob^F(s, \tau^*\hat{\alpha}, \mathcal{C}) = Prob^F(Exec^F(s, \tau^*\hat{\alpha}, \mathcal{C}))$. The next proposition derives directly by this definition.

Proposition 2.1 *It holds that* $Prob^F_A(s, \tau^*\hat{\alpha}, \mathcal{C}) =$

$$\begin{cases} 1 & if\ \hat{\alpha} = \tau \wedge s \in \mathcal{C} \\ \sum_{q \in \mathcal{S}_A} Prob^F(s, \tau, q) \cdot Prob^F(q, \tau^*, \mathcal{C}) & if\ \hat{\alpha} = \tau \wedge s \notin \mathcal{C} \\ \sum_{q \in \mathcal{S}_A} Prob^F(s, \tau, q) \cdot Prob^F(q, \tau^*\alpha, \mathcal{C}) + Prob^F(s, \alpha, \mathcal{C}) & if\ \hat{\alpha} \neq \tau \end{cases}$$

Example 2.1 *In Figure 2.1 we show an example of PTA with* $\Pi = \{\pi\}$ *and* $Inv(q_i) = true$ *for all* $i \in [0, 3]$.

Intuitively, from the initial state q_0*, the PTA may nondeterministically choose whether to perform some time step and update the value of clock* x *or to perform, probabilistically, transitions* e_1*,* e_2 *or* e_3 *with probabilities* $\frac{1}{6}$*,* $\frac{1}{3}$ *and* $\frac{1}{2}$*, respectively.*

If some time steps are performed from state q_0*, such that values of clock* x *becomes greater than* 5*, then transition labeled with* b *cannot be performed any more and probabilities of performing the other transitions should be redistributed.*

Examples of executions of the PTA in Figure 2.1 are $\sigma_1 = (q_0, 0) \xrightarrow{9.7} (q_0, 9.7) \xrightarrow{(a,\pi)} (q_2, 0) \xrightarrow{3} (q_2, 3)$ *and* $\sigma_2 = (q_0, 0) \xrightarrow{3} (q_0, 3) \xrightarrow{(b,\pi)} (q_1, 3) \xrightarrow{1.2} (q_0, 4.2)$ *with* $P(\sigma_1) = \frac{2}{5}$ *and* $P(\sigma_2) = \frac{1}{6}$*, where* (q, t) *represents the configuration composed by the state* q *and the valuation* v *such that* $v(x) = t$.

2.1.2 Regions of PTAs

As we have seen, configurations reachable by a PTA depend on the constraints and updates over the set of clocks. Actually, probability does only affect the frequencies with which those configurations are reached, therefore we have exactly the same concepts and properties given for regions of TAs in Section 1.3.2 also for regions of PTAs.

2.2 Weak bisimulation

We introduce a notion of weak bisimulation for PTAs.

As already mentioned, weak internal transitions $s \stackrel{\tau}{\Longrightarrow} s'$ are replaced by the probability $Prob(s, \tau^*, s')$ of reaching configuration s' from s via internal moves. Similarly, for visible actions α, transitions $\stackrel{\alpha}{\Longrightarrow}$ are replaced by the probability $Prob(s, \tau^*\alpha, s')$. The next definition is obtained by adapting Definition 1.8 to the model of PTAs.

$$
\begin{array}{ll}
e_1 = (q_0, b, 0 \leq x \leq 5, \emptyset, q_1) & \pi(e_1) = \frac{1}{6} \\
e_2 = (q_0, a, true, \{x\}, q_2) & \pi(e_2) = \frac{1}{3} \\
e_3 = (q_0, a, true, \emptyset, q_3) & \pi(e_3) = \frac{1}{2}
\end{array}
$$

$\sum_{e \in \delta(q_0)} \pi(e) = 1$

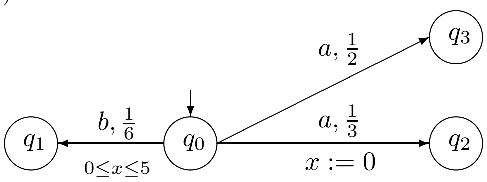

Figure 2.1: Example of PTA.

Definition 2.2 *Let $A = (\Sigma, X, Q, q_0, \delta, Inv, \Pi)$ be a PTA. A* weak bisimulation *on A is an equivalence relation $\mathcal{R}$ on $\mathcal{S}_A$ such that, for all $(s, s') \in \mathcal{R}$, $\mathcal{C} \in \mathcal{S}_A/\mathcal{R}$ and schedulers F, there exists a scheduler F' such that*

$$Prob^F(s, \tau^*\alpha, \mathcal{C}) = Prob^{F'}(s', \tau^*\alpha, \mathcal{C}) \qquad \forall \alpha \in \Sigma \cup \{\tau\} \cup \mathbb{R}^{>0}$$

and vice versa.
Two configurations s, s' are called weakly bisimilar *on A (denoted $s \approx_A s'$) iff $(s, s') \in \mathcal{R}$ for some weak bisimulation $\mathcal{R}$.*

Definition 2.3 *Two PTAs $A = (\Sigma, X, Q, q_0, \delta, Inv, \Pi)$ and $A' = (\Sigma', X', Q', q_0', \delta', Inv, \Pi')$ such that $Q \cap Q' = \emptyset$ and $X \cap X' = \emptyset$ are called weakly bisimilar (denoted by $A \approx A'$) if, given the PTA $\hat{A} = (\Sigma \cup \Sigma', X \cup X', Q \cup Q', q_0, \delta \cup \delta', \hat{Inv}, \hat{\Pi})$, where, for each couple $(\pi, \pi') \in \Pi \times \Pi'$, $\hat{\pi} \in \hat{\Pi}$ such that*

$$\hat{\pi}(e) = \begin{cases} \pi(e) & if\ e \in \delta \\ \pi'(e) & if\ e \in \delta' \end{cases}$$

and

$$\hat{Inv}(q) = \begin{cases} Inv(q) & if\ q \in Q \\ Inv'(q) & if\ q \in Q' \end{cases}$$

it holds $(q_0, \mathbf{0}) \approx_{\hat{A}} (q_0', \mathbf{0})$, where the valuation $\mathbf{0}$ is defined over all clocks of the set $X \cup X'$.

Example 2.2 *Consider the PTAs of Figure 2.2. Intuitively, they both can perform action a or action b after 5 time units, with probability $\frac{1}{2}$. By applying the notion of weak bisimulation introduced above, the two PTAs turn out to be equivalent. Given the automaton $\hat{A}$, built from the automata A_1 and A_2 by following the procedure described in Definition 2.3, with $R_{\hat{A}}$ we denote the set of regions of $\hat{A}$ such that $R_{\hat{A}} = \{R_1 = (q_0, x < 5), R_2 = (q_0, x = 5), R_3 = (q_0, x > 5), R_4 = (q_1, x \geq 0), R_5 = (q_2, x \geq 0), R_6 = (r_0, z < 5), R_7 = (r_0, z = 5), R_8 = (r_0, z > 5), R_9 = (r_1, z \geq 0), R_{10} = (r_2, z \geq 0)\}$. With $\mathcal{R}$ we denote the equivalence relation on $\mathcal{S}_{\hat{A}}$ such that $\mathcal{S}_{\hat{A}}/\mathcal{R} = \{C_1, C_2, C_3\}$, where $C_1 = \{R_1, R_6\}$, $C_2 = \{R_2, R_7\}$ and $C_3 = \{R_3, R_4, R_5, R_8, R_9, R_{10}\}$. In the following we assume $\alpha \in \{a, b, \tau\} \cup \mathbb{R}^{>0}$, where α is chosen according to a scheduler F in case $\alpha \in \mathbb{R}^{>0}$. For each configuration $s \in C_1$ and for each scheduler F we have $Prob^F(s, \tau^*, C_1) = 1$, $Prob^F(s, \tau^*\alpha, C_1) = 1$ if $\alpha < 5 - x$, $Prob^F(s, \tau^*\alpha, C_2) = 1$ if $\alpha = 5 - x$, $Prob^F(s, \tau^*\alpha, C_3) = 1$ if $\alpha > 5 - x$, and $Prob^F(s, \tau^*\alpha, C_3) = \frac{1}{2}$ if $\alpha = a$ or $\alpha = b$. For each $s \in C_2$ we have $Prob^F(s, \tau^*, C_2) = 1$, $Prob^F(s, \tau^*\alpha, C_1) = 0$, $Prob^F(s, \tau^*\alpha, C_3) = \frac{1}{2}$ if $\alpha = a$ or $\alpha = b$, $Prob^F(s, \tau^*\alpha, C_3) = 1$ if $\alpha \in \mathbb{R}^{>0}$. Finally, for each configuration $s \in C_3$ we have $Prob^F(s, \tau^*, C_3) = 1$, $Prob^F(s, \tau^*\alpha, C_3) = 1$ if $\alpha \in \mathbb{R}^{>0}$, $Prob^F(s, \tau^*\alpha, C_2) = 0$ and $Prob^F(s, \tau^*\alpha, C_1) = 0$. Hence, $\mathcal{R}$ is a weak bisimulation on $\hat{A}$ and, since $(q_0, \mathbf{0})$ and $(r_0, \mathbf{0})$ are in the same class, A_1 and A_2 are weak bisimilar.*

In Chapter 3 we develop a detailed proof for the decidability of weak bisimulation for PTAs.

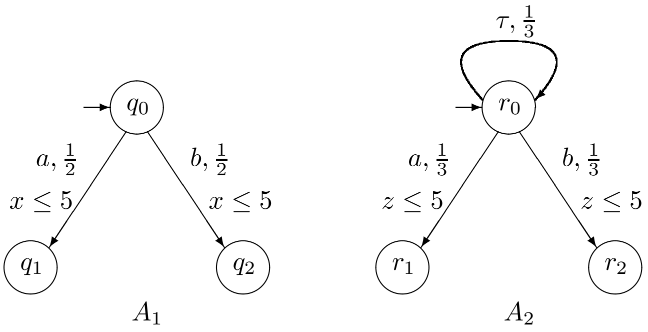

Figure 2.2: $A_1 \approx A_2$.

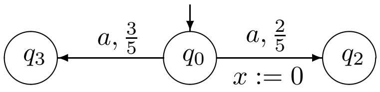

Figure 2.3: Example of restriction for PTAs.

2.3 Operators for Probabilistic Timed Automata

We define operations of *restriction*, *hiding* and *parallel composition* on PTAs.

2.3.1 Restriction

We assume a PTA $A = (\Sigma, X, Q, q_0, \delta, Inv, \Pi)$ and a set $L \subseteq \Sigma$ of actions.

Definition 2.4 *The* restriction *of a PTA A with respect to the set of actions L is $A \setminus L = (\Sigma, X, Q, q_0, \delta', Inv, \Pi')$, where:*

- $\delta' = \{(q, a, \phi, B, q') \in \delta \mid a \notin L\}$.
- $\pi' \in \Pi'$ *iff* $\pi \in \Pi$ *where, for all* $e = (q, a, \phi, B, q') \in \delta'$, $\pi'(e) = \frac{\pi(e)}{\sum_{e' \in \delta' \cap start(q)} \pi(e')}$.

The second condition is assumed in order to normalize the probability of each transition according to the ones remaining after the restriction. Thanks to this rule the condition $\sum_{e \in start(q)} \pi'(e) \in \{0, 1\}$ continues to be true for each state q of $A \setminus L$.

Example 2.3 *Let A be the PTA in Figure 2.1 and $L = \{b\}$. In Figure 2.3 we show the PTA $A \setminus L$, where every transition with label b is prevented and probabilities are redistributed.*

Proposition 2.2 *Given a PTA A, $A \setminus L$ is a PTA for all $L \subseteq \Sigma$.*

2.3.2 Hiding

Again, we assume a PTA $A = (\Sigma, X, Q, q_0, \delta, Inv, \Pi)$ and a set $L \subseteq \Sigma$ of actions.

The *hiding* of a transition $e = (q, a, \phi, B, q')$ with respect to the set of actions L (written e/L) is defined as:

$$e/L = \begin{cases} e & if\ a \notin L \\ (q, \tau, \phi, B, q') & if\ a \in L \end{cases}$$

Definition 2.5 *The hiding of a PTA A with respect to the set of actions L is given by $A/L = (\Sigma, X, Q, q_0, \delta', Inv, \Pi')$, where $\delta' = \{e/L \mid e \in \delta\}$, and $\Pi' = \{\pi' | \exists \pi \in \Pi. \forall e' \in \delta'\ \pi'(e') = \sum_{e \in \delta : e/L = e'} \pi(e)\}$.*

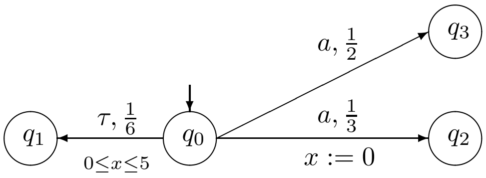

Figure 2.4: Example of hiding for PTAs.

Example 2.4 *Let A be the PTA in Figure 2.1 and $L = \{b\}$. In Figure 2.3 we show the PTA A/L, where every transition with label b is replaced with a transition labeled with τ.*

Proposition 2.3 *Given a PTA A, A/L is a PTA for all $L \subseteq \Sigma$.*

2.3.3 Parallel Composition

The definition of parallel composition for PTAs we use here is slightly different from the one in [86], in order to result in fully probabilistic discrete transitions.

Assume PTAs $A_1 = (\Sigma, X_1, Q_1, r_0, \delta_1, Inv_1, \Pi_1)$ and $A_2 = (\Sigma, X_2, Q_2, u_0, \delta_2, Inv_2, \Pi_2)$ with disjoint sets of states and clocks ($Q_1 \cap Q_2 = \emptyset$, $X_1 \cap X_2 = \emptyset$). We also assume a set $L \subseteq \Sigma$ of synchronization actions. Finally, for $i \in \{1,2\}$, given a transition $e = (q, a, \phi, B, q') \in \delta_i$, and a probability distribution $\pi_i \in \Pi_i$ with $\pi_{i_a}(e)$ we denote the normalized probability of executing transition e with respect to all other transitions starting from q and labeled with a, i.e. $\pi_{i_a}(e) = \frac{\pi_i(e)}{\sum_{e' \in start_i^a(q)} \pi_i(e')}$, where $start_i^a(q)$ denotes the set of transitions in δ_i with q as source state and a as labeling action, i.e. the set $\{(q_1, a', \phi, B, q_2) \in \delta_i \mid q_1 = q \wedge a' = a\}$.

Definition 2.6 *The* parallel composition *of two PTAs A_1 and A_2, with respect to the synchronization set L and the advancing speed parameter $p \in]0,1[$, is defined as the automaton $A_1||_L^p A_2 = (\Sigma, X, Q, (r_0, u_0), \delta, Inv, \Pi)$. The set $Q = Q_1 \times Q_2$ of states of $A_1||_L^p A_2$ is given by the cartesian product of the states of the two automata A_1 and A_2, while the set of clocks $X = X_1 \cup X_2$ is given by the union of X_1 and X_2. State invariants are defined as $Inv : Q_1 \times Q_2 \to \Phi(X_1 \cup X_2)$ where, for any $r \in Q_1$ and $u \in Q_2$ such that $Inv_1(r) = \phi_1$ and $Inv_2(u) = \phi_2$ we have that $Inv(r,u) = \phi_1 \wedge \phi_2$. Given a state (r,u) of $A_1||_L^p A_2$ there is a probability distribution $\pi \in \Pi$ for any two probability distributions $\pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2$. In particular, $\delta = S_1 \cup S_2 \cup \bigcup_{a \in L} S_3^a$ where $S_1 = \{((r,u), b, \phi, B, (r', u)) \mid (r, b, \phi, B, r') \in \delta_1, u \in Q_2, b \notin L\}$, $S_2 = \{((r,u), b, \phi, B, (r, u')) \mid (u, b, \phi, B, u') \in \delta_2, r \in Q_1, b \notin L\}$ and, for any $a \in L$, $S_3^a = \{((r,u), \tau, \phi_1 \wedge \phi_2, B_1 \cup B_2, (r', u')) \mid (r, a, \phi_1, B_1, r') \in \delta_1, (u, a, \phi_2, B_2, u') \in \delta_2\}$. Moreover, for any pair $\pi_1 \in \Pi_1$, $\pi_2 \in \Pi_2$, there exists $\pi \in \Pi$ such that, for all $e = (q, a, \phi, B, q') \in \delta$, it holds that $\pi(e) = \frac{f(e)}{\sum_{e' \in \delta \cap start(q)} f(e')}$ where*

$$f(e) = \begin{cases} \pi_1(e) \cdot p & \text{if } e \in S_1 \\ \pi_2(e) \cdot (1-p) & \text{if } e \in S_2 \\ \pi_1(e) \cdot p \cdot \pi_{2_a}(e) + \pi_2(e) \cdot (1-p) \cdot \pi_{1_a}(e) & \text{if } e \in \bigcup_{a \in L} S_3^a \end{cases}$$

Note that, given such a definition of parallel composition, automata A_1 and A_2 are prevented of performing transitions with label in L without synchronizing. Moreover, whenever A_1 and A_2 synchronize they give rise to an internal action τ. Also note that, chosen a transition e_1 (e_2) with label $a \in L$ of automaton A_1 (A_2) the transition e_2 (e_1) of A_2 (A_1) that synchronizes with e_1 (e_2) is chosen according to the probability $\pi_{2_a}(e_2)$ ($\pi_{1_a}(e_1)$) normalized with respect to all the other transitions labeled with a. Besides, according to Definition 2.1, given the parallel composition defined above, it holds that $\sum_{e \in start(q)} \pi(e) \in \{0,1\}$ for each state q of $A_1||_L^p A_2$. This is done

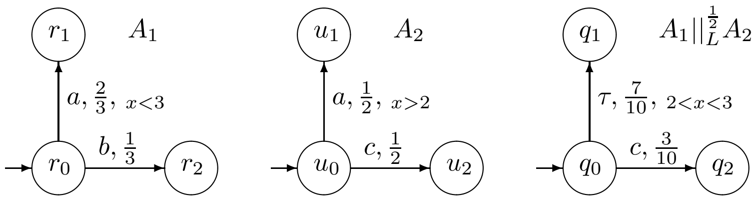

Figure 2.5: Example of parallel composition for PTAs.

due to the last rule, that uses the auxiliary structure $f(e)$ in order to compute the normalized probabilities in π. In fact, transitions of the single automata A_1 and A_2 with label $a \in L$ are not allowed to be performed without synchronization, and therefore they are lost in the compound system together with their probabilities (and therefore probabilities of the compound system must be renormalized).

Example 2.5 *In Figure 2.5 we show the PAs A_1, A_2 and $A_1||_L^{\frac{1}{2}}A_2$ with $L = \{a, b\}$. States q_0, q_1 and q_2 of $A_1||_L A_2$ correspond to the pairs (r_0, u_0), (r_1, u_1) and (r_0, u_2), respectively.*

Proposition 2.4 *Given the PTAs A_1 and A_2, $A_1||_L^p A_2$ is a PTA for all $p \in]0, 1[$ and $L \subseteq \Sigma$.*

2.4 Removing Time and Probability from PTAs

We define operators to remove probability and time from a given PTA, and we show that these operators preserve weak bisimulation.

2.4.1 From PTAs to TAs

Given a PTA A, we call *unprob*(A) the TA obtained from A by simply removing probabilities from A. This can be done since we assumed that for each transition of A there is at least a probability distribution which assigns to such a transition a probability greater than 0 (see Definition 2.1).

Definition 2.7 *Given a PTA $A = (\Sigma, X, Q, q_0, \delta, Inv, \Pi)$, unprob$(A) = (\Sigma, X, Q, q_0, \delta, Inv)$.*

Example 2.6 *Let A be the PTA in Figure 2.1. If we remove probabilities from A the TA unprob(A) can be found in Figure 1.9.*

2.4.2 From PTAs to PAs

Given a PTA A, we call *untime*(A) the PA obtained as the *region automaton* of A, with probability functions chosen according to Π. Intuitively, the region automaton (see [9]) is obtained by considering timed regions as states. Note that in the region automaton there is a step between regions R and R' with symbol (a, π) if and only if there is an admissible run $s \xrightarrow{t} s'' \xrightarrow{(a,\pi)} s'$ of the PTA such that $t \in \mathbb{R}^{>0}$ and where $s \in R$ and $s' \in R'$. We consider the special symbol λ to label all the transitions of the PA *untime*(A) arising from time steps of the PTA A. Intuitively, since time steps are no more visible in the untimed setting, the invisible action τ is then used to hide all the λ steps.

Definition 2.8 *Given a PTA $A = (\Sigma, X, Q, q_0, \delta, Inv, \Pi)$, we define the PA untime$(A) = (\Sigma \cup \{\lambda\}, Q \times [V], (q_0, [v_0]), \delta', \Pi')/\{\lambda\}$ where:*

- *$((q, [v]), \lambda, (q, [v'])) \in \delta'$ iff $v' = v + t$ for some time $t \in \mathbb{R}^{>0}$ and $v + t' \models Inv(q)$ $\forall t' \in [0, t]$;*
- *$((q, [v]), a, (q', [v'])) \in \delta'$ iff $(q, a, \phi, B, q') \in \delta$, $v \models \phi \wedge Inv(q)$, $v' = v[B]$ and $v' \models Inv(q')$;*

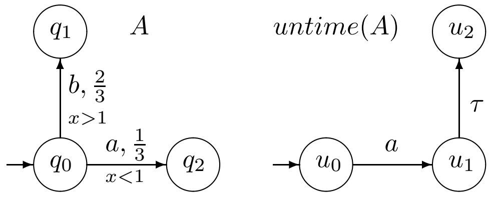

Figure 2.6: Example of *untime*(A).

- *For any* λ*-transition* $e' = ((q,[v]),\lambda,(q',[v'])) \in \delta'$ *there exists* $\pi' \in \Pi'$ *such that* $\pi'(e') = 1$. *Moreover, for all* $\pi \in \Pi$ *there exists* $\pi' \in \Pi'$ *such that,* $\pi'(e') = \frac{\sum_{e_i \in S} \pi(e_i)}{\sum_{e_j \in Adm((q,v))} \pi(e_j)}$ *where* $e' = ((q,[v]),a,(q',[v'])) \in \delta'$ *and* $S = \{(q,a,\phi,B,q') \in \delta \mid v \models \phi,\ \ v' = v[B]\}$.

Example 2.7 *In Figure 2.6 we show the PTA A, and its untimed version, the PA untime(A). We assume* $Inv(q_0) = x < 1$ *and* $Inv(q_1) = Inv(q_2) = true$. *States* u_0, u_1 *and* u_2 *correspond, respectively, to the pairs* $(q_0,[v_0])$, $(q_2,[v_0])$ *and* $(q_2,[v_1])$, *where* $[v_0] = \{v \mid v(x) < 1\}$ *and* $[v_1] = \{v \mid v(x) \geq 1\}$. *In the figure we omitted self-loop transitions* $e_i(u_i,\tau,u_i)$ *for any* $i \in [0,2]$. *Note that, since* $Inv(q_0) = x < 1$, *transition labeled with b form* q_0 *to* q_2 *cannot be executed (it has constraint* $x > 1$*). Such a transition is lost in the PA untime(A) and probabilities are redistributed. If* $e_3 = (u_0,a,u_1)$ *and* $e_4 = (u_1,\tau,u_2)$ *we have two probability distributions, namely* π_1 *and* π_2, *for the PA untime(A). In particular, we have* $\pi_1(e_i) = 1$ *for* $i \in [0,2]$ *and* $\pi_1(e_i) = 0$ *for* $i \in [3,4]$ *and* $\pi_2(e_i) = 0$ *for* $i \in [0,2]$ *and* $\pi_2(e_i) = 1$ *for* $i \in [3,4]$.

Given an execution $\sigma = (q_0,v_0) \to \ldots \to (q_n,v_n)$ of A, with $[\sigma]$ we denote the execution $(q_0,[v_0]) \to \ldots \to (q_n,[v_n])$ of *untime*(A).

As a consequence of Lemma 4.13 in [9] and Lemma 4.8 in [90] we have the following result.

Lemma 2.1 *Given a PTA A, we have that, for any scheduler F of A, there exists a scheduler* F' *of* $untime(A)$ *such that, for any* $\sigma \in ExecFrag_A^F$, $Prob_A^F(\sigma) = Prob_{untime(A)}^{F'}([\sigma])$. *Viceversa, for any scheduler F of* $untime(A)$, *there exists a scheduler* F' *of A such that, for any* $[\sigma] \in ExecFrag_{untime(A)}^F$, $Prob_{untime(A)}^F([\sigma]) = Prob_A^{F'}(\sigma')$ *for some* $\sigma' \in [\sigma]$.

The following proposition states that given a PTA, we may obtain a NS by removing time and probability in two successive steps, no matter about the order.

Proposition 2.5 *Given a PTA A,* $unprob(untime(A)) = untime(unprob(A))$.

Proof. The proof derives trivially by the construction of the region automaton in the *untime* operators. Such an operator, in fact, is exactly the same for PTAs and TAs (except for the last point of Definition 2.8, were probability distributions are considered). Now, the NS obtained does not change if we remove probabilities through the *unprob* operator either before or after applying the *untime* construction. This holds since we assumed that for each transition of A there is at least a probability distribution which assigns to such a transition a probability greater than 0 (see Definition 2.1). □

2.4.3 Relations among weak bisimulations

Lemma 2.2 *The following statements hold:*

1. *Given PTAs A and* A', $A \approx A' \Rightarrow unprob(A) \approx unprob(A')$.

2. *Given PTAs A and A', $A \approx A' \Rightarrow untime(A) \approx untime(A')$.*

Proof. For case 1, let us assume $A = (\Sigma, X, Q, q_0, \delta, Inv, \Pi)$, $A' = (\Sigma', X', Q', q'_0, \delta', Inv, \Pi')$ and $\hat{A}$ constructed as in Definition 2.3. Since $A \approx A'$ for a weak bisimulation $\mathcal{R}$, we have that for all $(s,r) \in \mathcal{R}$, $\mathcal{C} \in \mathcal{S}_{\hat{A}}/\mathcal{R}$ and schedulers F, there exists a scheduler F' such that $Prob^F_{\hat{A}}(s, \tau^*\alpha, \mathcal{C}) = Prob^{F'}_{\hat{A}}(r, \tau^*\alpha, \mathcal{C})\ \forall \alpha \in \Sigma \cup \{\tau\} \cup \mathbb{R}^{>0}$. Now, if $Prob^F_{\hat{A}}(s, \alpha, s') > 0$ for some $s' \in \mathcal{C}$ there exists a configuration r' and a scheduler F' such that $Prob^{F'}_{\hat{A}}(r, \tau^*\alpha, r') = Prob^F_{\hat{A}}(s, \alpha, s') > 0$. Therefore if $s \xrightarrow{\alpha} s'$, then there exists r' such that $r \stackrel{\alpha}{\Longrightarrow} r'$ and, since s' and r' are in the same equivalence class, there exists also a bisimulation $\mathcal{R}'$ on $\mathcal{S}_{\hat{A}_{np}}$ such that $(s', r') \in \mathcal{R}'$, where $\hat{A}_{np}$ is constructed as in Definition 1.16 starting from $unprob(A)$ and $unprob(A')$. The same holds if we exchange the roles of s and r.

For cases 2, the implication holds by the construction of the region automaton and by Lemmata 2.1 and 1.2. Actually, for each run of a PTA (or TA), there exists an analogous run for the PA (or NS) obtained with $untime(A)$. Weak bisimulations are, therefore, preserved. □

Chapter 3

Decidability of Weak Bisimulation for PTA

Bisimilarity is widely accepted as the finest extensional behavioral equivalence one would want to impose on systems, and it may be used to verify a property of a system by assessing the bisimilarity of the system considered, with a system one knows to enjoy the property. Most of the times one wants to assess bisimilarity without taking into account system internal moves. This is called, as we have seen, *weak bisimulation.*

In this chapter we develop an algorithm which computes the classes of the weak bisimulation equivalence and decides if two configurations are weak bisimilar by checking that they are in the same class. To do this, we have to verify the condition of Definition 2.2.

We give the algorithm and the proof for a restricted version of PTAs which assumes that state invariants are *true* for all states and that the set of probability functions Π contains one only function π ($|\Pi| = 1$). Therefore, in this chapter, a PTA is a tuple $A = (\Sigma, X, Q, q_0, \delta, \pi)$. It will be easy to extend the algorithm to deal with general PTAs.

In Section 3.1 we briefly introduce some problems that arise when dealing with configurations of a PTA. In Section 3.2 we define classes of configurations. In Section 3.3 we introduce the algorithm *Clean* in order to remove the relations between configurations from which it is not possible to perform the same time step. In Section 3.4 we propose a method to remove relations from which it is not possible to perform the same probabilistic discrete step. In Section 3.5 we give the algorithm for computing the set of classes of bisimilar configurations.

3.1 The Context

In order to describe pairs of configurations $((q, v), (q', v'))$ within a certain equivalence relation we resort to disjoint sets of clocks. We use X to represent the evaluations of the clocks in X for configuration (q, v), and $\overline{X}$ to represent the evaluations for configuration (q', v') where $\overline{X} = \{\overline{x} \mid x \in X\}$.

Example 3.1 *Consider the PTAs in Figure 3.1. If we do not consider the time constraints, the probability of reaching q_4 from the state q_2 is $\frac{1}{2}$. However, in the timed setting, we observe that in state q_2, when clock x has value smaller then 3, the automaton may execute both transitions with probability $\frac{1}{2}$. Otherwise, if clock x has value greater than 3, the transition labeled with a cannot be executed, and hence the probability has to be redistributed; in such a case the probability of executing the transition with label a is 0, whereas the transition labeled with b gets probability 1. Therefore, we need to consider the different cases in which a subset of transitions are enabled or not.*

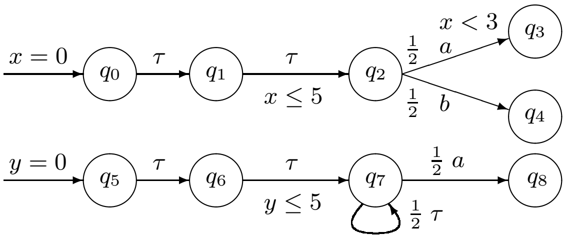

Figure 3.1: An example.

Moreover, we might consider to use the algorithm for the untimed version on the region graphs of the two automata, i.e. the graph of regions resulting by applying the predecessor operator. However, this is not a good solution; in fact, if we consider the clock zone reached in state q_1 we have $x \geq 0$ and in state q_6 we have $y \geq 0$. Let us suppose that one wants to compare the probability of reaching q_2 from q_1 with the probability of reaching q_7 from q_6. Now, we must check the two probabilities for each time $t \in \mathbb{R}^{\geq 0}$, and these are equal for every time if and only if $x = y$. This means that we cannot consider the clocks separately, but we must have formulae on all the pairs of states.

Since we have to check also the bisimilarity for states of the same automaton, as an example q_0 and q_1, we have to consider formulae that express conditions on the value of clocks at state q_0 together with the value of clocks at state q_1. Therefore, we should represent a set of bisimilar configurations (called class*) by a set of triples. As an example the triple $(q_0, q_1, x < \overline{x})$ means that the values of clocks x at state q_0 are less than those of clock x at state q_1.*

For deciding our notion of weak bisimulation, we follow the classical approach of refining relations between configurations ([15, 28]). In particular, starting from the initial relation where all configurations of a PTA are equivalent, we stepwise specialize relations until we obtain a weak bisimulation.

At each step we refine the set of classes by deleting the relations between configurations s^1 and s^2 which do not satisfy the condition that, for all schedulers F, there exists a scheduler F' such that $Prob^F(s^1, \tau^*\alpha, \mathcal{C}) = Prob^{F'}(s^2, \tau^*\alpha, \mathcal{C})$ and vice versa.

To compute the probabilities $Prob^F(s^1, \tau^*\alpha, \mathcal{C})$ and $Prob^{F'}(s^2, \tau^*\alpha, \mathcal{C})$ we construct two PAs A^1 and A^2. PAs A^i has triples (q, q', ψ) as states, while transitions (computed by means of predecessors operators) reflect the possibility and the probability of performing certain steps from configurations reached starting from s^i, for $i = 1, 2$.

When α is a time, we require that the unsatisfiability of bisimulation requirements is not caused by the fact that a time step α cannot be performed. Actually, even if a time step α can be performed from s^1 but not from s^2, both A^1 and A^2 do not have a transition representing that step. This because, since states of A^1 and A^2 are triples, steps from s^1 are affected by configuration s^2 and hence there is no time successor from the triple representing s^1 and s^2. This does not hold for transition steps since the guard of a transition triggered from s^1 is not affected by the valuation in s^2. We solve this problem by defining an algorithm *Clean* that refines the classes in such a manner that the unsatisfiability of bisimulation requirements for $\alpha \in \mathbb{R}^{>0}$, is not caused by the fact that we cannot perform α. Intuitively, algorithm *Clean* removes the relations between configurations from which it is not possible to perform the same time step to reach a certain class of bisimilar configurations.

The correctness of the methodology we propose is set up on the following inductive definition of equivalence relations $\sim_n$ on $\mathcal{S}_A$.

Definition 3.1 *Let $A = (\Sigma, X, Q, q_0, \delta, \pi)$ be a PTA. We set $\sim_0 = \mathcal{S}_A \times \mathcal{S}_A$ and, for $n = 0, 1, \ldots$, $s \sim_{n+1} s'$ iff $\forall \alpha \in \Sigma \cup \{\tau\} \cup \mathbb{R}^{>0}$ $\forall \mathcal{C} \in \mathcal{S}_A / \sim_n$ it holds that for all schedulers F there exists a scheduler F' such that $Prob^F(s, \tau^*\alpha, \mathcal{C}) = Prob^{F'}(s', \tau^*\alpha, \mathcal{C})$ and vice versa.*

The next lemma allows us to define an algorithm for computing weak bisimulation equivalence classes by following the technique discussed above.

Lemma 3.1 *Let $A = (\Sigma, X, Q, q_0, \delta, \pi)$ be a PTA and $s, s' \in \mathcal{S}_A$. Then, $s \approx s' \Leftrightarrow \forall n \geq 0 \quad s \sim_n s'$.*

Proof. Let $\sim' = \bigcap_{n \geq 0} \sim_n$. We have to show that $\approx = \sim'$. It easy to see that $\sim'$ is an equivalence relation. By induction on n we can show that $\sim_0 \supseteq \sim_1 \supseteq \ldots \supseteq \approx$. Hence, $\sim' \supseteq \approx$. In order to show that $\sim' \subseteq \approx$ we prove that $\sim'$ is a weak bisimulation.

For each $n \geq 0$ and each $B \in \mathcal{S}_A / \sim'$, there exists a unique element $B_n \in \mathcal{S}_A / \sim_n$ with $B \subseteq B_n$. Then, $B_0 = \mathcal{S}_A \supseteq B_1 \supseteq B_2 \supseteq \ldots$ and $B = \bigcap_{n \geq 0} B_n$.

Claim 1: We want to prove that, for all schedulers F of A, if $Prob^F(s, \tau^*\alpha, B) > 0$ and $B \in \mathcal{S}_A / \sim'$, then $Prob^F(s, \tau^*\alpha, B) = inf_{n \geq 0} Prob^F(s, \tau^*\alpha, B_n)$. For short, let us call $P[B_n]$ the probability $Prob^F(s, \tau^*\alpha, B_n)$. Since $B = \bigcap_{n \geq 0} B_n$ and $B_n \supseteq B_{n+1}$, we have $1 = P[B_0] \geq P[B_1] \geq \ldots \geq P[B_n]$. We put $r = inf_{n \geq 0} P[B_n]$. Clearly $r \geq P[B]$. We suppose, by contradiction, that $r > P[B]$. Let $\Delta = r - P[B]$, then $\Delta > 0$. There exists a subset X of $\mathcal{S}_A \setminus B$ such that $P[Y] < \Delta$ where $Y = \mathcal{S}_A \setminus (B \cup X)$. For all $n \geq 0$, $B_n = B \cup (Y \cap B_n) \cup (X \cap B_n)$. The sets B, $Y \cap B_n$ and $X \cap B_n$ are pairwise disjoint. Hence, $P[B_n] = P[B] + P[Y \cap B_n] + P[X \cap B_n] < P[B] + \Delta + P[X \cap B_n] = r + P[X \cap B_n]$. Since $r \leq P[B_n]$ we get $X \cap B_n \neq \emptyset$. As a consequence $X \cap B \neq \emptyset$, giving a contradiction.

Claim 2: We want to prove that $\sim'$ is a weak bisimulation. Let $s \sim' s'$ and $Prob^F(s, \tau^*\alpha, C) > 0$ for some scheduler F and $C \subseteq \mathcal{S}_A$. By Claim 1 it suffices to show that there exists a scheduler F' such that $Prob^F(s', \tau^*\alpha, C) = Prob^{F'}(s, \tau^*\alpha, C)$, for all $n \geq 1$ and $C \in \mathcal{S}_A / \sim_n$. But this directly derives from the definition of $\sim_n$. In fact, since for all F and $n \geq 1$ $s \sim_{n+1} s'$, we have that there exists a scheduler F' such that $Prob^F(s, \tau^*\alpha, C) = Prob^{F'}(s', \tau^*\alpha, C)$ $\forall C \in \mathcal{S}_A / \sim_n$. □

3.2 Classes

In this section we define the notion of *class* of a PTA. A class represents a set of pairs of configurations that belong to the relation we are considering. Hence, a set of classes $\mathcal{G}$ defines a relation $\approx_{\mathcal{G}}$.

Definition 3.2 *A* class *of the PTA A is a finite set $g \subseteq Q \times Q \times \Psi(X \cup \overline{X})$. A class g defines the relation $\approx_g \subseteq \mathcal{S}_{\mathcal{A}} \times \mathcal{S}_{\mathcal{A}}$ containing pairs $((q, v), (q', v'))$ such that there exists $(q, q', \psi) \in g$ with $v'' \models \psi$, where $v''(x) = v(x)$ and $v''(\overline{x}) = v'(x)$, for any $x \in X$.*

From now on, without loss of generality, we suppose that for any $q, q' \in Q$ there exists at most one $(q, q', \psi) \in g$, and, in such a case, $\psi \neq false$. Actually, the triple $(q, q', false)$ can be deleted, and, two triples (q, q', ψ) and (q, q', ψ') can be replaced with the triple $(q, q', \psi \vee \psi')$.

Given two classes g_1 and g_2, we define the class resulting from the intersection $g_1 \cap g_2$ and from the union $g_1 \cup g_2$. Namely, $g_1 \cap g_2 = \{(q, q', \psi \wedge \psi') \mid (q, q', \psi) \in g_1$ and $(q, q', \psi') \in g_2\}$ and $g_1 \cup g_2 = \{(q, q', \psi \vee \psi') \mid (q, q', \psi) \in g_1$ and $(q, q', \psi') \in g_2\}$. Moreover, with $\neg g$ we denote the class $\{(q, q', \neg\psi) \mid (q, q', \psi) \in g)\}$.

With $Set(A)$ we denote the set of set of classes $\mathcal{G}$ of A such that, for any $g_1, g_2 \in \mathcal{G}$ with $g_1 \neq g_2$, it holds that, for any $(q, q', \psi) \in g_1$ and $(q, q', \psi') \in g_2$ it holds that $\psi \wedge \psi' \equiv false$. A set of classes $\mathcal{G}$ defines the relation $\approx_{\mathcal{G}}$ that is the relation $\bigcup_{g \in \mathcal{G}} \approx_g$.

Now, classes are finitely many since the number of clock zones is finite (see Proposition 1.10). Moreover, the number of clocks is of the order of $|X|$. Hence, the number of triples (q, q', ψ) is in the order of the number of clock zones of A.

Example 3.2 *The set $\mathcal{G} = \{(q_0, q_2, 0 \leq x \leq \overline{x})\}$ is in $Set(A)$ where A is the PTA of Example 3.1. The set $\mathcal{G}$ defines the relation $\approx_{\mathcal{G}}$ such that $(q, v) \approx_{\mathcal{G}} (q', v')$ iff $q = q_0$, $q' = q_2$ and $v(x) \leq v'(x)$.*

3.3 The algorithm *Clean*

We introduce the algorithm $Clean^A(\mathcal{G})$ that eliminates the relations between configurations such that the unsatisfiability of bisimulation requirements is not caused by the fact that we cannot perform the time step α. Intuitively, we use the algorithm *Clean* in order to remove the relations between configurations from which it is not possible to perform the same time step to reach a certain class of bisimilar configurations.

We denote with ψ_g the formula

$$\bigwedge_{(q_1,q_2,\psi')\in g} \psi[X^{q_1} := X^{q_1} + t][\overline{X}^{q_2} := \overline{X}^{q_2} + t]$$

where t is a new variable representing the time elapsed, and, x^q and $\overline{x}^q$ are new clocks, for any state q and for any clock $x \in X$.

Definition 3.3 *Given a class g, with $prec_T(g)$ we denote the class*

$$\bigcup_{(q,q',\psi)\in g} \{(q, q', \exists X^Q \cup \overline{X}^Q \cup \{t\}.t > 0 \wedge x = x^q \wedge \overline{x} = \overline{x}^{q'} \wedge \psi_g)\}$$

where $X^Q = \{x^q \mid x \in X,\ q \in Q\}$ and $\overline{X}^Q = \{\overline{x}^q \mid x \in X,\ q \in Q\}$.

The class $prec_T(g)$ contains the configurations from which g can be reached with the same time step.

The following lemma states that deleting the relations between configurations such that the unsatisfiability of bisimulation requirements is not caused by the fact that one cannot perform the time step α, is equivalent to deleting the relations between configurations from which it is not possible to perform the same time step to reach a certain class of bisimilar configurations.

Lemma 3.2 *Given a commutative reflexive and transitive relation $\approx$, the two following statements are equivalent:*

- *for any $s_1 \approx s_2$, if $s_1 \xrightarrow{\alpha} s_1'$ and $s_2 \xrightarrow{\alpha} s_2'$, with $\alpha \in \mathbb{R}^{>0}$, then $s_1' \approx s_2'$;*
- *for any $s_1 \approx s_2$, there exists a scheduler F such that for any scheduler F', $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) \neq Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C})$ or viceversa, with $\alpha \in \mathbb{R}^{>0}$, then it holds that $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) > 0$ and $Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C}) > 0$, for $\mathcal{C} \in \mathcal{S}_A/\approx$.*

Proof. We prove the two implications:

1. We prove that if $s_1 \approx s_2$ and if $s_1 \xrightarrow{\alpha} s_1'$ and $s_2 \xrightarrow{\alpha} s_2'$, with $\alpha \in \mathbb{R}^{>0}$, then $s_1' \approx s_2'$ implies the property that if $s_1 \approx s_2$ and there exists a scheduler F such that for any scheduler F', $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) \neq Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C})$ or viceversa, then $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) > 0$ and $Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C}) > 0$, for $\mathcal{C} \in \mathcal{S}_A/\approx$.

   Let us suppose by contradiction that there exists a scheduler F such that for any scheduler F', $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) \neq Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C})$ or viceversa, and either $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) = 0$ or $Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C}) = 0$, for any $\mathcal{C} \in \mathcal{S}_A/\approx$.

   Let us suppose that $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) = 0$ (the other case is similar). Since $s_1' \approx s_2'$ there exists a $\mathcal{C} \in \mathcal{S}_A/\approx$ such that $s_1', s_2' \in \mathcal{C}$. But $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) = 0$ and $s_1 \xrightarrow{\alpha} s_1'$ imply $s_1' \notin \mathcal{C}$, which is a contradiction.

2. We prove that if $s_1 \approx s_2$ and there exists a scheduler F such that for any scheduler F', $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) \neq Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C})$ or viceversa, then $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) > 0$ and $Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C}) > 0$, for $\mathcal{C} \in \mathcal{S}_A/\approx$, implies the property that $s_1 \approx s_2$ and if $s_1 \xrightarrow{\alpha} s_1'$ and $s_2 \xrightarrow{\alpha} s_2'$, with $\alpha \in \mathbb{R}^{>0}$, then $s_1' \approx s_2'$

Let us suppose by contradiction that $s'_1 \not\approx s'_2$. The fact that $s'_1 \not\approx s'_2$ implies that either $s'_1 \notin \mathcal{C}$ or $s'_2 \notin \mathcal{C}$, for any $\mathcal{C} \in \mathcal{S}_A/\approx$.

Let us suppose that $s'_1 \notin \mathcal{C}$ and $s'_2 \in \mathcal{C}$ (the other case is similar). Let F be a scheduler such that $F(s_1) = \alpha$ and $F(s_2) = \alpha$; we have that $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) \neq Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C})$ implying that $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) > 0$ and $Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C}) > 0$. But $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) > 0$ and $s_1 \xrightarrow{\alpha} s'_1$ imply that $s'_1 \in \mathcal{C}$, which is a contradiction. □

We give now the algorithm $Clean^A(\mathcal{G})$ which refines $\mathcal{G}$ by using $prec_T$ until the fixpoint is reached. By Lemma 3.2, $Clean^A(\mathcal{G})$ deletes the relations between configurations from which it is not possible to perform the same time step to reach a certain class of bisimilar configurations.

$Clean^A(\mathcal{G} : Set(A)) : Set(A)$
  repeat
    $\mathcal{G}' := \mathcal{G}$
    $g' := \bigcap_{g\in\mathcal{G}} \neg g$
    $\mathcal{G} := \bigcup_{g\in\mathcal{G}} g \cap \neg \left(\bigcup_{(q,q',\psi)\in g'} prec_T(\{(q,q',\psi)\})\right)$
  until $(\mathcal{G} == \mathcal{G}')$
  return $\mathcal{G}$

The class g' represents the set of pairs (s, s') such that $s \not\approx_{\mathcal{G}} s'$. Moreover, we have that $\left(\bigcup_{(q,q',\psi)\in g'} prec_T(\{(q,q',\psi)\})\right)$ represents the set of configurations from which we can reach configurations non bisimilar through some time step. Therefore, $\neg\left(\bigcup_{(q,q',\psi)\in g'} prec_T(\{(q,q',\psi)\})\right)$ represents the set of configurations from which, through any time step, bisimilar configurations are reached. As a consequence, at each step, we refine the relation $\approx_{\mathcal{G}}$ by deleting the pairs (s, s') such that it is not possible to perform the same time step from s and s' to reach a certain class.

The following lemma states the correctness of the algorithm $Clean^A$.

Lemma 3.3 $\approx_{Clean^A(\mathcal{G})}$ *is the biggest relation enclosed in* $\approx_{\mathcal{G}}$ *such that if* $s_1 \approx_{Clean^A(\mathcal{G})} s_2$ *and there exists a scheduler* F *such that for any scheduler* F', $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) \neq Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C})$ *or viceversa, with* $\alpha \in \mathbb{R}^{>0}$, *then* $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) > 0$ *and* $Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C}) > 0$, *for* $\mathcal{C} \in \mathcal{S}_A/\approx_{\mathcal{G}}$. *Moreover,* $Clean^A(\mathcal{G})$ *is computable in exponential time w.r.t. the size of A and, if* $\approx_g$ *is commutative, reflexive and transitive, for any* $g \in \mathcal{G}$, *then* $\approx_{g'}$ *is commutative, reflexive and transitive, for any* $g' \in Clean^A(\mathcal{G})$.

Proof. Since at each step we refine a triple (q, q', ϕ) with a triple (q, q', ϕ') such that $\phi' \Rightarrow \phi$, it is obvious that if $s \approx_{Clean^A(\mathcal{G})} s'$, then $s_1 \approx_{\mathcal{G}} s_2$. Therefore the relation $\approx_{Clean^A(\mathcal{G})}$ is enclosed in the relation $\approx_{\mathcal{G}}$.

Hence, we must prove that if $s_1 \approx_{Clean^A(\mathcal{G})} s_2$ and there exists a scheduler F such that for any scheduler F', $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) \neq Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C})$ or viceversa, with $\alpha \in \mathbb{R}^{>0}$, then $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) > 0$ and $Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C}) > 0$.

If this holds, then, since $\approx_g$ is commutative, reflexive and transitive for any $g \in \mathcal{G}$, $\approx_{g'}$ is commutative, reflexive and transitive for any $g' \in Clean^A(\mathcal{G})$.

By Lemma 3.2, it is sufficient to prove that if $s_1 \xrightarrow{\alpha} s'_1$ and $s_2 \xrightarrow{\alpha} s'_2$, for some s'_1 and s'_2 and $\alpha \in \mathbb{R}^{>0}$, then $s'_1 \approx_{Clean^A(\mathcal{G})} s'_2$. Given $s_1 \approx_{Clean^A(\mathcal{G})} s_2$, let us suppose, by contradiction, that $s'_1 \not\approx_{Clean^A(\mathcal{G})} s'_2$, for some s'_1 and s'_2 such that $s_i \xrightarrow{\alpha} s'_i$, for $i = 1, 2$.

Since $s_1 \approx_{Clean^A(\mathcal{G})} s_2$, it means that $s_1 \approx_{g'} s_2$ where $g' = \bigcap_{g\in Clean^A(\mathcal{G})} \neg g$. Moreover $s_1 \approx_g s_2$ for some $g \in Clean^A(\mathcal{G})$. Therefore, by Proposition 1.11, since $\alpha \in \mathbb{R}^{>0}$, $\mathcal{G}' \neq Clean^A(\mathcal{G})$, where $\mathcal{G}' = \bigcup_{g\in Clean^A(\mathcal{G})} g \cap \neg \left(\bigcup_{(q,q',\psi)\in g'} prec_T(\{(q,q',\psi)\})\right)$. But this contradicts the fact that $Clean^A(\mathcal{G})$ is the fixpoint.

We must prove now that for each relation $\approx'$ satisfying the same requirements of $\approx_{Clean^A(\mathcal{G})}$ we have that $\approx'$ is contained in $\approx_{Clean^A(\mathcal{G})}$.

We suppose by contradiction that there exists such a relation $\approx'$ with $\approx' \supset \approx_{Clean^A(\mathcal{G})}$. Hence, there exist s_1 and s'_1 such that $s_1 \approx' s'_1$ and $s_1 \not\approx_{Clean^A(\mathcal{G})} s'_1$.

Let us suppose that $Clean^A(\mathcal{G})$ terminates after n steps and let $\mathcal{G}_1, \ldots, \mathcal{G}_n$ be the sets of classes in $Set(A)$ corresponding to the variable $\mathcal{G}'$ of the beginning of each step.

By induction and by Proposition 1.11, it is easy to show that $s_1 \approx_{\mathcal{G}_i} s'_1$ if:

- $s_1 \approx_{\mathcal{G}} s'_1$
- for any runs $s_1 \xrightarrow{\alpha_1} s_2 \ldots \xrightarrow{\alpha_{m-1}} s_m$ and $s'_1 \xrightarrow{\alpha_1} s'_2 \ldots \xrightarrow{\alpha_{m-1}} s'_m$, with $s_m \not\approx_{\mathcal{G}} s'_m$ and $s_h \approx_{\mathcal{G}} s'_h$ and $\alpha_h \in \mathbb{R}^{>0}$, for any $h = 1, \ldots, m-1$, it holds that $m > i$.

Now, $s_1 \approx' s'_1$ implies $s_1 \approx_{\mathcal{G}} s'_1$. Since $s_1 \not\approx_{Clean^A(\mathcal{G})} s'_1$, there exists $i > 1$ such that $s_1 \approx_{\mathcal{G}_j} s'_1$, for any $j \in [1, i-1]$, and $s_1 \not\approx_{\mathcal{G}_i} s'_1$. Hence there exist two runs $s_1 \xrightarrow{\alpha_1} s_2 \ldots \xrightarrow{\alpha_{i-1}} s_i$ and $s'_1 \xrightarrow{\alpha_1} s'_2 \ldots \xrightarrow{\alpha_{i-1}} s'_i$ such that $s_h \not\approx_{\mathcal{G}} s'_h$ for some $h \leq i$. Therefore, since $\approx' \subseteq \approx_{\mathcal{G}}$ it holds that $s_h \not\approx' s'_h$, and hence $s_l \not\approx' s'_l$, for any $l = 1, \ldots, h$, contradicting the fact that $s_1 \approx' s'_1$.

The fixpoint is computable after a finite number of steps since the number of formulae are finitely many and, at each step of the algorithm, the set $\mathcal{G}$ is updated with an enclosed set of classes. Since the number of classes is exponential w.r.t. the size of A, we have that $Clean^A(\mathcal{G})$ is computable in exponential time w.r.t. the size of A. $\square$

We note that after applying the algorithm $Clean$ to a set of classes $\mathcal{G}$ for $\alpha \in \mathbb{R}^{>0}$, it is sufficient to check whether $Prob^F(s_1, \tau^*\alpha, \mathcal{C}) \neq Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C})$ for $Prob^F(s_1, \tau^*\alpha, \mathcal{C}), Prob^{F'}(s_2, \tau^*\alpha, \mathcal{C}) > 0$.

Example 3.3 *Given the set of classes $\mathcal{G} = \{g_1, g_2\}$ of Example 3.1 where*

$$g_1 = \{(q_3, q_8, 0 \leq x \leq 3 \wedge 0 \leq \overline{x} \leq 3)\}$$

and

$$g_2 = \{(q_3, q_8, x > 3 \wedge \overline{x} > 3)\}.$$

Since from q_3 and q_8 only time steps can be performed, we have that, by following Proposition 1.9, variable t can be deleted, and hence $\neg\left(\bigcup_{(q,q',\psi)\in g'} prec_T((q, q', \psi))\right) = \{(q_3, q_8, (x = \overline{x}) \vee (x > 3 \wedge \overline{x} > 3))\}$. Therefore, refining $\mathcal{G}$, we have the class $\{(q_3, q_8, 0 \leq x \leq 3 \wedge x = \overline{x})\}$ and the class $\{(q_3, q_8, x > 3 \wedge \overline{x} > 3\}$ instead of g_1 and g_2, respectively. Actually, a time step does not give any problem for the class g_2, but for the class g_1 we must have that $x = \overline{x}$ to reach bisimilar configurations. As an example, $(q_3, x = 2.5) \approx_{\mathcal{G}} (q_8, x = 3)$, but $(q_3, x = 2.5) \xrightarrow{0.2} (q_3, x = 2.7)$, $(q_8, x = 3) \xrightarrow{0.2} (q_8, x = 3.2)$ and $(q_3, x = 2.7) \not\approx_{\mathcal{G}} (q_8, x = 3.2)$.

3.4 The Cut operator

We introduce a model of Unlabeled Probabilistic Automata that differ slightly from the ones in Definition 1.6. In this chapter, when we refer to PAs we mean Unlabeled PAs. Given a finite set Q, a *distribution* π on Q is a function $\pi : Q \to [0, 1]$ such that $\sum_{q\in Q} \pi(q) = 1$. Intuitively, $\pi(q)$ models the probability of a transitions with q as target state. With $Dist(Q)$ we denote the set of finite sets D of distributions on Q.

Definition 3.4 *An* Unlabeled Probabilistic Automaton *(PA) is a pair $A = (Q, \delta)$, where:*

- *Q is a finite set of states.*
- *$\delta \subseteq Q \times Dist(Q)$ is a finite set of transitions.*

Let $\sigma \in ExecFrag$ such that $\sigma = q_0 \xrightarrow{\pi_1} q_1 \xrightarrow{\pi_2} \ldots q_k$; we define $P(\sigma) = \pi_1(q_1) \cdot \ldots \cdot \pi_k(q_k)$.

As usual, we need a notion of scheduler to resolve the nondeterminism that arises when choosing the distribution within a set $D \in Dist(Q)$. A *scheduler* for a Probabilistic Automaton A is a function F assigning to each finite sequence σ in $ExecFrag$ a distribution $\pi \in D$ such that there exists a transition (q, D) with $last(\sigma) = q$.

For a scheduler F of a PA A we define $ExecFrag^F$ (resp. $Exec^F$) as the set of execution fragments (resp. the set of executions) $\sigma = q_0 \xrightarrow{\pi_1} q_1 \xrightarrow{\pi_2} q_2 \xrightarrow{\pi_3} \ldots$ such that $F(\sigma^{i-1}) = \pi_i$, for any $0 < i < |\sigma|$.

Given a scheduler F, $Exec^F(q)$ is the set of executions in $Exec^F$ starting from q, $ExecFrag^F(q)$ is the set of execution fragments in $ExecFrag^F$ starting from q, and $\Sigma\_Field^F(q)$ is the smallest sigma field on $Exec^F(q)$ that contains the basic cylinders $\sigma \uparrow$, where $\sigma \in ExecFrag^F(q)$. The probability measure $Prob^F$ is the unique measure on $\Sigma\_Field^F(q)$ such that $Prob^F(\sigma \uparrow) = P(\sigma)$.

Given a set of states Q', with $Exec^F(q, Q') = \{\sigma \in Exec^F(q) \mid last(\sigma^i) \in Q', \text{ for some } i\}$ we denote the set of executions starting from q and reaching a state in Q'.

In this section we define a cut operator that, given a class, splits it into a set of classes satisfying the requirements of the definition of weak bisimulation. For this purpose we construct two PAs by using predecessor operators for computing the probabilities $Prob^F(s, \tau^*\alpha, \mathcal{C})$.

For PTAs the probability strongly depends on the transitions enabled. The following proposition ensures that configurations s' reachable from a configuration s by discrete steps have either the same value for the clocks or the clocks are equal to zero.

Proposition 3.1 *For each $\sigma = (q_0, v_0) \xrightarrow{\alpha_1} s_1 \ldots \xrightarrow{\alpha_k} (q_k, v_k)$, with $\alpha_i \in \Sigma \cup \{\tau\}$, it holds that, either $v_i(x) = v_0(x)$ or $v_i(x) = 0$, for any i and $x \in X$.*

Proof. Since there is no time step, $v_i(x) \neq v_0(x)$ if and only if x is reset and $v_0(x) > 0$. □

Since g expresses a set of configurations, for any configurations (q, v) and (q, v') such that $(q, v) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} (q, v')$, with $\{\alpha_1, \ldots, \alpha_n\} \subseteq \Sigma \cup \{\tau\}$, $Adm((q, v))$ can be different from $Adm((q, v'))$. But, by Proposition 3.1, the set of transitions that can be taken from a configuration can be deterministically associated with a state, if it is known that $x > 0$ or $x = 0$, for each clock x.

Therefore, we define the set $\mathcal{F}$ as the set of functions $f : Q \times 2^X \to 2^\delta$ such that $f(q, B) \subseteq \delta(q)$, for any state q and set of clocks B. The set $f(q, B)$ represents the transitions that are enabled when a configuration is composed by the state q and a valuation with only clocks in B equal to 0. Sometimes we will write $f(q, B, a)$ with $a \in \Sigma \cup \{\tau\}$ to denote the set $f(q, B) \cap \delta(a)$.

Definition 3.5 *Given a class g and a set of transitions E, with $prec(g, E)$ and $\overline{prec}(g, E)$ we denote, respectively, the classes*

$$\bigcup_{(q,a,\phi,B,q') \in E} \{(q, q'', Ap_A(\phi \wedge \exists B.\psi \wedge B = 0) \mid (q', q'', \psi) \in g)\}$$

and

$$\bigcup_{(q,a,\phi,B,q') \in E} \{(q'', q, Ap_A(\phi[X := \overline{X}] \wedge \exists \overline{B}.\psi \wedge \overline{B} = 0) \mid (q'', q', \psi) \in g\}.$$

Moreover, $[g]$ is the set of triples (q, q', ψ) such that $\psi \not\equiv false$ and there exists $(q, q', \psi') \in g$ with, for any $x \in X \cup \overline{X}$, either $\psi \Rightarrow (\psi' \wedge x = 0)$ or $\psi \Rightarrow (\psi' \wedge x > 0)$.

The formulae $prec(g, e)$ and $\overline{prec}(g, e)$ give the set of configurations from which, by performing a transition in E, the first and the second component are reachable, respectively. The set $[g]$ is the set of triples enclosed in g such that each variable is either equal to 0 or greater than 0.

We can calculate the probability of performing $\tau^*\alpha$ by reducing this problem to the reachability problem in the model of Probabilistic Automata.

Given a class $\bar{g}$ such that $\approx_{\bar{g}}$ is commutative, reflexive and transitive, with $\bar{g}$ sometimes we will denote the set of configurations $\{s \mid s \approx_{\bar{g}} s' \text{ and } s, s' \in \mathcal{S}_A\}$. More precisely, with $Prob^F(s, \tau^*\alpha, \bar{g})$

and $ExecFrag^F(s, \tau^*\alpha, \bar{g})$ we denote $Prob^F(s, \tau^*\alpha, \mathcal{C})$ and $ExecFrag^F(s, \tau^*\alpha, \mathcal{C})$, respectively, where $\mathcal{C} = \{s \mid s \approx_{\bar{g}} s' \text{ and } s, s' \in \mathcal{S}_A\}$.

Hence, we have to check whether for any scheduler F there exists a scheduler F' such that $Prob^F(s_1, \tau^*\alpha, \bar{g}) = Prob^{F'}(s_2, \tau^*\alpha, \bar{g})$ and vice versa. Therefore, we calculate the probability p_i to reach $\bar{g}$ from s_i by performing $\tau^*\alpha$, for $i = 1, 2$. To this purpose we construct two PAs A_1 and A_2 such that the probability to reach the special state *good* from a certain state of A_i is equal to p_i, for $i = 1, 2$.

States of A_1 are either triples (q, q', ψ) or a state in $\{wrong, good\}$. The triple (q, q', ψ) represents the configurations (q, v) and (q', v'), with $(q, v) \approx_{\{(q,q',\psi)\}} (q', v')$, that can be crossed for reaching $\bar{g}$ starting from s_1 and s_2, respectively. The state *wrong* represents that a step α', with $\alpha' \notin \{\alpha, \tau\}$, is performed. If $\alpha \in \mathbb{R}^{>0}$, then the state *good* represents that class $\bar{g}$ is reached with the time step α. Otherwise, if $\alpha \in \Sigma \cup \{\tau\}$, then the state *good* represents that class $\bar{g}$ is reached with the discrete step α performed by both the configurations reached from s_1 and s_2.

Hence, the probability associated with two triples $z = (q_1, q_2, \psi)$ and $z' = (q'_1, q'_2, \psi')$ with $q'_2 = q_2$ is equal to the probability of reaching states expressed by z from those expressed by z' by a step performed from q_1.

The probability associated with two states $z = (q_1, q_2, \psi)$ and $z' = (q'_1, q'_2, \psi')$ with $q'_1 = q_1$ is equal to 1 if ψ' is reached from ψ by a τ step performed from q_2. Actually, the probability computed for the first configuration is not affected by the probabilities of the second one.

The PA A_2 is constructed similarly.

Definition 3.6 *Let $\alpha \in \Sigma \cup \{\tau, \lambda\}$, where λ denotes a generic time step. We define $A_1(g, \bar{g}, f, f', \alpha)$, the PA $(\Sigma, Q', q'_0, \delta', \Pi')$ such that*

- *$Q' = [Q \times Q \times \{true\}] \cup \{good, wrong\}$.*
- *δ' is the set of pairs $((q_1, q_2, \psi), D)$ such that $\pi' \in D$ is a distribution satisfying one of the following requirements:*

  1. *$\pi'(z) = \dfrac{\sum_{e \in E} \pi(e)}{\sum_{e \in f(q_1, B)} \pi(e)}$ where $B = \{x \mid \psi \Rightarrow x = 0\}$ and one of the following requirements hold:*
     - *$z \neq good, wrong$ and $E = \{e \in f(q_1, B, \tau) | (q_1, q_2, \psi) = prec(z, e)\}$, namely a τ step is performed from q_1.*
     - *$z = wrong$ and E is the set $\bigcup_{a \in \Sigma \setminus \alpha} f(q_1, B, a)$. Namely, a wrong state is reached if a symbol in $\Sigma \setminus \alpha$ is performed.*
     - *$z = good$ and $\alpha \in \Sigma \cup \{\tau\}$ and there exists $(q_1, q_2, \psi') \in prec(z, f(q_1, B, \alpha))$ such that $\exists \overline{X}.\psi' \Rightarrow \exists \overline{X}.\psi$, namely z has been reached correctly by a $\alpha \in \Sigma \cup \{\tau\}$ step.*
  2. *$\pi'(wrong) = 1$, in any state one can perform a time step with label different from α, thus reaching the wrong state.*
  3. *$\pi'(good) = 1$, if $\alpha = \lambda$ and $(q_1, q_2, \psi) \in prec_T(\{(q, q', \psi) \in z \mid (q, q', \psi') \in g\})$, namely z has been reached correctly by a time step.*
  4. *$\pi'(z) = 1$, $z \neq good, wrong$ and $(q_1, q_2, \psi) = \overline{prec}(z, e)$ for some $e \in f'(q_2, \{\overline{x} \mid \psi \Rightarrow \overline{x} = 0\}, \tau)$, namely, a τ step is performed from q_2.*

The PA $A_2(g, \bar{g}, f, f', \alpha)$ can be defined symmetrically.

The PAs $A_i(g, \bar{g}, f, f', \alpha)$ may have an exponential number of states, but they can be constructed by using a more efficient technique based on backward symbolic analysis thanks to the *prec* operator and the theory of regions [24, 68, 87].

The following lemma states that it is possible to compute the probability of reaching with $\tau^*\alpha$ the class $\bar{g}$ from the configurations s and s' (with $s \approx_z s'$) by computing the probabilities of reaching the state *good* of A^1 and A^2 from z, where $A_i = A_i(g, \bar{g}, f, f', \alpha)$ for $i = 1, 2$.

Lemma 3.4 *Let $\mathcal{G}$ be such that $Clean^A(\mathcal{G}) = \mathcal{G}$ and $\approx_g$ is commutative, reflexive and transitive, for any $g \in \mathcal{G}$. Let $g, \bar{g} \in \mathcal{G}$, and (q_1, v_1) and (q_2, v_2) be two configurations such that $(q_1, v_1) \approx_{\{z_0\}} (q_2, v_2)$ with $z_0 \in [g]$. For any $i = 1, 2$ and scheduler F of A there exists a scheduler F' of $A^i = A_i(g, \bar{g}, f, f', \alpha')$, for some f and f', such that $Prob_A^F((q_i, v_i), \tau^*\alpha, \bar{g}) = Prob_{A^i}^{F'}(z_0, good)$, and vice versa, where $\alpha' = \alpha$ if $\alpha \in \Sigma \cup \{\tau\}$, and $\alpha' = \lambda$ if $\alpha \in \mathbb{R}^{>0}$.*

Proof. By Proposition 1.11 and 3.1 we have that, for any $a \in \Sigma \cup \{\tau\} \cup \mathbb{R}^{>0}$ and $s \approx_{\{z\}} s'$ with $z \in [Q \times Q \times \{true\}]$, given the possible set of steps $s \xrightarrow{a} s_1$, $s \xrightarrow{a} s_2, \ldots, s \xrightarrow{a} s_n$ and $s' \xrightarrow{a} s'_1$, $s' \xrightarrow{a} s'_2, \ldots, s' \xrightarrow{a} s'_m$, such that, if $a = \alpha$, then either $\{(s_i, s'_j) \mid i \in [1, n], j \in [1, m]\} \subseteq \approx_{\bar{g}}$ or $\{(s_i, s'_j) \mid i \in [1, n], j \in [1, m]\} \cap \approx_{\bar{g}} = \emptyset$, it holds that

1. $a \notin \alpha \cup \{\tau\}$ iff there exist $\sigma^l = z \longrightarrow z'_l$ of A^l, for $l = 1, 2$, such that:
   (a) $z'_l = wrong$, for any $l = 1, 2$;
   (b) $P_{A_1}(\sigma^1) = \sum_{i \in [1,n]} P_A(s \xrightarrow{a} s_i)$ and $P_{A^2}(\sigma^2) = \sum_{j \in [1,m]} P_A(s' \xrightarrow{a} s'_j)$.
2. $a = \alpha$ iff there exist $\sigma^l = z \longrightarrow z'_l$ of A^l, for $l = 1, 2$, such that:
   (a) if $\{(s_i, s'_j) \mid i \in [1, n], j \in [1, m]\} \subseteq \approx_{\bar{g}}$, then $z_l = good$, for any $l = 1, 2$;
   (b) if $\{(s_i, s'_j) \mid i \in [1, n], j \in [1, m]\} \cap \approx_{\bar{g}} = \emptyset$, then $z_l = wrong$, for any $l = 1, 2$;
   (c) $P_{A^1}(\sigma^1) = \sum_{i \in [1,n]} P_A(s \xrightarrow{a} s_i)$ and $P_{A^2}(\sigma^2) = \sum_{j \in [1,m]} P_A(s' \xrightarrow{a} s'_j)$.
3. $a = \tau$ and $n, m > 0$ iff there exist $\sigma^l = z \longrightarrow z'_l \longrightarrow z''_l$ of A^l, for $l = 1, 2$, such that $P_{A^1}(\sigma^1) = \sum_{i \in [1,n]} P_A(s \xrightarrow{a} s_i)$ and $P_{A^2}(\sigma^2) = \sum_{j \in [1,m]} P_A(s' \xrightarrow{a} s'_j)$;
4. $a = \tau$ and $n > 0$ and $m = 0$ iff there exist $\sigma^l = z \longrightarrow z'_l$ of A^l, for $l = 1, 2$, such that $P_{A^1}(\sigma^1) = \sum_{i \in [1,n]} P_A(s \xrightarrow{a} s_i)$ and $P_{A^2}(\sigma^2) = 1$;
5. $a = \tau$ and $n = 0$ and $m > 0$ iff there exist $\sigma^l = z \longrightarrow z'_l$ of A^l, for $l = 1, 2$, such that $P_{A^1}(\sigma^1) = 1$ and $P_{A^2}(\sigma^2) = \sum_{j \in [1,m]} P_A(s' \xrightarrow{a} s'_j)$.

We consider now two cases: $\alpha \in \mathbb{R}^{>0}$ and $\alpha \in \Sigma \cup \{\tau\}$.

If $\alpha \in \mathbb{R}^{>0}$, then it holds that, for any scheduler F such that $ExecFrag_A^F((q_1, v_1), \tau^*\alpha, \bar{g})$ is empty, there exists a scheduler F' such that $ExecFrag_{A'}^{F'}((q_2, v_2), \tau^*\alpha, \bar{g})$ is empty. Actually, by items 1, 2, 3, 4 and 5 and since $\approx_g$ is commutative, reflexive and transitive, for any $g \in \mathcal{G}$, there exists a surjective function ζ from $ExecFrag_A$ to $ExecFrag_{A^i}$ such that $Prob_A^F(\{\sigma' \in ExecFrag \mid \zeta(\sigma') = \sigma\}) = Prob_{A^i}^{F'}(\sigma)$, for any $\sigma \in ExecFrag_{A^i}$ and $i = 1, 2$, if and only if it holds that $ExecFrag_A((q_1, v_1), \tau^*\alpha, \bar{g}) \neq \emptyset$ or $ExecFrag_A((q_2, v_2), \tau^*\alpha, \bar{g}) \neq \emptyset$. In fact, we note that, if either $ExecFrag_A((q_1, v_1), \tau^*\alpha, \bar{g}) = \emptyset$ or $ExecFrag_A((q_2, v_2), \tau^*\alpha, \bar{g}) = \emptyset$, then A^1 and A^2 cannot reach the state *good*, and hence $Prob_A^F((q_i, v_i), \tau^*\alpha, \bar{g})$ could be different from the probability of reaching the state *good* of A^i, for $i = 1, 2$.

But, since $Clean^A(\mathcal{G}) = \mathcal{G}$, by Lemma 3.3, it holds that, if there exists a scheduler F of A such that for any scheduler F' of A', $Prob_A^F((q_1, v_1), \tau^*\alpha, \bar{g}) \neq Prob_{A'}^{F'}((q_2, v_2), \tau^*\alpha, \bar{g})$, then it holds that $Prob_A^F((q_1, v_1), \tau^*\alpha, \bar{g}) > 0$ and $Prob_{A'}^{F'}((q_2, v_2), \tau^*\alpha, \bar{g}) > 0$. Hence, if $Prob_A^F((q_1, v_1), \tau^*\alpha, \bar{g}) \neq Prob_{A'}^{F'}((q_2, v_2), \tau^*\alpha, \bar{g})$, then $ExecFrag_A((q_1, v_1), \tau^*\alpha, \bar{g})$ and $ExecFrag_A((q_2, v_2), \tau^*\alpha, \bar{g})$ are not empty.

Otherwise, if $\alpha \in \Sigma \cup \{\tau\}$, it holds that for any $i = 1, 2$ and scheduler F of A there exists a scheduler F' of A^i such that $Prob_A^F((q_i, v_i), \tau^*\alpha, \bar{g}) = Prob_{A^i}^{F'}(z_0, good)$, and vice versa. Actually, by items 1, 2, 3, 4 and 5 and since $\approx_g$ is commutative, reflexive and transitive, for any $g \in \mathcal{G}$, there exists a surjective function ζ from $ExecFrag_A$ to $ExecFrag_{A^i}$ such that $Prob_A(\{\sigma' \in ExecFrag \mid \zeta(\sigma') = \sigma\}) = Prob_{A^i}(\sigma)$, for any $\sigma \in ExecFrag_{A^i}$ and $i = 1, 2$.

Since we have required that (q_1, q_2, ψ) is such that $\exists \overline{X}.\psi' \Rightarrow \exists \overline{X}.\psi$, it is not necessary that $ExecFrag_A((q_2, v_2), \tau^*\alpha, \bar{g})$ is not empty. In effect, $\exists \overline{X}.\psi$ hides valuations in the right side of the relation since, given two configurations s and s', the possibility of performing a transition step from s is not affected by s'. Similarly, if $ExecFrag_A((q_1, v_1), \tau^*\alpha, \bar{g})$ is empty. $\square$

Example 3.4 *Let us assume the set of classes $\{g_1, g_2\}$ of the PTA in Example 3.1 such that $g_1 = \{(q_2, q_7, 0 < x < 3 \wedge x = \overline{x}), (q_7, q_2, 0 < x < 3 \wedge x = \overline{x}), (q_2, q_2, 0 < x < 3 \wedge x = \overline{x}), (q_7, q_7, 0 < x < 3 \wedge x = \overline{x})\}$ and $g_2 = \{(q_3, q_8, true), (q_8, q_3, true), (q_3, q_3, x = \overline{x}), (q_8, q_8, x = \overline{x})\}$. We note that $\approx_{g_1}$ and $\approx_{g_2}$ are commutative, reflexive and transitive. Let f be the function such that $f(q, B) = \delta(q)$.*

For the state $(q_2, q_7, 0 < x < 3 \wedge x = \overline{x})$ of $A_1(g_1, g_2, f, f, a)$ we have the distributions π such that either $\pi(good) = \frac{1}{2}$ and $\pi(wrong) = \frac{1}{2}$ or $\pi(wrong) = 1$. For $A_2(g_1, g_2, f, f, a)$ we have the distributions π such that either $\pi((q_2, q_7, 0 < x < 3 \wedge x = \overline{x})) = \frac{1}{2}$ and $\pi(good) = \frac{1}{2}$ or $\pi(wrong) = 1$.

For the state $(q_7, q_2, 0 < x < 3 \wedge x = \overline{x})$ of $A_1(g_1, g_2, f, f, a)$ we have the distributions π such that either $\pi((q_7, q_2, 0 < x < 3 \wedge x = \overline{x})) = \frac{1}{2}$ and $\pi(good) = \frac{1}{2}$ or $\pi(wrong) = 1$. For $A_2(g_1, g_2, f, f, a)$ we have the distributions π such that either $\pi(good) = \frac{1}{2}$ and $\pi(wrong) = \frac{1}{2}$ or $\pi(wrong) = 1$.

For the state $(q_2, q_2, 0 < x < 3 \wedge x = \overline{x})$ of $A_1(g_1, g_2, f, f, a)$ we have the distributions π such that either $\pi(good) = \frac{1}{2}$ and $\pi(wrong) = \frac{1}{2}$ or $\pi(wrong) = 1$. For $A_2(g_1, g_2, f, f, a)$ we have the same distributions.

For the state $(q_7, q_7, 0 < x < 3 \wedge x = \overline{x})$ of $A_1(g_1, g_2, f, f, a)$ we have the distributions π such that either $\pi((q_7, q_7, 0 < x < 3 \wedge x = \overline{x})) = \frac{1}{2}$ and $\pi(good) = 1$ or $\pi(wrong) = 1$. For $A_2(g_1, g_2, f, f, a)$ we have the same distributions.

With $Cut^A(g, \bar{g})$ we denote the set $\{g_1, \ldots, g_n\} \in Set(A)$ such that, $[g] \supset \bigcup_{i \in [1,n]} [g_i]$, and for any $f, f' \in \mathcal{F}$ and $\alpha \in \Sigma \cup \{\tau, \lambda\}$ it holds that $z_1, z_2 \in [g_i]$, for some $i = 1, \ldots, n$, iff for any scheduler F there exists a scheduler F' such that $Prob^F_{A^1}(z_1, good) = Prob^{F'}_{A^2}(z_2, good)$, where $A^j = A_j(g, \bar{g}, f, f', \alpha)$ with $j = 1, 2$.

Example 3.5 *Let us assume the classes g_1 and g_2 of Example 3.4. We have that $Cut^A(g_1, g_2) = \{\{(q_2, q_2, 0 < x < 3 \wedge x = \overline{x})\}, \{(q_7, q_7, 0 < x < 3 \wedge x = \overline{x})\}\}$.*

Note that to compute the probabilities $Prob^F_{A^i}(z, good)$ it is sufficient to compute only once the automata $A^i = A_i(g, \bar{g}, f, f', \alpha)$, for $i = 1, 2$. Therefore, since the number of clock zones is exponential w.r.t. the size of A, we have the following corollary.

Corollary 3.1 *For any $g, \bar{g}$, $Cut^A(g, \bar{g})$ is computable in double exponential time w.r.t. the size of A. If A has no τ transitions, $Cut^A(g, \bar{g})$ is computable in exponential time w.r.t. the size of A.*

3.5 Computing the Equivalence Classes

We can now define the algorithm $Classes(A)$ that returns a set in $Set(A)$ giving the configurations that are bisimilar. The algorithm refines the classes by using the triples returned by the Cut operator and the algorithm $Clean$ until the fixpoint is reached when starting from the class $\bigcup_{q,q' \in Q} \{(q, q', X \geq 0 \wedge \overline{X} \geq 0)\}$ (namely the class containing all configurations). The refinement is done by splitting according to $Cut^A(g, g')$.

$Classes(A : PTA) : Set(A)$
  $\mathcal{G} := \bigcup_{q,q' \in Q} \{(q, q', X \geq 0 \wedge \overline{X} \geq 0)\}$
  repeat
    $\mathcal{G}' := \mathcal{G}$
    $Clean^A(\mathcal{G})$
    for each $g, \bar{g} \in \mathcal{G}$
      $\mathcal{G}'' := Cut^A(g, g')$
      if $\mathcal{G}'' \neq \{g\}$ then $\mathcal{G} := (\mathcal{G} \setminus \{g\}) \cup \mathcal{G}''$
  until $(\mathcal{G} == \mathcal{G}')$
  return $\mathcal{G}$

We have the following theorem stating the correctness of the algorithm. This implies the decidability of weak bisimulation for PTAs.

Theorem 3.1 *For any configurations $s, s' \in \mathcal{S}_A$, $s \approx s'$ if and only $s \approx_{Classes(A)} s'$. Moreover, $Classes(A)$ is computable in double exponential time w.r.t. the size of A. If A has no τ transitions, then $Classes(A)$ is computable in exponential time w.r.t. the size of A.*

Proof. By Lemma 3.1, it is sufficient to prove that $s \approx_{Class(A)} s' \Leftrightarrow \forall n \geq 0 \quad s \sim_n s'$. Let $\mathcal{G}_1, \ldots, \mathcal{G}_n, \ldots$ be the value of variable $\mathcal{G}$ at each step of the algorithm. We prove by induction that $s \approx_{\mathcal{G}_i} s' \Leftrightarrow s \sim_n s'$, for all $n \in [0, i]$.

The base case is obvious since $\mathcal{G}_0 := \bigcup_{q,q' \in Q}\{(q, q', X \geq 0 \wedge \overline{X} \geq 0)\}$.

By induction we have that $s \approx_{\mathcal{G}_i} s' \Leftrightarrow s \sim_n s'$, for all $n \in [0, i]$. Therefore $\approx_g$ is commutative, reflexive and transitive, for any $g \in \mathcal{G}_i$. Hence, by Lemma 3.3 and Lemma 3.4, we have that $s \approx_{\mathcal{G}_{i+1}} s' \Leftrightarrow s \sim_n s'$, for all $n \in [0, i+1]$.

The fixpoint is computable after a finite number of steps since the number of formulae are finitely many and, at each step of the algorithm, the set $\mathcal{G}$ is updated with a set of classes included in the previous set of classes. By Corollary 3.1 $Class^A(\mathcal{G})$ is computable in double exponential time w.r.t. the size of A, and, if A has no τ transitions, then $Class(A)$ is computable in exponential time w.r.t. the size of A. $\square$

Chapter 4

Security Properties

In a multilevel system every agent is confined in a bounded security level; information can flow from a certain agent to another agent only if the level of the former is lower than the level of the latter. Access rules can be imposed by the system in order to control direct unwanted transmission from higher levels to lower levels; however, it could be possible to transmit information indirectly by using system side effects. Usually, this kind of indirect transmissions, called *covert channels*, do not violate the access rules imposed by the system.

The existence of covert channels has led to the more general approach of *information flow security*, which aims at controlling the way information may flow among different entities. The idea is to try to directly control the whole flow of information, rather than only the direct communication among agents. In [54] the authors introduce the notion of *Non Interference*, stating, intuitively, that low level agents should not be able to deduce anything about the activity of high level agents. By imposing some information flow rules, it is possible to control direct and indirect leakages, as both of them give rise to unwanted information flows.

In the literature there are many different definitions of security based on the information flow idea, and each is formulated in some system model (see, e.g., [54, 104, 57, 51, 52, 50, 17, 6]). Most of the properties considered are based on analysis of information flow that does not take into consideration aspects of time or probability, and therefore they are not useful to check the existence of probabilistic or timing covert channels. To overcome this, a significant work has been done in order to extend the study by considering either time (see, e.g., [52, 50, 17]) or probability (see, e.g., [57, 6, 123]).

Consider, for example, the two level system scheme in which one does not want interference between confidential data of the high level and the low level behaviors observed by an attacker who could infer confidential information (exploiting, for example, a covert channel).

Aldini et al. [6] observe that an attack could be done in the following way. Assume that a secret 1-bit value can be communicated to the unauthorized user among randomly created low-level noise, and that both secret value and random low-level noise belong to the same domain. The high behavior does not alter the set of possible low outcomes which are always the same with or without the high-level interaction. However, for a fixed value of the high input, an attacker observing the frequency of the low results deriving from repeated executions of the system could infer (with a certain probability) which one is directly communicated by the high user.

A system which in a nondeterministic setting is considered to be secure, in a richer framework, where the probability distribution of the possible events is known, may reveal information leakages. A probabilistic setting may allow estimating the level of security of a system by checking if the probability of detecting an illegal information flow is beyond a certain threshold.

This has required the use of descriptive means for systems which allow expressing time and probability. In this chapter we are interested in a general framework where both probabilistic and timing covert channels can be studied. Hence, for the description of systems we use the class of Probabilistic Timed Automata (PTAs) introduced in Chapter 2. Such a formalism reveals to be well-suited for the analysis of information flow security properties.

The framework of PTAs allows description of timed systems showing a probabilistic behavior in an intuitive and succinct way. Therefore, within the framework of PTAs, where time and probabilities are taken into consideration, the modeler can describe different aspects of a system, and analyze, on a single model, real-time properties, performance and reliability properties (by using classical model checking techniques), and information flow security properties useful to detect both probabilistic and timing covert channels.

We show here that these concepts, together with the analogous concepts for Probabilistic Automata and Timed Automata, can be expressed in a unique framework where both probability and time are considered.

In Section 4.1 we briefly set the context for dealing with multilevel security within the model of PTAs. In Section 4.2 we define *Non Interference* and *Non Deducibility on Composition* security properties for the models of NSs, PAs, TAs and PTAs. We give a classification of those properties according to their discriminating powers. In Section 4.3, as an application, we analyze information flow that may arise in a shared buffer.

4.1 The Context

One classical definition of secure information flow presupposes that data and principals have different levels of clearance, and requires that high information never flows downwards.

Given a system model with the basic operators of restriction, hiding and parallel composition together with a notion of observational equivalence, it is easy to set up a framework for the analysis of information flow.

A finite alphabet Σ of visible actions is assumed. A multilevel system interacts with agents confined in different levels of clearance. In order to analyze the information flow between parties with different levels of confidentiality, the set of visible actions is partitioned into high level actions and low level actions. Formally, we assume the set of possible actions $\Sigma = \Sigma_H \cup \Sigma_L$, with $\Sigma_H \cap \Sigma_L = \emptyset$. In the following, with $l, l' \dots$ and $h, h', \dots$ we denote low level actions of Σ_L and high level actions of Σ_H respectively. With Γ_H and Γ_L we denote the set of high level agents and low level agents. Formally, an automaton A with a set of action labels Σ' is in Γ_H (Γ_L) if $\Sigma' \subseteq \Sigma_H$ ($\Sigma' \subseteq \Sigma_L$). For simplicity, we specify only two-level systems; note, however, that this is not a real limitation, since it is always possible to deal with the case of more levels by iteratively grouping them in two clusters.

A low level agent is able to observe the execution of all the steps labeled with actions in Σ_L. The basic idea of Non Interference is that the high level does not interfere with the low level if the effects of high level communications are not visible by a low level agent. Finally, an important assumption when dealing with Non Interference analysis is that a system is considered to be *secure* (no information flow can occur) if there is no interaction with high level agents (if high level actions are prevented).

Other properties have been introduced in the literature in order to capture different behaviour of systems that have to be considered not secure. In [51] Focardi and Gorrieri promote the classification of a set of properties capturing the idea of information flow and Non Interference. One of the most interesting and intuitive security properties is the *Non Deducibility on Composition* (*NDC*), which states that a system A in isolation has not to be altered when considering all the potential interactions of A with the high level agents of the external environment.

4.2 Information Flow Security Properties

We define the Non Interference and the Non Deducibility on Composition security properties in a nondeterministic, in a probabilistic and in a timed setting. The concept of Non Interference was proposed originally in a purely nondeterministic setting [54, 104, 51]. Non Deducibility on

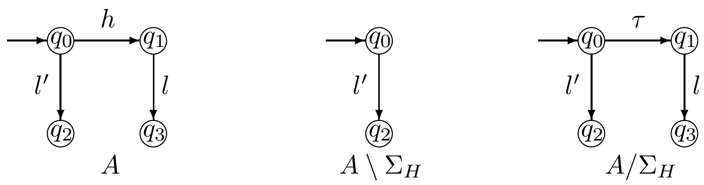

Figure 4.1: A nondeterministic covert channel.

Composition was proposed by Focardi and Gorrieri in order to detect insecure behaviour that the Non Interference property is not able to detect.

4.2.1 Non Interference

We define Non Interference properties, *Probabilistic Timed Non Interference* (*PTNI*), *Probabilistic Non Interference* (*PNI*), *Timed Non Interference* (*TNI*) and *Nondeterministic Non Interference* (*NNI*).

Definition 4.1 *Given a system A in PTAs (PAs, TAs, NSs, resp.) A is* PTNI *(*PNI, TNI, NNI*, resp.)-secure if and only if $A/\Sigma_H \approx A \setminus \Sigma_H$. We write $A \in$* PTNI *($A \in$* PNI, *$A \in$* TNI, *$A \in$* NNI, *resp.) when the system A is* PTNI *(*PNI, TNI, NNI, *resp.)-secure.*

In the definition above, $A \setminus \Sigma_H$ represents the isolated system, where all high level actions are prevented. As we have seen, such a system is considered secure due to the notion of Non Interference. If the observational behavior of the isolated system is equal to the behavior of A/Σ_H, which represents the system that communicates with high level agents in an invisible manner for the low agents point of view, A satisfies the security property.

Note that the *PNI* property is the *BSPNI* property defined in [6], the *TNI* property is an analogous of the *tBSNNI* property defined in [52], and *NNI* is the *BSNNI* property of [51].

The *PTNI* property, defined in an environment where both probability and time are studied, is able to detect information flow that may occur either due to the probabilistic behavior of the system, to the time when an action occurs or to a combination of them.

Proposition 4.1 *It is decidable whether a PTA (PA, TA, NS, resp.) A satisfies the* PTNI *(*PNI, TNI, NNI, *resp.) property.*

Proof. The result derives directly by the decidability of weak bisimulation for all the models, and by the computable definitions of the operators of hiding and restriction. □

Example 4.1 *In Figure 4.1 we show a case of nondeterministic information flow. Here, the high level action h interferes with the observation of the action l. Formally, in $A \setminus \Sigma_H$, a low level agent observes only the execution of l', whereas, in A/Σ_H a low level user may either observe the action l or the action l'. Hence, a low level agent, observing the event l knows that action h has occurred. This gives rise to an unsecure information flow. Formally we have $A/\Sigma_H \not\approx A \setminus \Sigma_H$, so that the* NNI *property reveals the covert channel.*

The security properties defined in the probabilistic and/or timed settings are conservative extensions of the security properties defined in the possibilistic and/or untimed settings.

Proposition 4.2 *The following implications hold:*

- $A \in \mathrm{PNI} \Rightarrow \mathit{unprob}(A) \in \mathrm{NNI}$.
- $A \in \mathrm{TNI} \Rightarrow \mathit{untime}(A) \in \mathrm{NNI}$.
- $A \in \mathrm{PTNI} \Rightarrow \mathit{unprob}(A) \in \mathrm{TNI} \wedge \mathit{untime}(A) \in \mathrm{PNI}$.

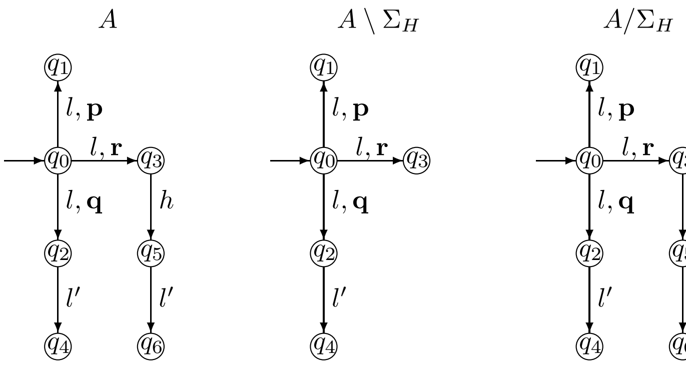

Figure 4.2: A probabilistic covert channel.

Proof. The implications follow by the bisimulation based definitions of the security properties and by the conservativeness of the notions of weak bisimulation (Lemmata 1.1, 1.3 and 2.2). Consider a PA A. If $A \in PNI$, by Lemma 1.1 we have that $unprob(A/\Sigma_H) \approx unprob(A \setminus \Sigma_H)$. Now, by definitions of *unprob*, hiding and restriction, it is easy to see that $unprob(A/\Sigma_H) = unprob(A)/\Sigma_H$ and that $unprob(A \setminus \Sigma_H) = unprob(A) \setminus \Sigma_H$. Therefore, we have that $unprob(A)/\Sigma_H \approx unprob(A) \setminus \Sigma_H$, proving that $unprob(A) \in NNI$. The proof is similar for the other cases. $\square$

The converse implications do not hold. The integration of probability and time adds new information that extends what is already known in the nondeterministic case. Therefore, systems considered to be secure in a purely possibilistic setting, may turn out to be insecure when considering aspects either of probability or of time. This is shown in Examples 4.2 and 4.3.

Example 4.2 *In Figure 4.2 we show a case of probabilistic information flow. System A may be seen either as PA, or as a PTA with $Inv(q_i) = true$ for every $i \in [0, 6]$. Abstracting away from probability, the system A could be considered secure. In a purely possibilistic setting, in both systems $unprob(A)/\Sigma_H$ and $unprob(A) \setminus \Sigma_H$ a low level agent can observe the action l or the sequence ll' without further information about the execution of action h. It holds that $unprob(A)/\Sigma_H \approx unprob(A) \setminus \Sigma_H$ and, therefore, $unprob(A) \in$* NNI *(if A is PA) or $unprob(A) \in$* TNI *(if A is a PTA). In a probabilistic framework, given $\mathbf{p} + \mathbf{r} + \mathbf{q} = 1$, the high level action h interferes with the probability of observing either a single action l or the sequence ll'. Formally, in $A \setminus \Sigma_H$, a low level agent observes either the single action l with probability $\mathbf{p} + \mathbf{r}$ or the sequence ll' with probability $\mathbf{q}$. However, in A/Σ_H the single event l is observed with probability $\mathbf{p}$ and the sequence ll' with probability $\mathbf{r} + \mathbf{q}$. As a consequence we have $A/\Sigma_H \not\approx A \setminus \Sigma_H$, so that the* PNI *and the* PTNI *properties reveal the probabilistic covert channel.*

Example 4.3 *In Figure 4.3 we show a case of timing information flow for the PTA A. We assume $Inv(q_i) = true$ for $i \in [2, 3]$ and $Inv(q_i) = x \leq 5$ for $i \in [0, 1]$. Abstracting away from time, the system A could be considered secure. In an untimed setting, in both systems $untime(A)/\Sigma_H$ and $untime(A) \setminus \Sigma_H$ a low level agent can observe only the action l executed with probability 1 without further information about the execution of action h. It holds that $untime(A)/\Sigma_H \approx untime(A) \setminus \Sigma_H$, and, therefore, $untime(A) \in$* PNI. *In a timed framework, given a clock $x \in \mathbb{R}^{\geq 0}$, the high level action h interferes with the time of observing the action l. Formally, in $A \setminus \Sigma_H$, a low level agent observes the single action l executed immediately. However, in A/Σ_H the single event l could either be observed immediately or when the clock x reaches value 5. A low level agent, observing the event l when clock x has value 5 knows that action h has occurred. As a consequence, we have $A/\Sigma_H \not\approx A \setminus \Sigma_H$, so that the* PTNI *property reveals the timing covert channel. The same holds for $unprob(A)$; in this case the covert channel is detected by the* TNI *property.*

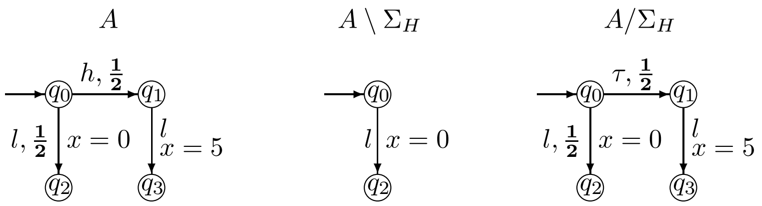

Figure 4.3: A timing covert channel.

For system A in Figure 4.2, *untime*(A) is not *PNI*, but *unprob*$(A) \in$ *TNI*. On the contrary, for system A in Figure 4.3, *unprob*(A) is not *TNI*, but *untime*$(A) \in$ *PNI*. This shows that the discriminating powers of time and probability, as regards the Non Interference property, are incomparable as stated in the next proposition.

Proposition 4.3 *The following implications hold:*

- $\exists\, PTA\ A :\ unprob(A) \in \mathrm{TNI} \wedge untime(A) \notin \mathrm{PNI}$*;*
- $\exists\, PTA\ A\ : untime(A) \in \mathrm{PNI} \wedge unprob(A) \notin \mathrm{TNI}$.

If we can express both time and probability as in PTAs, we are able to describe systems exhibiting information flow that neither a formalism with only probability nor a formalism with only time can express. For such systems we are able to show that they are not *PTNI*, even if they are both PTI and *TNI*, and therefore we are able to reveal a new covert channel.

Proposition 4.4 $\exists A : A \notin \mathrm{PTNI} \wedge unprob(A) \in \mathrm{TNI} \wedge untime(A) \in \mathrm{PNI}$.

Proof. Consider the PTA A in Figure 4.4. We assume $Inv(q_i) = true$ for $i \in \{3,4,8,9\}$, $Inv(q_i) = x \leq 3$ for $i \in \{0,1,5,7\}$ and $Inv(q_i) = x \leq 4$ for $i \in \{2,6\}$. It is easy to see that $untime(A) \in PTI$. In both $untime(A)/\Sigma_H$ and $untime(A) \setminus \Sigma_H$, a low level agent observes the single event l taken with probability 1, and therefore $untime(A)/\Sigma_H \approx untime(A) \setminus \Sigma_H$. It is also easy to see that *unprob*$(A) \in$ *TNI*. In both $unprob(A)/\Sigma_H$ and $unprob(A) \setminus \Sigma_H$, a low level agent could either observe the single event l taken when $x = 3$ or the event l taken when $x = 4$, and therefore $unprob(A)/\Sigma_H \approx unprob(A) \setminus \Sigma_H$. Finally, we show that A is not *PTNI*. In a probabilistic and timed framework, the high level action h interferes with the probability of observing the action l executed either when $x = 3$ or when $x = 4$. Formally, in $A \setminus \Sigma_H$, a low level agent observes the action l either when $x = 3$ or when $x = 4$ with probability $\frac{1}{2}$, respectively. However, in A/Σ_H the event l taken when $x = 3$ is observed with probability $\frac{19}{30}$, while the action l taken when $x = 4$ is observed with probability $\frac{11}{30}$. As a consequence, we have $A/\Sigma_H \not\approx A \setminus \Sigma_H$, so that the *PTNI* properties reveals the probabilistic timing covert channel. $\square$

The Venn diagram in Figure 4.5 summarizes the classification of timed and probabilistic Non Interference security properties.

4.2.2 Non Deducibility on Composition

We define the Non Deducibility on Composition properties *Probabilistic Timed Non Deducibility on Composition* (*PTNDC*), *Probabilistic Non Deducibility on Composition* (*PNDC*), *Timed Non Deducibility on Composition* (*TNDC*) and *Nondeterministic Non Deducibility on Composition* (*NNDC*).

Definition 4.2 *Given a system A in PTAs (PAs, TAs, NSs, resp.), A is* PTNDC *(*PNDC, TNDC, NNDC*, resp.)-secure if and only if* $\forall E \in \Gamma_H, \forall p \in]0,1[,\ \forall L \subseteq \Sigma_H \quad A/\Sigma_H \approx (A||^p_L E)\setminus\Sigma_H$. *We write* $A \in \mathrm{PTNDC}$ *(*$A \in \mathrm{PNDC}$, $A \in \mathrm{TNDC}$, $A \in \mathrm{NNDC}$*, resp.) when the system A is* PTNDC *(*PNDC, TNDC, NNDC*, resp.)-secure.*

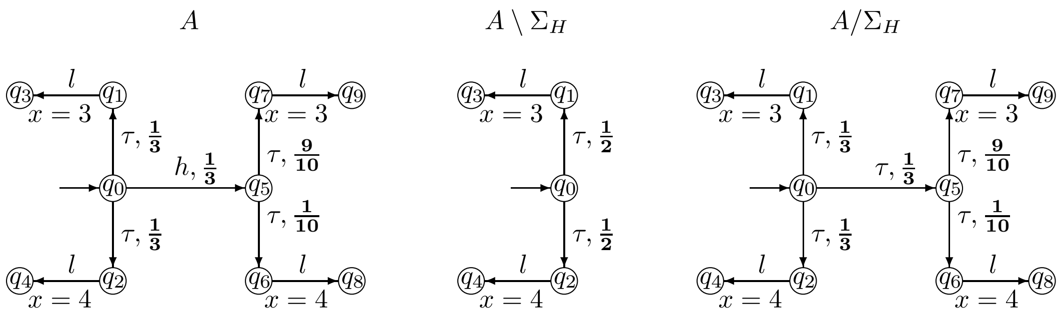

Figure 4.4: A probabilistic timing covert channel.

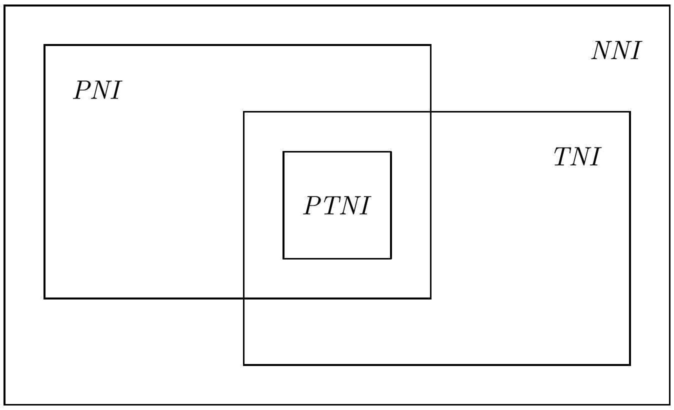

Figure 4.5: Relations among Non Interference security properties.

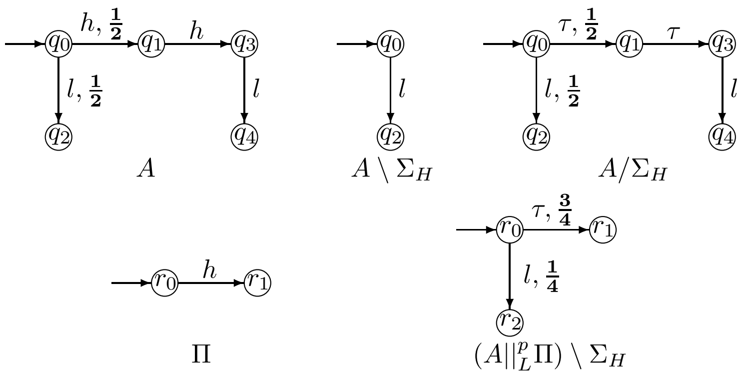

Figure 4.6: $A \in$ *PTNI*, but $A \notin$ *PTNDC*.

As we have seen, A/Σ_H represents the observable behavior of A from a low level agent point of view (i.e. the isolated system where all high level actions are hidden). System $(A||^p_L E) \setminus \Sigma_H$ represents, instead, system A communicating with the high agent E and then prevented by the execution of other high level actions. If the observational behavior of the isolated system is equal to the behavior of the system communicating with any high level agent, A satisfies the security property.

Proposition 4.5 *$A \in$ PTNDC (PNDC, TNDC, NNDC, resp.) $\Rightarrow A \in$ PTNI (PNI, TNI, NNI, resp.).*

Proof. If A is a NS, consider $E = (\emptyset, \{q\}, q, \emptyset) \in \Gamma_H$, if A is a PA consider $E = (\emptyset, \{q\}, q, \emptyset, \Pi) \in \Gamma_H$, if A is a TA consider $E = (\emptyset, \emptyset, \{q\}, q, \emptyset, Inv(q) = false) \in \Gamma_H$ while if A is a PTA consider $E = (\emptyset, \emptyset, \{q\}, q, \emptyset, Inv(q) = false, \Pi) \in \Gamma_H$. Intuitively, E is an automaton representing a high level agent which does not perform any transition. Consider then the set $L = \emptyset$. If system A is *PTNDC* (*PNDC*, *TNDC*, *NNDC*, resp.), then $\forall p \in]0, 1[$, $A/\Sigma_H \approx (A||^p_L E) \setminus \Sigma_H$. Now, by the definition of parallel composition, $(A||^p_L E) = A'$ where A' is an automaton behaving like A with only different state names (i.e., the states of A paired with the only state q of E). Now it is easy to see that $A' = A$ after a renaming of the states (i.e. by renaming each state (q', q) of A' with q'). As a consequence we have that $(A||^p_L E) = A$ and, therefore, $A/\Sigma_H \approx A \setminus \Sigma_H$, stating that $A \in$ *PTNI* (*PNI*, *TNI*, *NNI*, resp.). The converse implication does not hold, as it is shown in Example 4.4. □

Example 4.4 *Consider the PTA A of Figure 4.6. Again we assume $Inv(q_i) = true$ for every $i \in [0, 4]$. It is easy to see that A is* PTNI *secure, since $A/\Sigma_H \approx A \setminus \Sigma_H$. In both A/Σ_H and $A \setminus \Sigma_H$, a low level agent observes the single event l taken with probability* 1*. If we consider, instead, the high level agent Π of Figure 4.6, the set $L = \{h\}$ and assume probability $p = \frac{1}{2}$, we can observe that $A/\Sigma_H \not\approx (A||^p_L \Pi) \setminus \Sigma_H$. In fact, system A/Σ_H always performs action l with probability* 1*, while system $(A||^p_L \Pi) \setminus \Sigma_H$ reaches a deadlock state r_1 and does not perform any visible action with probability $\frac{3}{4}$ (as it turns out after the parallel composition of A and Π). As a consequence, automaton A is not* PTNDC *secure. We also have that $untime(A) \in$* PNI *but $untime(A) \notin$* PNDC*, $unprob(A)$ is* TNI *but not* TNDC*, and, finally, $unprob(untime(A)) \in$* NNI*, but $unprob(untime(A)) \notin$* NNDC*.*

Proposition 4.5 and Example 4.4 show that the *PTNI* property is not able to detect some potential deadlock due to high level activities, as put in evidence in [51] for the nondeterministic setting. For this reason we resort to the *PTNDC* property, which implies *PTNI*, in order to capture these finer undesirable behaviour.

Note that the PTAs in Figure 4.2 and Figure 4.3 are also *PTNDC*, this can be proven by considering two simple classes of high level agents. For the first class we consider just an high level agent that may synchronize with the two PTAs by performing an action h, whereas for the second class we consider an inactive high level agent that does not perform action h (see proof of Proposition 4.5).

As we did for the Non Interference security properties in the previous section, now we distinguish the discriminating power of time and probability with the Non Deducibility on Composition properties.

Proposition 4.6 *The following implications hold:*

- $A \in \text{PNDC} \Rightarrow unprob(A) \in \text{NNDC}$;
- $A \in \text{TNDC} \Rightarrow untime(A) \in \text{NNDC}$;
- $A \in \text{PTNDC} \Rightarrow unprob(A) \in \text{TNDC} \wedge untime(A) \in \text{PNDC}$.

Proof. As for Proposition 4.2, the implications follow by the bisimulation based definitions of the security properties and by the conservativeness of the notions of weak bisimulation. We prove the first implication and consider a PA A. If $A \in PNDC$, by Lemma 1.1 we have that for all $E \in \Gamma_H$, $p \in]0,1[$, $L \subseteq \Sigma_H$, $unprob(A/\Sigma_H) \approx unprob((A||^p_L E) \setminus \Sigma_H)$. Now, by definitions of *unprob*, hiding and restriction, it is easy to see that $unprob(A/\Sigma_H) = unprob(A)/\Sigma_H$ and that $unprob((A||^p_L E) \setminus \Sigma_H) = unprob(A||^p_L E) \setminus \Sigma_H$. Moreover, by definition of parallel composition for all $p \in]0,1[$ we have that $unprob(A||^p_L E) = unprob(A)||_L unprob(E)$. Therefore, we have that $unprob(A)/\Sigma_H \approx (unprob(A)||_L unprob(E)) \setminus \Sigma_H$. Now, since this condition holds for each PA $E \in \Gamma_H$ and since any NS E' may be derived by $unprob(E)$ for some PA E, we also have that for all NSs $E' \in \Gamma_H$, $unprob(A)/\Sigma_H \approx (unprob(A)||_L E') \setminus \Sigma_H$, thus proving that $unprob(A) \in$ *NNDC*. The proof is similar for the other cases. □

For the PTA A in Figure 4.2, $untime(A)$ is not *PNDC*, but $unprob(A) \in$ *TNDC*. On the contrary, for the PTA A in Figure 4.3, $unprob(A)$ is not *TNDC*, but $untime(A) \in$ *PNDC*. This shows that also for the Non Deducibility on Composition, time and probability add discriminating powers, as stated in the next proposition.

Proposition 4.7 *The following implications hold:*

- $\exists$ *PTA* A : $unprob(A) \in \text{TNDC} \wedge untime(A) \notin \text{PNDC}$;
- $\exists$ *PTA* A : $untime(A) \in \text{PNDC} \wedge unprob(A) \notin \text{TNDC}$.

If we can express both time and probability as in PTAs, we are able to describe systems exhibiting information flow that neither a formalism with only probability nor a formalism with only time can express. For such systems we are able to show that they are not *PTNDC*, even if they are both *PNDC* and *TNDC*, and therefore we are able to reveal a new covert channel.

Proposition 4.8 $\exists A : A \notin \text{PTNDC} \wedge unprob(A) \in \text{TNDC} \wedge untime(A) \in \text{PNDC}$.

Proof. If we consider again the PTA A in Figure 4.4 we may show that $untime(A) \in PNDC$ and $unprob(A) \in TNDC$ by considering only two classes of high level systems (one containing high systems which may synchronize with A and one with systems which may not). But, as we have seen, $A \notin PTNI$, therefore, for Proposition 4.5 we also have that $A \notin PTNDC$. □

The Venn diagram in Figure 4.7 summarizes our results.

It is worth noticing that, as it happens for the analogous properties defined in [51, 52, 6], the definition of the *PTNDC* property is difficult to be used in practice because of the universal quantification on the high level agents. As we have seen, decidability of *PTNDC* depends, in fact, on the possibility of reducing all the high level automata in Γ_H to a finite case suitable for the particular automaton A we would like to study. In [51, 52, 6] other properties have been defined, stronger than the *PTNDC* property, which are easier to check. Such properties, defined for a CCS-like process algebra, discrete-time process algebra and probabilistic process algebra respectively, could be translated within our framework of PTAs.

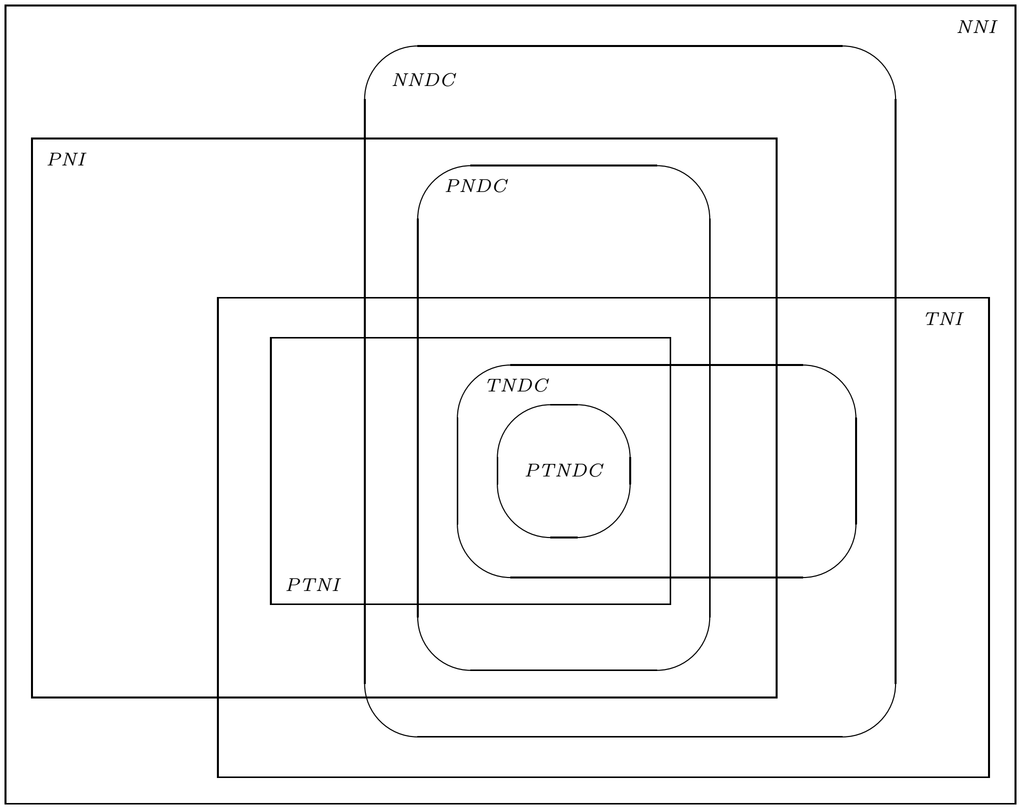

Figure 4.7: Relations among security properties.

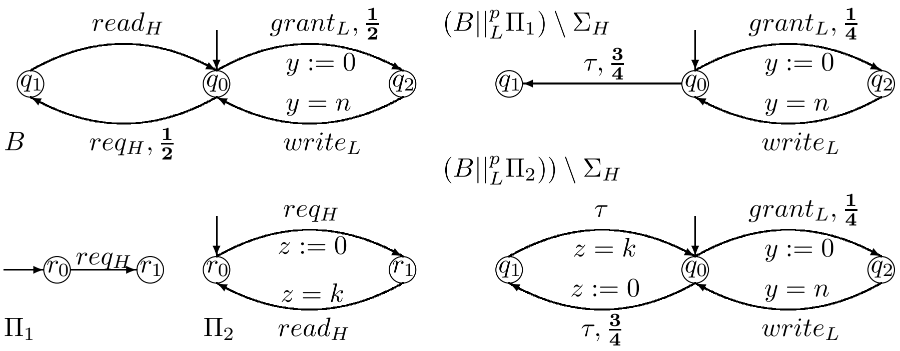

Figure 4.8: Device description with timing covert channels.

4.3 An Application

As an application we consider a network device, also studied in [52] in a timed framework, that manages the access to a shared buffer following a mutual exclusion policy. Assuming that the agents on the network are classified as low and high level agents, the device implements the *no-write-down no-read-up* policy [54]. Intuitively, the policy states that high level users can only read the buffer, while low level users can only write on it. Such a policy avoids direct information flow from high level to low level, however malicious agents can exploit some covert channel in order to transmit information indirectly. For example, a low level user could get information about the high level activity by observing the amount of time the device is locked (non accessible by the low level) when high agents are reading, or by observing the frequencies with which high level agents make access on it. We would like to check whether some covert channel can be exploited, by giving a description of the network device and then checking the $PTNDC$ property.

In the following we consider only a low level user and a high level user communicating with the network device. We assume that the low level user is always ready to write in the buffer, so we consider an agent that infinitely waits for a grant from the device and then writes in the buffer. In this manner we are considering a low level user that continuously monitors the activity of the device. We also assume that the entire procedure of receiving a grant in the network and writing in the buffer is executed in a time n. In Figure 4.8, we model a simple device (see the PTA B). Actions req_H, $read_H$, $grant_L$ and $write_L$ model respectively high level read requests, high level reads, low level write grants and low level writes. The set Σ_H of high level actions is $\{req_H, read_H\}$. We assume $Inv(q_0) = Inv(q_1) = true$ and $Inv(q_2) = y \leq n$. The device B is always ready to accept an access request from the high level agent with probability $\frac{1}{2}$ and to grant a write access to the low level user with the same probability. Obviously, we always consider the device composed with a high level agent according to $||^p_L$ (we assume $p = \frac{1}{2}$ and $L = \{req_H, read_H\}$). On the one hand, when the device is composed with a high level agent that performs action req_H with probability 1, it synchronizes with the high agent accepting his request with probability $\frac{3}{4}$. On the other hand, if the high level agent does not perform req_H, the composed system performs action $grant_L$ with probability 1. As a consequence, we can find out the following covert channels. Consider the high agent Π_1 of Figure 4.8, which executes a read request without performing the reading afterwards. System $(B||^p_L\Pi_1) \setminus \Sigma_H$ reaches a deadlock state that is not reached by B/Σ_H. In this way, the high level agent could transmit the bit 0 or 1 by alternatively blocking or not the device. Such a covert channel can be detected by the $PTNDC$ property, in fact we have that $B/\Sigma_H \not\approx (B||^p_L\Pi_1) \setminus \Sigma_H$, so that $B \notin PTNDC$. Another interesting covert channel arises if one considers Π_2, which locks the buffer and executes a reading only after a time k. A low level user observing the behavior of $(B||^p_L\Pi_2) \setminus \Sigma_H$ does not receive any grant access for a time k when a req_H action is performed. In this way the high level agent could indirectly transmit value k to the low level user. We obviously have again that $B/\Sigma_H \not\approx (B||^p_L\Pi_2) \setminus \Sigma_H$.

The two covert channels discussed above could be avoided by introducing a timeout mechanism which releases the device if $read_H$ is not performed and by always releasing the device after a fixed

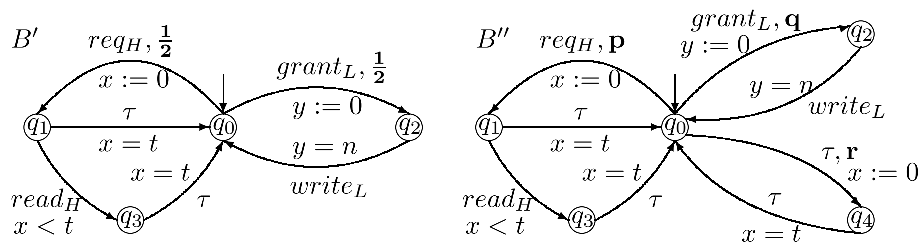

Figure 4.9: Improved device descriptions.

amount of time has passed. In Figure 4.9 we show a device B' that accepts a high level request, and uses a clock x as timer and t as timeout. When it executes action req_H the timer is set to 0, action $read_H$ could be performed only when $x < t$, and when x reaches value t the device unlocks the buffer going back to q_0. When transitions starting from a given state have disjoint conditions we omit probabilities since their execution depends on the time configuration, rather than on the effective probability. We assume $Inv(q_0) = true$, $Inv(q_1) = Inv(q_3) = x \leq t$ and $Inv(q_2) = y \leq n$. The timing covert channels shown in the previous case could not be exploited anymore, however device B' is still insecure. In fact the device is unavailable for the fixed amount of time when a high level access is performed, and this is clearly observable by the low level user that has to wait the termination of the high level request before obtaining access to the buffer. This represents a typical situation where the unavailability of a shared resource can be encoded as 0 or 1 in order to transmit data. Such a situation is captured by the *PTNDC* property by considering again the automaton Π_2 and assuming $k < t$. In fact we have again that $B'/\Sigma_H \not\approx (B'||^p_L \Pi_2) \setminus \Sigma_H$.

The capacity of such a covert channel could be reduced, but not totally avoided, by considering a buffer that probabilistically locks himself without any high level request. In this manner the low level user could not be sure whether the buffer is really locked by the high user or not. In Figure 4.9, B'' represents a device that behaves in such a manner, locking himself with a probability $\mathbf{r}$. Again, $Inv(q_0) = true$, $Inv(q_2) = y \leq n$ and $Inv(q_1) = Inv(q_3) = Inv(q_4) = x \leq t$. As we have said, this does not avoid entirely the covert channel, but the knowledge the low level user acquires is affected by some uncertainty. In fact, if the device is locked, the low level user could deduce that the high user locked the device with a certain probability while with probability $\mathbf{r}$ the device has locked himself for simulating a false higher user's activity. In this case, if we resorted to a ε-tolerant weak bisimulation (i.e. for which the probabilities of reaching a certain class from certain configurations are allowed to be different up to the threshold ε [6]), we would be able to give a measure of the probabilistic covert channel, by estimating the probability that the information flow arises according to $\mathbf{r}$, and therefore a measure of the security level of the device.

We can completely hide the high level activity to the low level user by partitioning into two sessions the time in which users can access the buffer. During a *low session*, lasting a fixed amount of time n, only the low level user can access the buffer, then the device goes to the *high session*, where access is reserved, for the same amount of time, to the high level user. This makes impossible for the low level user to discover something about the high level activity, since the same fixed amount of time is reserved to the high session even if the high user does nothing. In Figure 4.10 we specify a buffer B_s that behaves in such a manner: the buffer is reserved for a time t to the low level user and to the high level user alternatively. We assume $Inv(q_0) = Inv(q_2) = Inv(q_3) = x \leq t$ and $Inv(q_1) = y \leq n$. Automaton B_s is *PTNDC*, in fact, for every possible high level user Π, $(B_s||^p_L \Pi) \setminus \Sigma_H \approx B_c \approx B_s/\Sigma_H$. Intuitively, automaton B_c of Figure 4.10 represents the automaton resulting after the parallel composition between B_s and any high level user Π, and, therefore, B_c is weakly bisimilar to B_s composed with any possible high level user Π. Finally, it easy to see that $B_c \approx B_s/\Sigma_H$.

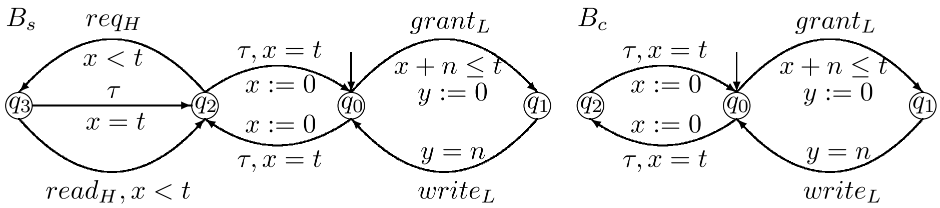

Figure 4.10: Secure Device.

Part II

Enriched Models

Chapter 5

Parametric Probabilistic Transition Systems

Real descriptions are often parametric. Actually, the design of a system may depend on certain parameters of the environment, and concrete instantiations make sense only in the context of a given concrete environment. Moreover, one may want to tune the parameters of a system on the values that allow reaching a desired reliability level, in hardware design, or a security threshold, in protocol design.

In this chapter we develop the model of Parametric Probabilistic Transition Systems (PPTSs). Intuitively, PPTSs are Discrete–time Markov Chains where probabilities associated with transitions may be parameters assuming real values.

It is, in general, undecidable whether there exists a valuation of the parameters appearing in a PPTS that satisfies a given formula. However, we prove that valuations satisfying a given formula can be computed for a subfamily of PPTSs with at most two parameters. Hence, we study the restrictions under which it is possible to compute instantiations of the parameters satisfying a given formula (which may contain also reachability conditions), and we give a syntactical characterization of the PPTSs satisfying those restrictions.

Finally, we propose a technique to compute valuations that maximize or minimize the probability of reaching a certain state of the PPTS.

The class of PPTSs satisfying the restrictions we impose is expressive enough to describe and analyze many real-life systems. One may think, for example, of systems with one or two hardware components which can undermine the reliability of the whole system. If the designer is able to adjust the reliability levels of the components, considered as parameters, our framework allows to analyze and maximize the reliability of the whole system. A similar analysis can be done for the study of reliability and/or security level of probabilistic protocols. In fact, a large class of probabilistic protocols perform just one probabilistic choice, which can be considered as the only parameter of the system. Applications of this type range from academic protocols, e.g. Chaum's dining cryptographers protocol (see [30]) and the probabilistic Non-repudiation protocol introduced in [118], to industrial protocols, e.g. the IPv4 Zeroconf protocol as studied in [23], the Crowds [131] and Onion Routing [129] anonymity protocols, etc..

Summing up, the framework we present allows performing a parametric analysis of a rich class of systems, e.g., checking whether a formula holds for different values of the parameters, and computing the values of the parameters such that some property is satisfied or maximize (minimize) the probability of reaching success (failure) states.

The remainder of this chapter is organized as follows. In Section 5.1 we summarize some related work. In Section 5.2 we recall some basic notions about Discrete–time Markov Chains and polynomial terms with parameters. In Section 5.3 we introduce the model of Parametric Probabilistic Transition Systems by extending Markov Chains with parameters. In Section 5.4 we tackle the problems of the decidability of the existence of instances of parameters that satisfy a given property and we propose a technique to find optimal instances. In Section 5.5, as an application, we show the model of a probabilistic non–repudiation protocol where some probabilistic choices are modeled with parameters.

5.1 Related Works

In [11] Alur et al. address the problem of parametric reasoning on the timing properties of real-time systems. *Parametric Timed Automata* are proposed as a generalization of the *Timed Automata* in [9]. In particular transition guards on clock formulas are allowed to contain parameters. In this setting, the number of clocks is crucial for the decidability of the emptiness problem: for a subclass of Parametric Timed Automata, where just one clock is parametrically constrained, the emptiness problem is shown to be decidable.

In [143], a parametric model checking algorithm is proposed for a subclass of Timed Automata called Parametric Time-Interval Automata (PTIA). In a PTIA, one can specify upper- and lower-bounds of the execution time (time-interval) of each transition by using parameter variables. The proposed algorithm takes two inputs, a model described as a PTIA and a property specified as a PTIA accepting all the invalid runs. The algorithm constructs the parallel composition of the PTIAs expressing the model and the property, and checks the emptiness of the language accepted by the product automaton. The output is the weakest condition on the parameters such that the given model never executes the invalid runs.

The interest about parametric real-time properties has also been extended to model checkers for real-time systems. In [73] Hune et al. introduce an extension of the UPPAAL model checker (see [12]) for analyzing Parametric Timed Automata. The authors identify a subclass of Parametric Timed Automata (called L/U automata), for which the emptiness problem is decidable.

With HyTech [66], a tool for model checking hybrid automata, Henzinger et al. also consider a parametric analysis. However, the tool seems to apply successfully only to relatively small examples.

While the study of parametric real-time systems has been strongly pursued, parametric analysis of probabilistic systems has not been considered until recent years. In 2004, two studies on parametric probabilistic analysis were brought on independently (see [38, 95]).

In [38], Daws presents a language-theoretic approach to symbolic model checking of PCTL formulas over Discrete-time Markov Chains. The probability with which a path formula is satisfied is represented by a regular expression. A recursive evaluation of the regular expression yields an exact rational value when transition probabilities are rational, and rational functions when some probabilities are left unspecified as parameters of the system. This allows for parametric model checking by evaluating the regular expression for different parameter values in the rational domain. Formulas, however, are assumed without nested probabilistic operators. In this chapter, which is an extension of [95], we give an in-depth analysis of parametric probabilistic systems, where parameters are real values. We study decidable subclasses of PPTSs and we propose a technique for minimizing/maximizing the probability of reaching certain states under a given condition on the parameters.

5.2 Basic notions

In this section we recall some basic notions about Discrete-time Markov chains (DTMCs) [20, 71] and polynomial terms with parameters.

5.2.1 Discrete-time Markov Chains

In this section we recall the definition of Discrete-time Markov chains (DTMCs) [20, 71].

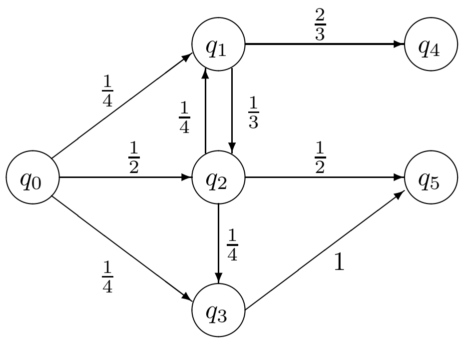

Figure 5.1: A Discrete-time Markov Chain.

Definition 5.1 *A* Discrete-time Markov chain *(DTMC) is a tuple* (Q, q_0, δ, π) *where:*

- Q *is a finite set of states;*
- $q_0 \in Q$ *is the initial state;*
- $\delta \subseteq Q \times Q$ *is a set of* transitions*;*
- $\pi : \delta \rightarrow [0, 1]$ *is a function assigning a probability to each transition. For all states* $q \in Q$ *it is required that* $\sum_{e \in Start(q)} \pi(e) \in \{0, 1\}$.

With $Start(q)$ we denote the set of transitions with q as source state, namely $Start(q) = \{(q_i, q_j) \in \delta \mid q_i = q\}$.

Example 5.1 *Consider the DTMC M depicted in Figure 5.1. The set of transitions of M is:* $\delta = \{(q_0, q_1), (q_0, q_2), (q_0, q_3), (q_1, q_2), (q_1, q_4), (q_2, q_1), (q_2, q_5), (q_3, q_5)\}$. *The probability of a transition appears over the transition. We have, e.g.,* $\pi((q_0, q_2)) = \pi((q_2, q_5)) = \frac{1}{2}$.

A *run* of a DTMC $M = (Q, q_0, \delta, \pi)$ is a (possibly infinite) sequence of steps of the form $\omega = q_0 \rightarrow q_1 \rightarrow \ldots$ where (q_i, q_{i+1}) is in δ and $\pi((q_i, q_{i+1})) > 0$. The length of ω, denoted $|\omega|$, is the number of transitions between states performed by the run and is equal to n if ω is the finite run $q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n$, and ∞ otherwise. With $Path_{fin}(M)$ (resp. $Path_{ful}(M)$) we denote the set of finite (resp. infinite) runs of M.

Let $k \leq |\omega|$; with $\omega(k)$ we denote the state q_k and with $\omega^{(k)}$ we denote the run $q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_k$.

If $k = |\omega|$, then we say that ω *is a prefix of* ω' if and only if $length(\omega') \geq k$ and $\omega = (\omega')^{(k)}$. With $last(\omega)$ we denote the state $\omega(k)$.

Assuming the basic notions of probability theory (see e.g. [59]), we may assign a probability measure to the runs of a DTMC M by following the traditional *Borel σ-algebra* approach of basic cylinders sets (see [78, 148]). We denote the probability of a finite run $\omega = q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n$ with $\mu(\omega)$, defined as follows:

$$\mu_M(\omega) = \begin{cases} 1 & \text{if } n = 0 \\ \mu_M(\omega^{(n-1)}) \cdot \pi((q_{n-1}, q_n)) & \text{if } n > 0 \end{cases}$$

Finally, we can extend the notion of measure to sets of infinite runs. The measure μ defined on sets of runs in $Path_{ful}(M)$ is the unique measure such that

$$\mu_M(\{\omega' \mid \omega' \in Path_{ful}(M) \ \wedge \ \omega \text{ is a prefix of } \omega'\}) = \mu_M(\omega)$$

for any $\omega \in Path_{fin}(M)$.

Models as DTMCs, with at most one distribution per state, are said to be *fully probabilistic.* Note that in the model of DTMCs, only probabilistic choices are allowed, i.e. no non-determinism is considered.

In the following we focus on infinite paths. Notice that, in this case, states with no outgoing transitions can be considered by adding a self-loop transition with probability 1.

Definition 5.2 *Given a DTMC M and a state q, with $P_M(q)$ we denote the probability of reaching the state q from the initial state of M, more precisely:*

$$P_M(q) = \mu(\{\omega \in Path_{ful}(M) \mid \exists k : \omega(k) = q\}).$$

We note that the set $\{\omega \in Path_{ful}(M) \mid \exists k : \omega(k) = q\}$ is measurable, and hence the probability $P_M(q)$ is well defined.

With $Adm(q) \subseteq Q$ we denote the set of states that can be crossed for eventually reaching the state q from the initial state q_0. More precisely,

$$Adm(q) = \{q' \mid q_0 \to \ldots \to q' \to \ldots \to q \in Path_{fin}(M)\}.$$

Moreover, with $AdmTr(q', q) \subseteq \delta$ we denote the set of transitions starting from q' and reaching a state in $Adm(q)$, more precisely

$$AdmTr(q', q) = \{(q', q'') \in \delta \mid q'' \in Adm(q)\}.$$

Example 5.2 *Let us consider the DTMC M of the example in Figure 5.1. We have that $Adm(q_4) = \{q_0, q_1, q_2, q_4\}$. Moreover, we have that $AdmTr(q_1, q_4) = \{(q_1, q_2), (q_1, q_4)\}$ and $AdmTr(q_2, q_4) = \{(q_2, q_1)\}$.*

The following proposition has been proved in [61].

Proposition 5.1 *Let $M = (Q, q_0, \delta, \pi)$ be a DTMC and $q \in Q$; the probability $P_M(q)$ is equal to the solution of x_{q_0} of the following system of linear equations:*

$$\begin{cases} x_q = 1 \\ x_{q'} = \sum_{(q',q'') \in AdmTr(q',q)} \pi((q', q'')) \cdot x_{q''} \quad \forall q' \neq q \end{cases}.$$

5.2.2 Parameters

With $\alpha, \beta, \ldots$ we denote *parameters* assuming values in the set $\mathbb{R}$ of real numbers. Given a set of parameters Δ, an *instance* $u : \Delta \to \mathbb{R}$ for Δ is a function assigning a real value to each parameter in Δ.

We define the set $\mathcal{P}(\Delta)$ of *polynomial terms* over parameters in Δ as follows:

$$\tau ::= c \mid \alpha \mid \tau_1 + \tau_2 \mid \tau_1 \cdot \tau_2$$

where $\tau, \tau_1, \tau_2 \in \mathcal{P}(\Delta)$, $c \in \mathbb{R}$ and $\alpha \in \Delta$. A polynomial term τ is a *linear term* if there exist $c_1, \ldots, c_{n+1} \in \mathbb{R}$ and $\alpha_1, \ldots, \alpha_n \in \Delta$ such that τ is syntactically equal to $c_1 \cdot \alpha_1 + \ldots + c_n \cdot \alpha_n + c_{n+1}$.

An instance u extends to $\mathcal{P}(\Delta)$ as follows: $u(c) = c$, $u(\tau_1 + \tau_2) = u(\tau_1) + u(\tau_2)$ and $u(\tau_1 \cdot \tau_2) = u(\tau_1) \cdot u(\tau_2)$.

We define the set $\Phi(\Delta)$ of *formulae* as follows:

$$\phi ::= \tau \sim \tau' \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2$$

where ϕ, ϕ_1, ϕ_2 range over $\Phi(\Delta)$, τ, τ' are in $\mathcal{P}(\Delta)$, and $\sim \in \{<, \leq, =, \geq, >\}$. A formula ϕ in $\Phi(\Delta)$ is *linear* iff all terms $\tau \in \mathcal{P}(\Delta)$ appearing in ϕ are linear.

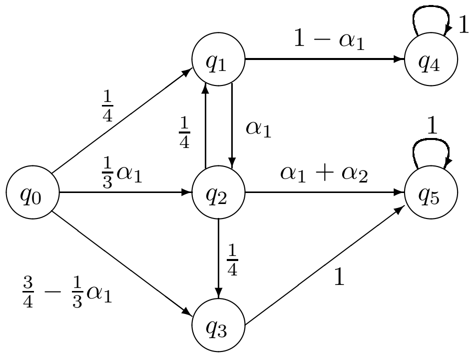

Figure 5.2: A Parametric Probabilistic Transition System.

Let $\phi \in \Phi(\Delta)$ and u be an instance; we say that u *satisfies* ϕ, written $u \models \phi$, iff:

$$\begin{array}{lll} u \models \tau \sim \tau' & \text{iff} & u(\tau) \sim u(\tau') \\ u \models \neg\phi_1 & \text{iff} & u \not\models \phi_1 \\ u \models \phi_1 \vee \phi_2 & \text{iff} & \text{either } u \models \phi_1 \text{ or } u \models \phi_2 \\ u \models \phi_1 \wedge \phi_2 & \text{iff} & \text{both } u \models \phi_1 \text{ and } u \models \phi_2. \end{array}$$

Definition 5.3 (Equivalence) *Two formulae $\phi_1, \phi_2 \in \Phi(\Delta)$ are* equivalent *(denoted $\phi_1 \equiv \phi_2$) if, for any instance u, it holds that $u \models \phi_1$ if and only if $u \models \phi_2$.*

A known property of formulae in Φ is the following (see [144] and [133]).

Theorem 5.1 *For each $\phi \in \Phi(\Delta)$, it is decidable in exponential time w.r.t. the number of parameters in ϕ and in polynomial space whether there exists an instance u such that $u \models \phi$.*

5.3 Parametric Probabilistic Transition Systems

In this section we introduce the model of Parametric Probabilistic Transition Systems.

Definition 5.4 *A* Parametric Probabilistic Transition System *(PPTS) S is a tuple $(\Delta, Q, q_0, \delta, \lambda)$ where:*

- *Δ is a finite set of parameters;*
- *Q is a set of states;*
- *$q_0 \in Q$ is the initial state;*
- *$\delta \subseteq Q \times Q$ is a set of* transitions*;*
- *$\lambda : \delta \to \mathcal{P}(\Delta)$ is a function assigning to each transition (q, q') a polynomial term τ representing the probability of taking that transition.*

With Start(q) $\subseteq \delta$ we denote the set of transitions with q as source state, namely the set $\{(q_i, q_j) \in \delta \mid q_i = q\}$. The PPTS S is linear *if $\lambda(e)$ is linear, for any $e \in \delta$.*

Example 5.3 *Let us consider the PPTS S of Figure 5.2. We have parameters $\Delta = \{\alpha_1, \alpha_2\}$, and, as examples, Start(q_2) $= \{(q_2, q_1), (q_2, q_3), (q_2, q_5)\}$ and $\lambda((q_2, q_5)) = \alpha_1 + \alpha_2$.*

The notions of *run* of a PPTS S and of sets of finite and infinite runs $Path_{fin}(S)$ and $Path_{ful}(S)$, respectively, are defined as for DTMCs.

The following definitions relate instances of parameters to PPTSs.

Definition 5.5 (Instances of PPTSs) *Given a PPTS $S = (\Delta, Q, q_0, \delta, \lambda)$ and an instance u, with $S(u)$ we denote the PPTS resulting by instantiating all the variables in Δ according to u. Namely, $S(u) = (\emptyset, Q, q_0, \delta, \lambda')$, where $\lambda'(e) = u(\lambda(e))$ for any $e \in \delta$.*

Definition 5.6 (Well Defined Instances) *Given an instance u and a PPTS $S = (\Delta, Q, q_0, \delta, \lambda)$, we say that u is* well defined *for S if and only if for each transition $e \in \delta$ we have that $u(\lambda(e)) \in [0, 1]$, and, for each state $q \in Q$, it holds that $\sum_{e \in Start(q)} u(\lambda(e)) = 1$.*

Example 5.4 *The instance u_1 such that $u_1(\alpha_1) = u_1(\alpha_2) = \frac{1}{4}$, and the instance u_2 such that $u_2(\alpha_1) = 0$, $u_2(\alpha_2) = \frac{1}{2}$ are well defined for the PPTS of Figure 5.2. The instance u_3 such that $u_3(\alpha_1) = u_3(\alpha_2) = 1$ is not well defined.*

Definition 5.7 (Realizability) *A PPTS S is* realizable *if and only if there exists a well defined instance for S. Given a formula $\phi \in \Phi(\Delta)$, S is* ϕ-realizable *if and only if there exists a well defined instance u for S, such that $u \models \phi$ (in this case, we say that u* ϕ-realizes *S).*

Remark: Given a well defined instance u for a PPTS S, we have that $S(u)$ is a DTMC.

Note that, as we did for DTMCs by restricting to infinite runs, we assumed that for any well defined instance u, $\sum_{e \in Start(q)} u(\lambda(e)) = 1$. As we have seen, this is not a real limitation, since for every state q_s with no transition to other states, we can add a self-loop transition (q_s, q_s) with $u(\lambda((q_s, q_s))) = 1$ (see states q_4 and q_5 in Figure 5.2).

As for DTMCs, we may assign a probability measure to the runs of a PPTS S for a well defined instance u. If $\omega \in Path_{fin}(S)$ and u is a well defined instance for S, then we denote with $\mu_{S(u)}(\omega)$ the probability of ω according to u, namely the probability of the run ω in the DTMC $S(u)$.

In the same way, given a well defined instance u, we can extend the notion of measure to sets of infinite runs in $Path_{ful}(S)$. Namely,

$$\mu_{S(u)}(\{\omega' \mid \omega' \in Path_{ful}(S(u)) \ \wedge \ \omega \text{ is a prefix of } \omega'\}) = \mu_{S(u)}(\omega)$$

for any $\omega \in Path_{fin}(S(u))$.

Given a PPTS S, a state q of S and a well defined instance u, $P_{S(u)}(q)$ returns the probability of reaching the state q for the DTMC $S(u)$.

The following corollary derives by Proposition 5.1.

Corollary 5.1 *Given a PPTS $S = (\Delta, Q, q_0, \delta, \lambda)$ a state $q \in Q$ and a well defined instance u, the probability $P_{S(u)}(q)$ of reaching the state q is equal to the solution of x_{q_0} of the following system of linear equations:*

$$\begin{cases} x_q = 1 \\ x_{q'} = \sum_{(q', q'') \in AdmTr(q', q)} u(\lambda((q', q''))) \cdot x_{q''} \quad \forall q' \neq q \end{cases}$$

5.4 ϕ-Realizability of Parametric Probabilistic Transition Systems

In this section we consider the problem of computing the probability of reaching a certain state. We tackle this problem in a parametric setting, hence we consider existence, search and optimization of well defined instances.

Our results are obtained under the assumption that the number of parameters is at most two. However, we notice that such a class of systems is expressive enough to describe and analyze many real-life systems. Moreover, if one faces systems with more than two parameters, the method we propose can still be useful insofar as the system considered may be studied by fixing the value of all parameters but two. Finally, it is often possible to decompose the system into independent components with at most two parameters (e.g., see the analysis of the IPv4 zeroconf protocol in [23]).

5.4.1 The Problem of Existence of an Instance

Given a PPTS, firstly we want to know whether the PPTS is realizable or not. In particular we are interested in the existence of well defined instances for the PPTS. The next propositions follows.

Proposition 5.2 (Realizability) *It is decidable in exponential time w.r.t. the number of parameters and in polynomial space whether a PPTS S is realizable.*

Proof. Given a PPTS $S = (\Delta, Q, q_0, \delta, \lambda)$, we build the formula:

$$\phi_R = (\bigwedge_{e \in \delta} \lambda(e) \in [0,1]) \wedge (\bigwedge_{q' \in Q} \sum_{e \in Start(q')} \lambda(e) = 1) \tag{5.1}$$

representing the requirements for the realizability of S.

An instance u satisfying the formula ϕ_R ($u \models \phi_R$) is well defined for S. By Theorem 5.1, the existence of an instance u that satisfies ϕ_R is decidable in exponential time. Hence, it is also decidable in exponential time w.r.t. the size of S whether S is realizable.

Proposition 5.2 can be extended to deal with ϕ-realizability.

Corollary 5.2 *It is decidable in exponential time whether a PPTS is ϕ-realizable .*

Proof. Given a PPTS $S = (\Delta, Q, q_0, \delta, \lambda)$, and a formula ϕ in $\Phi(\Delta)$, we build the formula:

$$\bar{\phi} = \phi \wedge \phi_R \tag{5.2}$$

where ϕ_R is the formula of Equation (5.1).

An instance u satisfying the formula $\bar{\phi}$ is well defined for S and satisfies ϕ. Hence, u ϕ-realizes S. Again, by Theorem 5.1, the existence of an instance u that satisfies $\bar{\phi}$ is decidable in exponential time, and hence it is also decidable in exponential time w.r.t. the size of S whether S is ϕ-realizable.

Corollary 5.2 can be then extended to deal with the reachability of states of PPTSs.

Corollary 5.3 (Existence) *For any PPTS S, state q and formula $\phi \in \Phi(\Delta)$, it is decidable in exponential time w.r.t. the size of S whether there exists an instance u that ϕ-realizes S such that q is reachable in $S(u)$.*

Proof. Given a PPTS $S = (\Delta, Q, q_0, \delta, \lambda)$, a state $q \in Q$ and a formula ϕ, we build the formula $\bar{\phi}$ as follows:

$$\bar{\phi} = \phi \ \wedge \ \phi_R \wedge \phi_S \tag{5.3}$$

where ϕ_R is again the formula in Equation (5.1) and ϕ_S is the formula:

$$\phi_S = (x_q = 1) \ \wedge \ (\bigwedge_{q' \in Adm(q)} x_{q'} = \sum_{(q',q'') \in AdmTr(q',q)} \lambda((q',q'')) \cdot x_{q''}). \tag{5.4}$$

Here, formula ϕ_S is such that for any $q' \in Q$, $x_{q'}$ is the polynomial term with variables in Δ modelling the probability of reaching q from q'.

An instance u satisfying the formula $\bar{\phi}$ is such that $u \models \phi \wedge \phi_R$ (hence, for Corollary 5.2, u ϕ-realizes S). Moreover, we have that if $u \models \bar{\phi}$, then x_{q_0} is the polynomial term modelling the probability of reaching state q. By Theorem 5.1, the existence of an instance u that satisfies $\bar{\phi}$ is decidable in exponential time, hence it is also decidable in exponential time w.r.t. the size of S whether there exists an instance u that ϕ-realizes S such that q is reachable in $S(u)$.

Definition 5.8 *With $Set(S, q, \phi) = \{u \mid u \models \bar{\phi}\}$, where $\bar{\phi}$ is the formula in Equation (5.3), we denote the set of well defined instances u such that u ϕ-realizes the PPTS S and the state q is reachable in $S(u)$.*

Example 5.5 *Let us consider the PPTS S in Figure 5.2. We want to know whether there exists an instance in the set $Set(S, q_5, (x_{q_0} > \alpha_1 \wedge \alpha_1 > 0))$. This set is not empty if and only if the following formula is satisfiable:*

$$
\begin{array}{l}
x_{q_0} > \alpha_1 \ \wedge \ \alpha_1 > 0 \\
\alpha_1 \leq 1 \ \wedge \ \alpha_1 + \alpha_2 = \frac{1}{2} \\
\wedge \ x_{q_5} = 1 \\
\wedge \ x_{q_3} = x_{q_5} \\
\wedge \ x_{q_2} = (\alpha_1 + \alpha_2)x_{q_5} + \frac{1}{4}x_{q_3} + \frac{1}{4}x_{q_1} \\
\wedge \ x_{q_1} = \alpha_1 x_{q_2} \\
\wedge \ x_{q_0} = \frac{1}{4}x_{q_1} + \frac{1}{3}\alpha_1 x_{q_2} + (\frac{3}{4} - \frac{1}{3}\alpha_1)x_{q_3}.
\end{array}
$$

Now, the formula above is a formula in $\Phi(\{\alpha_1, \alpha_2\})$, and hence, by Theorem 5.1, its satisfiability is decidable.

5.4.2 Finding a Solution

Given a PPTS S, a state q and a formula ϕ, we now consider the problem of finding an instance in $Set(S, q, \phi)$ such that the probability of reaching the state q is equal to a certain value $c \in [0, 1]$. Actually, Theorem 5.3 answers the problem of existence of an instance but does not give one. To find an instance u in $Set(S, q, \phi)$ such that $P_{S(u)}(q) = c$, is a harder problem with respect to the problem of existence of an instance. More precisely, to find an instance in $Set(S, q, \phi)$ is in general undecidable.

The following theorem, which derives from Galois' theory, states that it is in general impossible to find the roots of a polynomial of degree higher than 4 (see [140]).

Theorem 5.2 *It is impossible to give a general algebraic formula to solve polynomials of degree 5 and higher.*

The next proposition follows directly from Theorem 5.2.

Proposition 5.3 *Given a PPTS S, a state q and a formula ϕ, the problem of finding an instance u such that $u \in Set(S, q, \phi)$ is in general unsolvable.*

Proof. The problem of finding an instance $u \in Set(S, q, \phi)$ is equivalent to the problem of finding the roots of a general polynomial.

Hence, to have decidability, we must consider some restrictions. In particular, we give a restriction on the degrees of the polynomials generated by a PPTS.

Definition 5.9 (Degree of a Polynomial Term) *Let τ be a polynomial term and β be a parameter. The* degree *of τ w.r.t. β, denoted with $dg(\tau, \beta)$, is the natural n such that $\tau = c_n \cdot \beta^n + \ldots + c_1 \cdot \beta + c_0$ and $c_n \neq 0$.*

Let $Q = \{q_1, \ldots, q_n\}$ be the states of S such that q_n is equal to q.

By Corollary 5.1 $P_{S(u)}(q)$ is the solution of x_{q_0} in the following system of linear equations:

$$
I = \begin{cases} x_q = 1 \\ x_{q'} = \sum_{(q',q'') \in AdmTr(q',q)} \lambda((q', q'')) \cdot x_{q''} \quad \forall q' \neq q \end{cases}
$$

The system I can be written as $Ax = B$, where x is the vector of the variables $x_{q_1}, \ldots, x_{q_n}$, B is the vector of naturals representing the known terms, and A is the matrix such that the $A[i, j] = \lambda((q_i, q_j))$ if $(q_i, q_j) \in AdmTr(q_i, q)$ and $A[i, j] = 0$ otherwise.

Let A' be the matrix obtained from A by substituting the first column with B.

Let τ_1 be the polynomial computed as the determinant of the matrix A' and τ_2 be the polynomial computed as the determinant of the matrix A.

By using Cramer's rule, we prove the following proposition.

Proposition 5.4 *Given a PPTS $S = (\Delta, Q, q_0, \delta, \lambda)$ and a state $q \in Q$, two polynomials $\tau_1, \tau_2 \in \mathcal{P}(\Delta)$ can be found in polynomial time w.r.t. the size of S, such that, for any well defined instance u, it holds that $P_{S(u)}(q) = \frac{u(\tau_1)}{u(\tau_2)}$.*

Proof. By Corollary 5.1 and following Equation (5.4), we have that the possible values that $P_{S(u)}(q)$ can assume for any well defined instance u are those of the variable x_{q_0} of the system of equations

$$x_q = 1 \wedge \bigwedge_{q' \in Adm(q)} x_{q'} = \sum_{(q',q'') \in AdmTr(q',q)} \lambda((q',q'')) \cdot x_{q''}.$$

This can be solved as a system of linear equalities, and hence $x_{q_0} = \frac{\tau_1}{\tau_2}$ for some $\tau_1, \tau_2 \in \mathcal{P}(\Delta)$.

Example 5.6 *Let us consider the PPTS S of example of Figure 5.2. We have that $P_{S(u)}(q_5) = \frac{u(\tau_1)}{u(\tau_2)}$ where $\tau_1 = 7\alpha_1(\alpha_1 + \alpha_2 + \frac{1}{4}) + (12 - 3\alpha_1)(\frac{3}{4} - \frac{1}{3}\alpha_1)$ and $\tau_2 = 12 - 3\alpha_1$.*

We define now the degree of a PPTS S w.r.t. a state and a parameter.

Definition 5.10 (Degree of a PPTS) *For a PPTS S,* the degree of S for parameter α and state q *(written $dg(S, \alpha, q)$) is equal to the value $max(dg(\tau_1, \alpha), dg(\tau_2, \alpha))$, where τ_1 and τ_2 are the polynomials of Proposition 5.4.*

Example 5.7 *Let us consider the PPTS S of example of Figure 5.2. We have that $dg(S, \alpha_1, q_5) = 2$ and $dg(S, \alpha_2, q_5) = 1$.*

One Parameter

Theorem 5.3 *Let $S = (\{\alpha\}, Q, q_0, \delta, \lambda)$ be a PPTS and q be a state in Q such that $dg(S, \alpha, q) \leq 4$.*

Let ϕ be a general formula and $c \in [0, 1]$; a well defined instance u, such that $u \in Set(S, q, \phi)$ and $P_{S(u)}(q) = c$, can be found in polynomial time w.r.t. the size of S.

Proof. By Proposition 5.4 we have that $P_{S(u)}(q) = \frac{u(\tau_1)}{u(\tau_2)}$. Since we are looking for an instance u such that $P_{S(u)}(q) = c$, we must solve the equation $c = \frac{\tau_1}{\tau_2}$, but this is equivalent to $\tau_1 - c \cdot \tau_2 = 0$ when $\tau_2 \neq 0$.

The steps of the proof are the following:

1. $\tau_1 - c \cdot \tau_2 = 0$ is a polynomial of degree 4 and hence one can compute the finite set C of its roots.

2. We can check, for any root r in C, whether r satisfies $\tau_2 \neq 0$, ϕ and the definition od DTMC.

The set C of roots of the polynomial $\tau_1 - c \cdot \tau_2$ of degree at most 4 on the only parameter α is computable (see [140]). Obviously $|C| \leq dg(S, \alpha, q) \leq 4$. Let U be the set of instances u such that $u(\alpha) \in C$. Obviously $|U| \leq 4$. Let U' be the set of instances $u \in U$ such that $u \models \tau_2 \neq 0 \wedge \phi$; we have that $u \models \tau_2 \neq 0 \wedge \phi$ can be checked in polynomial time on the length on ϕ and τ_2, by using the definition of $\models$. Since $|U| \leq 4$, U' can be computed in polynomial time.

Now, we must look for a $u \in U'$ such that $S(u)$ is a DTMC. To do that it suffices to check whether there exists $u \in U'$ such that

$$u \models \bigwedge_{e \in \delta} \lambda(e) \in [0, 1] \wedge \bigwedge_{q' \in Q} \sum_{e \in Start(q')} \lambda(e) = 1.$$

Obviously, this can be checked in polynomial time on the size of S, by using the definition of $\models$.

Example 5.8 *Consider the PPTS S of Figure 5.2 where $\alpha_2 = 0$. We look for an instance $u \in Set(S, q_4, \alpha_1 \geq \frac{1}{2})$ such that $P_{S(u)}(q_4) = \frac{1}{6}$. We have that $dg(S, \alpha_1, q) = 2$. Actually, $P_{S(u)}(q_4) = \frac{u(\tau_1)}{u(\tau_2)}$, where $\tau_1 = \alpha_1^2 + 2\alpha_1 - 3$ and $\tau_2 = 3\alpha_1 - 12$.*

Hence we must find a value for α_1 such that $\frac{\tau_1}{\tau_2} = \frac{1}{6}$, which is equivalent to solving the equation $6\alpha_1^2 + 9\alpha_1 - 6 = 0$. The solutions are $\alpha_1 = -2$ and $\alpha_1 = \frac{1}{2}$. We must check whether these solutions are admissible. First of all we require that $\alpha_1 \geq \frac{1}{2}$. Hence $\alpha_1 = -2$ is not an admissible solution (note that a valuation with $\alpha_1 = -2$ is also not well defined). Now, it is easy to check that for $\alpha_1 = \frac{1}{2}$ we have that $\lambda(e) \in [0, 1]$, for all transitions e, and $\sum_{e \in Start(q')} \lambda(e) = 1$, for each state q'.

In the following lemma we give a syntactical characterization of PPTSs in terms of $dg(S, \alpha, q)$, allowing to check whether the hypotheses of Theorem 5.3 are satisfied. The idea is that, given a state q of a PPTS S with one only parameter, if the sum on the states q' of S of the maximal degree of polynomials labelling a transition in $AdmTr(q', q)$ is less than or equal to c, then $dg(S, \alpha, q) \leq c$. Obviously, if $c \leq 4$ Theorem 5.3 can be applied.

As an example, let us consider the PPTS S of Figure 5.2 with $\alpha_2 = 0$ (S has only one parameter, namely α_1). If we are interested in state q_4, the maximal degree of polynomials labelling a transition in $AdmTr(q_0, q_4)$ is 1, in $AdmTr(q_1, q_4)$ is 1, and in $AdmTr(q_2, q_4)$ is 0. Actually, in Example 5.8 we have already seen that $dg(S, \alpha_1, q) = 2$.

Lemma 5.1 *Let $S = (\Delta, Q, q_0, \delta, \lambda)$ be a PPTS, q be a state and $\alpha \in \Delta$ such that*

$$\left(\sum_{q' \in Q} max\{dg(\lambda(e), \alpha) | e \in AdmTr(q', q)\} \right) \leq c,$$

for some natural number c, then it holds that $dg(S, \alpha, q) \leq c$.

Proof. The steps of the proof are the following: by Proposition 5.4, τ_1 and τ_2 are computed as the determinants of A' and A, respectively. We compute the determinant of A by rows, hence, by induction on the number of rows in the algorithm for computing the determinant, we prove that $dg(\tau_2, \alpha) \leq c$ and $dg(\tau_1, \alpha) \leq c$, and hence $dg(S, \alpha, q) \leq c$.

Therefore, it is sufficient to prove that $dg(\tau_1, \alpha) \leq c$ and $dg(\tau_2, \alpha) \leq c$.

Now, the maximal degree that appears in each row i is $max\{dg(\lambda(e), \alpha) \,|\, e \in AdmTr(q_i, q)\}$. By induction on the number of rows in the algorithm for computing the determinant we have that:

$$dg(\tau_2, \alpha) \leq \left(\sum_{i=1}^{n} max\{dg(\lambda(e), \alpha) \,|\, e \in AdmTr(q_i, q)\} \right) =$$

$$= \left(\sum_{q' \in Q} max\{dg(\lambda(e), \alpha) \,|\, e \in AdmTr(q', q)\} \right) \leq c.$$

Hence, $dg(\tau_2, \alpha) \leq c$, and similarly we can prove that $dg(\tau_1, \alpha) \leq c$. Therefore $dg(S, \alpha, q) \leq c$.

Thus, by Lemma 5.1 and Theorem 5.3, we have the following theorem that gives a syntactical characterization of the PTTSs for which it is possible to find a solution.

Theorem 5.4 *Let $S = (\{\alpha\}, Q, q_0, \delta, \lambda)$ be a PPTS and q be a state such that*

$$\left(\sum_{q' \in Q} max\{dg(\lambda(e), \alpha) \,|\, e \in AdmTr(q', q)\} \right) \leq 4.$$

For any general formula ϕ and $c \in [0, 1]$, a well defined instance u, such that $u \in Set(S, q, \phi)$ and $P_{S(u)}(q) = c$ can be found in polynomial time w.r.t. the size of S.

Two Parameters

We extend the result of Theorem 5.3 to the case of two parameters. To do this, we must restrict to linear formulae and linear PPTS.

Theorem 5.5 *Let $S = (\{\alpha_1, \alpha_2\}, Q, q_0, \delta, \lambda)$ be a linear PPTS and q be a state in Q such that $dg(S, \alpha_1, q) \in [1, 3]$ and $dg(S, \alpha_2, q) = 1$. Let ϕ be a linear formula and $c \in [0, 1]$; a well defined instance u, such that $u \in Set(S, q, \phi)$ and $P_{S(u)}(q) = c$, can be found in polynomial time w.r.t. the size of S.*

Proof. By Proposition 5.4 we have that $P_{S(u)}(q) = \frac{u(\tau_1)}{u(\tau_2)}$. Since we are looking for an instance u such that $P_{S(u)}(q) = c$, we must solve the equation $c = \frac{\tau_1}{\tau_2}$, but this is equivalent to $\tau_1 - c \cdot \tau_2 = 0$ when $\tau_2 \neq 0$.

The steps of the proof are the following:

1. $\tau_1 - c \cdot \tau_2 = 0$ is a polynomial of degree 1 on α_2 hence one can easily express α_2 in terms of α_1. More precisely, $\tau_1 - c \cdot \tau_2 = 0$ can be written as the polynomial $a_1\alpha_2 + a_2 = 0$ where a_1 and a_2 are two polynomial on the only parameter α_1 of degree at most 3.
2. We substitute α_2 with $-\frac{a_2}{a_1}$ in $\tau_2 \neq 0$ and in ϕ. We have now formulae on the only parameter α_1 and with degree less or equal to 4.
3. Since it is decidable to find the roots of a polynomial on only one parameter and of degree less or equal to 4, one can find a finite union of intervals expressing the values that α_1 can assume. Hence, fixed α_1, we find α_2 thanks to the expression $\alpha_2 = -\frac{a_2}{a_1}$.

By hypothesis, $\tau_1 - c \cdot \tau_2$ has degree equal to 1 for parameter $\alpha_2 \in \Delta$. Hence, $\tau_1 - c \cdot \tau_2 = 0$ can be written as the polynomial $a_1\alpha_2 + a_2 = 0$ where a_1 and a_2 are two polynomials on the only parameter α_1 of degree at most 3.

As for Theorem 5.3, an instance u must satisfy $\tau_2 \neq 0$, the formula ϕ, and the requirements for S to be realizable (see ϕ_R in Equation (5.1)). Hence, the following ϕ' must be satisfied:

$$(\tau_2 \neq 0) \wedge \phi \wedge (\bigwedge_{e \in \delta} \lambda(e) \in [0, 1]) \wedge (\bigwedge_{q' \in Q} \sum_{e \in Start(q')} \lambda(e) = 1) \,.$$

Let ϕ'' be the formula resulting by substituting in the formula ϕ' the parameter α_2 with $-\frac{a_2}{a_1}$. We have that in ϕ'' only the parameter α_1 appears with degree at most 4. Actually, ϕ' is a linear formula since S and ϕ are linear, and hence, the product between a polynomial of degree 3 with a polynomial of degree 1 returns a polynomial of degree 4.

Given a generic polynomial τ of degree at most 4 on the only parameter α_1, the set C of roots of τ is computable in polynomial time ([140]). Obviously $|C| \leq dg(S, \alpha_1, q) \leq 4$. Hence, $\tau \sim 0$ can be easily translated into a formula of the form $\alpha_1 \in I_1 \cup \ldots \cup I_m$, for some intervals of reals I_j. Therefore, since $\alpha_1 \in I_1 \wedge \alpha_1 \in I_2$ is equivalent to $\alpha_1 \in I_1 \cap I_2$, then ϕ'' can be translated into a formula of the form $\alpha_1 \in I'_1 \cup \ldots \cup I'_k$, for some intervals of reals I'_j.

Hence, given the instance u such that $u(\alpha_1)$ in $I'_1 \cup \ldots \cup I'_k$ and $u(\alpha_2)$ is the value $-\frac{a_2}{a_1}$ where α_1 is substituted with $u(\alpha_1)$, we have that $u \in Set(S, q, \phi)$ and $P_{S(u)}(q) = c$.

Example 5.9 *Consider the PPTS S of Figure 5.2. S is linear. We look for an instance $u \in Set(S, q_5, \alpha_1 + \alpha_2 \geq \frac{1}{2})$ such that $P_{S(u)}(q_5) = \frac{1}{3}$. We have that $dg(S, \alpha_1, q_5) = 3$ and $dg(S, \alpha_2, q_5) = 1$ Actually, $P_{S(u)}(q_4) = \frac{u(\tau_1)}{u(\tau_2)}$, where $\tau_1 = -\frac{13}{16}\alpha_1 + \frac{17}{8}(\alpha_1)^2 + 2\alpha_1\alpha_2 - \frac{1}{4}(\alpha_1)^2\alpha_2 - \frac{1}{4}(\alpha_1)^3$ and $\tau_2 = \alpha_1\left(3 - \frac{3}{4}\right)$.*

Hence we must find a value for α_1 such that $\frac{\tau_1}{\tau_2} = \frac{1}{3}$, which is equivalent to solving the equation

$$\alpha_2 = -\frac{\frac{5}{4} - \frac{9}{16}\alpha_1 + \frac{17}{8}(\alpha_1)^2 - \frac{1}{4}(\alpha_1)^3}{2\alpha_1 - \frac{1}{4}(\alpha_1)^2}.$$

Now we must substitute α_2 in the formula ϕ' that is equal to

$$\left(2\alpha_1 - \frac{1}{4}(\alpha_1)^2 \neq 0\right) \wedge \alpha_1 + \alpha_2 \geq \frac{1}{2} \wedge \left(\bigwedge_{e\in\delta} \lambda(e) \in [0,1]\right) \wedge \left(\bigwedge_{q'\in Q} \sum_{e\in Start(q')} \lambda(e) = 1\right) .$$

We have that in ϕ' the parameter α_1 has a degree of at most 4. As an example, $\alpha_1 + \alpha_2 \geq \frac{1}{2}$ becomes

$$\alpha_1 \left(2\alpha_1 - \frac{1}{4}(\alpha_1)^2\right) + \frac{5}{4} - \frac{9}{16}\alpha_1 + \frac{17}{8}(\alpha_1)^2 - \frac{1}{4}(\alpha_1)^3 \geq \frac{1}{2} \geq \frac{1}{2}\left(2\alpha_1 - \frac{1}{4}(\alpha_1)^2)\right) .$$

Hence, by solving this formula, we can find the intervals to which α_1 must belong.

As a consequence, by Lemma 5.1, we have the following theorem giving a syntactical characterization of PPTSs with two parameters satisfying the hypotheses of Theorem 5.5.

Theorem 5.6 *Let $S = (\{\alpha_1, \alpha_2\}, Q, q_0, \delta, \lambda)$ be a linear PPTS and q be a state in Q such that*

$$\left(\sum_{q'\in Q} max\{dg(\lambda(e), \alpha_1) \mid e \in AdmTr(q', q)\}\right) \in [1,3]$$

and

$$\left(\sum_{q'\in Q} max\{dg(\lambda(e), \alpha_2) \mid e \in AdmTr(q', q)\}\right) = 1 .$$

For any ϕ and $c \in [0,1]$; a well defined instance u, such that $u \in Set(S, q, \phi)$ and $P_{S(u)}(q) = c$, can be found in polynomial time w.r.t. the size of S.

Rational Domain for Parameters

If one is interested in parameters assuming only rational values (as it may often be the case), our results are extended to systems with any degree. Namely, if one considers only rational instances, the following result can be derived by using Ruffini's method.

Corollary 5.4 *Let $S = (\{\alpha\}, Q, q_0, \delta, \lambda)$ be a PPTS, q be a state in Q, ϕ be a general formula and $c \in [0,1]$. A well defined instance u, such that $u \in Set(S, q, \phi)$, $u(\alpha)$ is a rational number and $P_{S(u)}(q) = c$, can be found in polynomial time w.r.t. the size of S.*

Proof. Along the lines of the proof of Theorem 5.3 and by Ruffini's method that allows to find rational roots of a polynomial. Hence, the set C of rational roots of the polynomial $\tau_1 - c \cdot \tau_2$ is computable.

5.4.3 Finding the Maximum/Minimum Instance

In this section, we consider the case in which one wants either to maximize or to minimize the probability of reaching a certain state. This problem may have interesting applications in practice, as we shall show in the next section.

One Parameter

Theorem 5.7 (Maximizing/Miniminizing) *Let $S = (\{\alpha\}, Q, q_0, \delta, \lambda)$ be a PPTS and q be a state in Q such that $dg(S, \alpha, q) \leq 3$. For any general formula ϕ, an instance $u \in Set(S, q, \phi)$ is computable in polynomial time w.r.t. the size of S such that, for each $u' \in Set(S, q, \phi)$, it holds that $P_{S(u)}(q) \geq P_{S(u')}(q)$ (resp. $P_{S(u)}(q) \leq P_{S(u')}(q)$).*

Proof. The main steps of the proof are:

1. By following the proof of Theorem 5.3, we have to maximize (minimize) the function $\frac{\tau_1}{\tau_2}$.
2. To find the maximum (minimum) we compute the values of α that make null the derivative function of $\frac{\tau_1}{\tau_2}$.

By following the proof of Theorem 5.3 we have that $x_{q_0} = \frac{\tau_1}{\tau_2}$. Now, by mimicking the proof of Theorem 5.3, it is sufficient to maximize (minimize) the function $\frac{\tau_1}{\tau_2}$ in the space ϕ' that is equal to

$$\phi \wedge (\alpha_q \in [0,1]) \wedge (\bigwedge_{e \in \delta} \lambda(e) \in [0,1]) \wedge (\bigwedge_{q' \in Q} \sum_{e \in Start(q')} \lambda(e) = 1).$$

As done for Theorem 5.5, ϕ' can be translated into a formula of the form $\alpha \in I_1 \cup \ldots \cup I_m$, where I_j is an interval of real values.

The maximum of $\frac{\tau_1}{\tau_2}$ is when the derivative function $\frac{d}{d\alpha}\frac{\tau_1}{\tau_2} = 0$.

We have that $\tau_i = a_3^i\alpha^3 + a_2^i\alpha^2 + a_1^i\alpha + a_0^i$, for $i = 1, 2$. Hence, $\frac{d}{d\alpha}\frac{\tau_1}{\tau_2}$ is equal to

$$\frac{(a_3^1a_2^2 - a_2^1a_3^2)\alpha^4 + (2a_3^1a_1^2 - 2a_1^1a_3^2)\alpha^3 + (3a_3^1a_0^2 - 3a_0^1a_3^2 + a_2^1a_1^2 - a_1^1a_2^2)\alpha^2}{(\tau_2)^2} +$$

$$+ \frac{(2a_2^1a_0^2 - 2a_0^1a_2^2)\alpha + a_1^1a_0^2 - a_0^1a_1^2}{(\tau_2)^2}.$$

Therefore, the maximum value can be found by studying the function

$$(a_3^1a_2^2 - a_2^1a_3^2)\alpha^4 + (2a_3^1a_1^2 - 2a_1^1a_3^2)\alpha^3 + (3a_3^1a_0^2 - 3a_0^1a_3^2 + a_2^1a_1^2 - a_1^1a_2^2)\alpha^2 + (2a_2^1a_0^2 - 2a_0^1a_2^2)\alpha + a_1^1a_0^2 - a_0^1a_1^2$$

(which is a function of degree 4) in the space $\alpha \in I_1 \cup \ldots \cup I_m$, where I_j is an interval.

Since computing the terms τ_1, τ_2 takes a polynomial time w.r.t. the size of S, the problem of finding a maximal (minimal) solution is polynomial time w.r.t. the size of S.

As done for Theorem 5.3, we can give a syntactical characterization of the PPTSs satisfying the conditions of Theorem 5.7. Actually, from Lemma 5.1 and Theorem 5.7 we have the following theorem.

Theorem 5.8 *Let $S = (\{\alpha\}, Q, q_0, \delta, \lambda)$ be a PPTS and q be a state such that*

$$\left(\sum_{q' \in Q} max\{dg(\lambda(e), \alpha) | e \in AdmTr(q', q)\}\right) \leq 3.$$

For any general ϕ, an instance $u \in Set(S, q, \phi)$ such that, for each $u' \in Set(S, q, \phi)$, it holds that $P_{S(u)}(q) \geq P_{S(u')}(q)$ (resp. $P_{S(u)}(q) \leq P_{S(u')}(q)$), is computable in polynomial time w.r.t. the size of S.

Two Parameters

As for the problem of finding a solution, in this section we extend the result of Theorem 5.7 to two parameters. Again, to do that we must restrict to linear formulae and linear PPTS.

Theorem 5.9 (Maximizing/Miniminizing) *Let $S = (\{\alpha_1, \alpha_2\}, Q, q_0, \delta, \lambda)$ be a linear PPTS and q be a state in Q such that $dg(S, \alpha_1, q) = dg(S, \alpha_2, q) = 1$. For any linear formula ϕ, it is decidable in polynomial time w.r.t. the size of S to find an instance $u \in Set(S, q, \phi)$ such that, for each $u' \in Set(S, q, \phi)$, it holds that $P_{S(u)}(q) \geq P_{S(u')}(q)$ (resp. $P_{S(u)}(q) \leq P_{S(u')}(q)$).*

Proof. The main steps of the proof are:

1. By following the proof of Theorem 5.3 we have to maximize (minimize) the function $\frac{\tau_1}{\tau_2}$.

2. To find the maximum (minimum) we compute the values of α_1 and α_2 that make null the derivative function of $\frac{\tau_1}{\tau_2}$.

By following the proof of Theorem 5.3, we have that $x_{q_0} = \frac{\tau_1}{\tau_2}$. Now, by mimicking the proof of Theorem 5.3, it is sufficient to maximize (minimize) the function $\frac{\tau_1}{\tau_2}$ in the space ϕ' that is equal to

$$\phi \wedge (\alpha_q \in [0, 1]) \wedge (\bigwedge_{e \in \delta} \lambda(e) \in [0, 1]) \wedge (\bigwedge_{q' \in Q} \sum_{e \in Start(q')} \lambda(e) = 1).$$

We have that $\tau_i \equiv a_1^i \alpha_1 \alpha_2 + a_2^i \alpha_1 + a_3^i \alpha_2 + a_4^i$, for $i = 1, 2$. Hence,

$$\frac{d}{d\alpha_1} \frac{\tau_1}{\tau_2} = \frac{(a_1^1 a_3^2 - a_1^2 a_3^1)(\alpha_2)^2 + (a_1^1 a_4^2 + a_2^1 a_3^2 - a_1^2 a_4^1 + a_2^2 a_3^1)\alpha_2 + (a_2^1 a_4^2 - a_2^2 a_4^1)}{(\tau_2)^2}.$$

We note that $(a_1^1 a_3^2 - a_1^2 a_3^1)(\alpha_2)^2 + (a_1^1 a_4^2 + a_2^1 a_3^2 - a_1^2 a_4^1 + a_2^2 a_3^1)\alpha_2 + (a_2^1 a_4^2 - a_2^2 a_4^1)$ is a polynomial of degree 2 on the only parameter α_2. Similarly we can find $\frac{d}{d\alpha_2} \frac{\tau_1}{\tau_2}$. Thus, the maximum can be studied as done in Theorem 5.7.

Now, as done for Theorem 5.7, from Lemma 5.1 and Theorem 5.9 we have the following theorem giving a syntactical characterization of PPTSs satisfying the hypoteses of Theorem 5.9.

Theorem 5.10 *Let $S = (\{\alpha_1, \alpha_2\}, Q, q_0, \delta, \lambda)$ be a linear PPTS and q be a state in Q such that*

$$\left(\sum_{q' \in Q} max\{dg(\lambda(e), \alpha_1) \mid e \in AdmTr(q', q)\} \right) = \left(\sum_{q' \in Q} max\{dg(\lambda(e), \alpha_2) \mid e \in AdmTr(q', q)\} \right) = 1.$$

For any linear formula ϕ, it is decidable in polynomial time w.r.t. the size of S to find an instance $u \in Set(S, q, \phi)$ such that, for each $u' \in Set(S, q, \phi)$, it holds that $P_{S(u)}(q) \geq P_{S(u')}(q)$ (resp. $P_{S(u)}(q) \leq P_{S(u')}(q)$).

5.5 An Application: Probabilistic Non-Repudiation

In this section, as an application, we model and analyze a non-repudiation protocol that employs a probabilistic algorithm to achieve a fairness property. This protocol has been studied, from different points of view, also in [7, 94, 97].

5.5.1 A Probabilistic Non-Repudiation Protocol

We consider a protocol that guarantees a non-repudiation service with a certain probability without resorting to a trusted third party [118]. In particular, such a probabilistic protocol is fair up to a given tolerance ε decided by the originator. Assume that an authentication phase precedes the protocol. We denote by $Sign_E(M)$ the encryption of message M under the private key of the entity E and with $\{M\}_K$ the encryption of M under the key K. Finally, we use t to denote a time stamp. The protocol can be described as follows (with the notation $R \to O : Msg$ we denote a message Msg sent by R and received by O):

$$
\begin{array}{lrll}
1. & R \to O: & Sign_R(request, R, O, t) & \\
2. & O \to R: & Sign_O(\{M\}_K, O, R, t) & (= M_1) \\
3. & R \to O: & Sign_R(ack_1) & \\
4. & & & \\
\quad a._{1-p} & O \to R: & Sign_O(M_r, O, R, t) & (= M_i) \\
 & R \to O: & Sign_R(ack_i) & \\
 & & \text{goto step 4} & \\
\quad b._p & O \to R: & Sign_O(K, O, R, t) & (= M_n) \\
5. & R \to O: & Sign_R(ack_n) &
\end{array}
$$

The recipient R starts the protocol by sending a signed, timestamped request to the originator O. This sends to R the requested message M ciphered under the key K, and waits for the ack from R (ack_i represents the acknowledgment related to message M_i). At step 4 the originator makes a probabilistic choice according to p. At step 4a (taken with probability $1-p$) O sends to R a random message M_r (i.e. a dummy key), receives the ack and returns to step 4, while at step 4b (taken with probability p) O sends to R the key K necessary to decrypt the message $\{M\}_K$. Upon reception of the last ack (ack_n), related to the message containing the key K, the originator terminates the protocol correctly. We suppose that each message ack_i has the following semantics: "R acknowledges having received message M_i from O". This could be easily obtained, for instance, by assuming that each ack_i message contains an hash of message M_i.

Intuitively, the non-repudiation of origin is guaranteed by the messages M_1 and M_n (signed with the private key of O), while the non repudiation of receipt is given by the last message $Sign_R(ack_n)$. If the protocol terminates after the delivery of the last ack, both parties obtain their expected information, and the protocol is fair. If the protocol terminates before sending the message containing the key K, then neither the originator nor the recipient obtains any valuable information, thus preserving fairness. A strategy for a dishonest recipient consists in guessing the last message containing the key K, verifying whether a received message contains the needed key, and then blocking the transmission of the last ack. Therefore, for the success of the protocol, it is necessary that the ack messages are sent back immediately. The originator decides a deadline for the reception of each ack, after which, if the ack is not received, the protocol is stopped. Obviously, the cryptosystem must be adequately chosen, in such a way that the time needed to verify a key, by deciphering the message, is longer than the transmission time of an ack message. Anyway, as we will see in the next section, a malicious recipient can try to randomly guess the message containing the key K, and in this case the probability for the recipient of guessing the last message depends on the parameter p chosen by the originator.

5.5.2 Parametric Analysis of the Protocol

In this section we describe the protocol by using the model of PPTSs. In particular we use two parameters, p and q. On the one hand, we assume that the originator follows a Bernoulli distribution with parameter p to decide whether to send the real key or to send a dummy key (see step 4 of the protocol). On the other hand, we assume that the recipient follows a Bernoulli distribution with parameter q to decide whether to send the ack message or to try to compute M by employing the last received message. In Figure 5.3 we show a parametric Probabilistic Transition

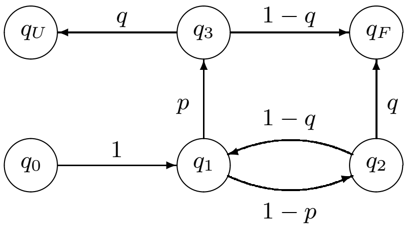

Figure 5.3: Parametric Representation of the Protocol.

System modelling the communication between the originator and the recipient according to the parameters p and q.

With the transition (q_0, q_1) we model the recipient starting a communication with the originator by sending a request, the originator sending the first ciphered message and the recipient acknowledging such a message. In state q_1 the originator sends, with probability $1-p$, a dummy key reaching state q_2 and, with probability p, sends the last message containing K and reaches state q_3. In state q_2 the recipient sends an ack to the originator with probability $1-q$ going back to state q_1, while with probability q the recipient uses the dummy key in order to decipher the first message, fails and the protocol is stopped. In this case, state q_F is reached. Intuitively, state q_F models a situation in which the protocol ends in a fair way (both participants receive their expected information or neither the originator nor the recipient obtains any valuable information). In state q_3 the recipient sends the last ack with probability $1-q$ and fairly terminates the protocol, and tries to decipher the first message with the last received key (in this case the correct key K) with probability q. In this case, without sending the last ack, the recipient breaks the fairness of the protocol (state q_U represents the situation in which the protocol ends in an unfair way).

Let us assume as S the PPTS of Figure 5.3 (where we omitted self-loops for states q_U and q_F).

If we are interested in the probability of reaching state q_F we can follow the system of linear equations of Corollary 5.1, and get $x_{q_0} = \frac{p+q-2pq}{p+q-pq}$.

Now, as an example, to find a well defined instance u such that $P_{S(u)}(q_F) = 0.9$ it is sufficient to apply Theorem 5.3. We have that $P_{S(u)}(q_F) = 0.9$ is equivalent to $\frac{1}{10}p + \frac{1}{10}q - \frac{11}{10}pq = 0$. Therefore, we have that $p = \frac{q}{11q-1}$ and $q \in \{0\} \cup (\frac{1}{11}, 1]$. We have a well defined instance for $u(p) = \frac{1}{4}$ and $u(q) = \frac{1}{7}$.

Now, we want to find a well defined instance u such that $\forall u' \in Set(S, q_F, true)$ $P_{S(u)}(q_F) \geq P_{S(u')}(q_F)$ and $P_{S(u)}(q_F) \in [0, 1]$ (namely, an instance that maximizes the probability of having a fair communication).

Now, we must find the maximum of the function $f(p, q) = \frac{p+q-2pq}{p+q-pq}$. We compute the derivative of f w.r.t. p and q. We have that $\frac{d}{dp}f(p, q) = \frac{-q^2}{(p+q-pq)^2}$ and $\frac{d}{dq}f(p, q) = \frac{-p^2}{(p+q-pq)^2}$. Hence, the function f is decreasing w.r.t. p for q (the point $(0, 0)$ is a point of contrary flexure), and therefore, the maximum in the space $[0, 1]$ is, indeed, 1 by choosing $q = 0$ or $p = 0$. Notice that the case $u(q) = 0$ models an attacker behaving correctly (he never tries to break the protocol), while the case $u(p) = 1$ models an originator that never sends the last message (hence the protocol cannot be broken).

We may suppose q to be a fixed constant and not a parameter. In this case, we want to find an instance for p (chosen by the originator) that maximizes the probability of reaching state q_F. In this manner the originator can choose the best value for p that minimizes the probability that the protocol ends in an unfair way. Let S' be the PPTS of Figure 5.3 with $q = \frac{1}{2}$ (namely, the attacker throws a coin to decide whether to decipher the key or not). We want to find a well defined instance u such that $\forall u' \in Set(S', q_F, true)$ $P_{S'(u)}(q_F) \geq P_{S'(u')}(q_F)$ (namely, an instance that maximizes the probability of having a fair communication).

For state q_F and with $q = \frac{1}{2}$, we get $x_{q_0} = \frac{1}{1+p}$. Now, we can find the value of p that maximizes the function $\frac{1}{1+p}$, by studying its derivative $\frac{d}{dp}\frac{1}{1+p} = \frac{-1}{(1+p)^2}$. Since such a function is decreasing

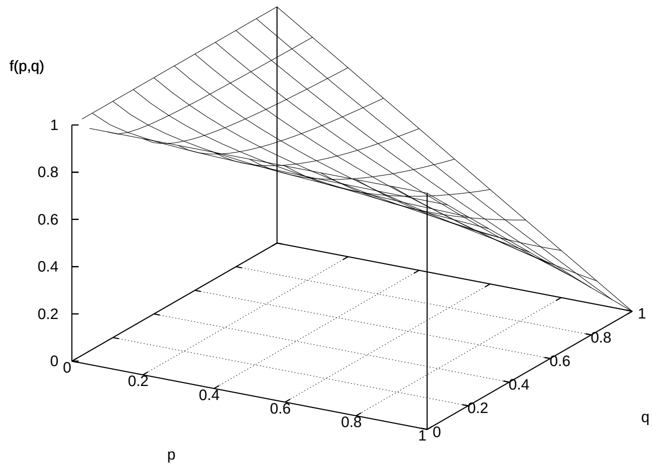

Figure 5.4: $f(p,q)$: probability of reaching q_F on the parameters p and q.

in $(-\infty,\infty)$ and $p \in [0,1]$, the maximum is for $u(p) = 0$ and, in this case, $P_{S'(u)}(q_F) = 1$. Again, if $p = 0$ the originator will never send the last message containing the correct key, fairness will not be broken, but the protocol will never terminate correctly. Differently, if the originator wants a probability of fair communication equal to 0.999, then it is sufficient to apply Theorem 5.3 which gives $\frac{1}{1+p} = 0.999$, and therefore $p = \frac{0.001}{0.999}$.

A full study of the function $f(p,q)$ is in Figure 5.4.

Viceversa, we may be interested in studying the value of q (chosen by the attacker) that maximizes the probability of breaking protocol fairness, namely the probability of reaching state q_U. In this case, we have $x_{q_0} = f'(p,q) = \frac{pq}{p+q-pq}$ and $\frac{df'}{dq} = \frac{p^2}{(p+q-pq)^2}$, which is decreasing w.r.t. q and, therefore, the maximum in the space $[0,1]$ is for $q = 1$. Thus, the maximum chance for the attacker of breaking protocol fairness is when choosing $u(q) = 1$, with $P_{S(u)}(q_U) = p$. This guarantees that protocol fairness (once the originator has chosen the parameter p) cannot be broken with probability greater than p, thus allowing the originator to chose the security threshold $1-p$ for the correct termination of the protocol. Moreover, the probability of a successful attack decreases by choosing a smaller p. In this case, however, the number of dummy messages sent by O, and hence the length of the protocol execution, will increase. Hence, the originator must choose a value for p that guarantees a good security level with the desired performance.

A full study of the function $f'(p,q)$ is in Figure 5.5.

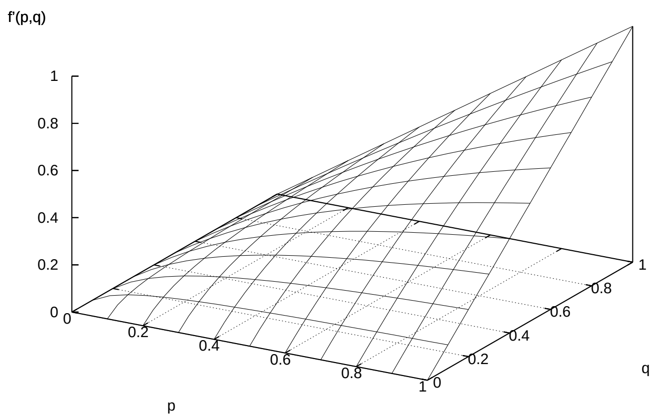

Figure 5.5: $f'(p, q)$: probability of reaching state q_U on the parameters p and q.

Chapter 6

Timed Automata with Data Structures

Systems of communicating agents can be described by automata composed in parallel and sharing synchronization channels. Transitions labeled with a complementing channel name can be taken at the same moment and data transmission is typically modeled by a synchronization, where global variables are updated ([33]).

Security protocols, like distributed programs in general, are sensitive to the passage of time; however, most methods for the formal analysis of security properties of protocols do not take time aspects into account (see, among them, [46, 1]). The role of time in the analysis of cryptographic protocols has only recently received some attention (see [55, 117, 33]).

Time aspects can influence the flow of messages during the execution of a protocol. For instance, if a message does not arrive in a certain time interval, retransmissions or other behaviour should be considered and, in this case, the protocol description should model these implementation details. Time information can also be used within a protocol in order to enrich the information contained in a message (for example by constructing *timestamps*). Finally, the timing of the message flow may be exploited by an adversary to violate the security of the protocol.

In [117] the authors propose a model of communicating automata tailored for describing and analyzing protocols with timing constraints and verifying their security. Each participant in the protocol is described by a state transition diagram where transitions are labeled by events which represent the sending of a structured message along a channel. The performance of a transition is conditioned by the trigger of a communication event and by the temporal constraints imposed by a delay and/or timeout. Communication is synchronous. Primitive messages (public/private keys, identities, nonces, etc.) can be composed by using cryptographic primitives (encryption, hashing, signature, etc.). An initial evidence function assigns each initial state the set of evidence that is known by each participant at the beginning. Such evidence can be augmented by receiving messages from other participants and is reset to the initial conditions every time an input state is reached. The semantics of these descriptions is given in terms of Timed Automata, on which properties can be verified.

In this chapter we define Systems of Data Management Timed Automata (SDMTAs). An SDMTA consists of a finite set of elementary messages, a finite set of functions to elaborate them and a finite set of Data Management Timed Automata (DMTAs). Each DMTA has a finite set of channel labels, a finite set of clocks, a finite set of states (one of them is the initial state), an initial condition and an initial knowledge, a finite set of transitions. Transitions from state to state of a DMTA represent either an internal move with the computation of a term (which enriches the knowledge of the automaton) or the input or the output of a term on a channel. The performance of a transition is conditioned by the fulfillment of a constraint. Two DMTAs of a system may perform a communication step modeling the communication of a term through a channel. Time elapsing is modeled by a time step of the automata of the system.

Our formalism extends the formalism of [117] to deal with a general class of distributed communicating systems with data structures. Note that the formalism of [117] assumes a bounded knowledge and has the power of regular languages, while our formalism assumes an unbound knowledge and, when endowed with a concept of recognized language, does accept languages which are not regular. Differently with respect to [117], we define a direct operational semantics of SDMTAs in terms of steps and runs, and we prove the decidability of the reachability. This allows proving properties expressible in terms of reached states.

We then introduce the model of Cryptographic SDMTAs (CSDMTAs) as a subclass of SDMTAs. We show how to model cryptographic protocols with CSDMTAs and give the formalization and the decidability proof of secrecy and authentication properties. As an application we model and analyze with CSDMTAs a version of the well-known Yahalom protocol adapted to take timeouts and retransmissions into account.

6.1 Basic notions

Let us assume a set of X positive real variables x called *clocks*. A *valuation* over a set of clocks is a mapping $v : X \to \mathbb{R}^{\geq 0}$ assigning real values to clocks. For a valuation v and a time value $t \in \mathbb{R}^{\geq 0}$, let $v + t$ denote the valuation such that $(v + t)(x) = v(x) + t$, for each clock $x \in X$.

Let $C = \{C_1, \ldots, C_m\}$ be a finite set, where C_i denotes a finite set of elementary messages. For an elementary message we mean non composed/manipulated message (i.e., names are elementary messages, lists of elementary messages are not).

Let us assume a set Υ of message variables μ that can assume values in $(\cup_{j=1}^{n} C_j) \cup \mathbb{N}$. Given a finite set of message variables, an instance I relates a message variable μ to a value in $(\cup_{j=1}^{n} C_j) \cup \mathbb{N}$. Namely, $I : \Upsilon \to (\cup_{j=1}^{n} C_j) \cup \mathbb{N}$.

Let $\Omega = \{f_1, \ldots, f_n\}$ denote a finite set of functions. Given a finite set of message variables Υ, the set of terms $\mathcal{T}(\Upsilon)$ is defined as:

$$\tau ::= c \mid w \mid \mu \mid f(\tau_1, \ldots, \tau_k)$$

where $c \in C_i$ for some i, $w \in \mathbb{N}$, $\mu \in \Upsilon$ is a message variable, f is a function in Ω with arity k.

We use C and Ω to represent a set of data structures and a set of functions to manipulate such structures. In general, C may be any set of data structures of different types and Ω may be any set of functions which represent operations on such structures. All the examples we shall use to explain the framework focus on the application of the model to the case of cryptographic security protocols. In such a context, the set C may contain sets of ground messages exchanged within the protocol (for example a set of plaintext messages, a set of agent names, a set of keys...), and the set Ω may contain the basic cryptographic primitives, as message pairing, encryption, nonce generation, hashing, etc.

Example 6.1 *Consider $C = \{A, M, K\}$, where $A = \{a, b, \ldots\}$ is a set of agent names, $M = \{m_1, m_2, \ldots\}$ is a set of basic messages (i.e. a set of plaintext messages represented by bitstrings of a fixed length) and $K = \{k_1, k_2, \ldots\}$ is a set of keys. Given a finite set of message variables Υ, we assume $\Omega = \{Pair, Enc, Nonce\}$, with domains $\mathcal{T}(\Upsilon) \times \mathcal{T}(\Upsilon)$, $\mathcal{T}(\Upsilon) \times K$ and $(A \cup \Upsilon) \times (\Upsilon \cup \mathbb{N})$, respectively. $Pair(\tau_1, \tau_2)$ denotes the concatenation of the terms τ_1 and τ_2, $Enc(m_1, k_1)$ denotes the encryption of message m_1 with the key k_1, and $Nonce(a, 100)$ denotes a nonce of the agent a with value 100. $Nonce(\mu_1, \mu_2)$, where $\mu_1, \mu_2 \in \Upsilon$ are variables, may be instantiated by $Nonce(a, w)$ for some $a \in A$ and $w \in \mathbb{N}$. $Enc(\mu, k)$, where μ is a variable can be instantiated by $Enc(m, k)$ for some $m \in M$, by $Enc(a, k)$ for some $a \in A$, by $Enc(k, k)$ for some $k \in K$ or by $Enc(w, k)$ for some $w \in \mathbb{N}$. We remark that μ_1, μ_2, μ cannot be complex terms, i.e. $Enc(Nonce(a, 1), k)$ or $Enc(Enc(m', k'), k)$ are not instances of $Enc(\mu, k)$.*

With $Var(\tau)$ we denote the message variables appearing in the term τ. For example, we have that $Var(Enc(Nonce(\mu_1, 100), \mu_2)) = \{\mu_1, \mu_2\}$.

We say that two terms τ and τ' in $\mathcal{T}(\Upsilon)$ are *reducible* (written $\tau \simeq \tau'$) if they have the same structure, namely $\tau \simeq \tau'$ if there exist variables $\mu_1, \ldots, \mu_n$ and $\overline{\mu}_1, \ldots, \overline{\mu}_n \in \Upsilon$ such that $\tau = \tau'[\overline{\mu}_1/\mu_1] \ldots [\overline{\mu}_n/\mu_n]$.

Finally, with $\mathcal{K}$ we denote a knowledge. A knowledge $\mathcal{K} \subset \mathcal{T}(\Upsilon)$ is a finite set of terms τ such that $Var(\tau) = \emptyset$.

Given a finite set of clocks X and a finite set of message variables Υ, we define the set $\Phi(X, \Upsilon)$ of *formulae* as follows:

$$\begin{array}{rl} \phi ::= & true \mid \tau \in \mathcal{K} \mid \tau = \tau' \mid \mu \in \mathbb{N} \mid \\ & x \sim c \mid x \sim y \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \end{array}$$

where $\phi, \phi_1, \phi_2 \in \Phi(X, \Upsilon)$, $\tau, \tau' \in \mathcal{T}(\Upsilon)$, $\mu \in \Upsilon$, $x, y \in X$, $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{Q}$.

We will write $\tau \neq \tau'$ for $\neg(\tau = \tau')$, $\tau \notin \mathcal{K}$ for $\neg(\tau \in \mathcal{K})$, $\mu \notin \mathbb{N}$ for $\neg(\mu \in \mathbb{N})$, and $\tau \in \{\tau_1, \ldots, \tau_k\}$ for $\tau = \tau_1 \vee \ldots \vee \tau = \tau_k$. As an example, with $\tau \in C_i$, for some i, we denote the formula $\bigvee_{m \in C_i} \tau = m$.

Given a term τ and an instance I, we define the instantiation of τ as $I(\tau) = \tau'$ where τ' is the term resulting after replacing each μ syntactically occurring in τ with $I(\mu)$.

Let $\phi \in \Phi(X, \Upsilon)$, I be an instance, v a valuation of clocks and $\overline{\mathcal{K}}$ a knowledge; we say that I, v and $\overline{\mathcal{K}}$ *satisfy* ϕ, written $I, v, \overline{\mathcal{K}} \models \phi$, in the following cases:

$$\begin{array}{lll} I, v, \overline{\mathcal{K}} \models true & & \\ I, v, \overline{\mathcal{K}} \models \tau \in \mathcal{K} & \text{iff} & I(\tau) \in \overline{\mathcal{K}} \\ I, v, \overline{\mathcal{K}} \models \tau = \tau' & \text{iff} & I(\tau) = I(\tau') \\ I, v, \overline{\mathcal{K}} \models \mu \in \mathbb{N} & \text{iff} & I(\mu) \in \mathbb{N} \\ I, v, \overline{\mathcal{K}} \models x \sim c & \text{iff} & v(x) \sim c \\ I, v, \overline{\mathcal{K}} \models x \sim y & \text{iff} & v(x) \sim v(y) \\ I, v, \overline{\mathcal{K}} \models \neg\phi_1 & \text{iff} & I, v, \overline{\mathcal{K}} \not\models \phi_1 \\ I, v, \overline{\mathcal{K}} \models \phi_1 \vee \phi_2 & \text{iff} & \text{either } I, v, \overline{\mathcal{K}} \models \phi_1 \text{ or } I, v, \overline{\mathcal{K}} \models \phi_2 \\ I, v, \overline{\mathcal{K}} \models \phi_1 \wedge \phi_2 & \text{iff} & \text{both } I, v, \overline{\mathcal{K}} \models \phi_1 \text{ and } I, v, \overline{\mathcal{K}} \models \phi_2. \end{array}$$

Example 6.2 *The formula $\mu_1 \neq \mu_2 \wedge \mu_1 \in \mathcal{K} \wedge \mu_2 \in \mathcal{K}$ says that in the knowledge $\mathcal{K}$ there are at least two elementary messages. Formula $\mu \in K \wedge \mu \in \mathcal{K}$, where $K \in C$ denotes the set of keys, means that in $\mathcal{K}$ there is at least one key.*

6.2 Data Management Timed Automata

Given a finite set of clocks X and a finite set of message variables Υ, with X' and Υ' we denote new sets of clocks and message variables such that $x' \in X'$ and $\mu' \in \Upsilon'$ iff $x \in X$ and $\mu \in \Upsilon$, respectively.

A System of Data Management Timed Automata (SDMTA) is a tuple $\mathcal{A} = (C, \Omega, A_1, \ldots, A_m)$, where:

- $C = \{C_1, \ldots, C_k\}$ is a set of elementary messages;
- $\Omega = \{f_1, \ldots, f_n\}$ is a set of functions;
- $A_1, \ldots, A_m$ are Data Management Timed Automata (DMTAs).

A DMTA is a tuple $A = (\Sigma, X, \Upsilon, Q, q_0, \phi_0, \mathcal{K}_0, \delta)$, where:

- Σ is a finite set of channel labels;
- X is a finite set of clocks;

- Υ is a finite set of message variables;
- Q is a finite set of states with $q_0 \in Q$ initial state;
- $\phi_0 \in \Phi(X, \Upsilon)$ is the initial condition;
- $\mathcal{K}_0$ is the initial knowledge of A.
- δ is a finite set of transitions. Each transition is a tuple (q, α, ϕ, q'), where $q, q' \in Q$ are the source and the target states respectively, for $a \in \Sigma$, $\alpha \in \{\epsilon(\tau), a?(\tau), a!(\tau)\}$ represents, respectively, the internal move producing the term τ, which enriches the knowledge of $\mathcal{A}$, or the input or the output of the term $\tau \in \mathcal{T}(\Upsilon)$ on channel a, ϕ is a formula in $\Phi(X \cup X', \Upsilon \cup \Upsilon')$. Variables in $X \cup \Upsilon$ and in $X' \cup \Upsilon'$ represent the value of variables before and after the firing of the transition, respectively.

An example of transition is $(q_0, a!(\mu), x < 5 \wedge \mu \in \mathcal{K} \wedge x' = x \wedge \mu' \in \mathbb{N}, q_1)$, stating that from the initial state q_0 the DMTA may reach state q_1 and output a message μ when this is in $\mathcal{K}$ and $x < 5$. The condition $x' = x$ means that the transition does not change the clock x, and the condition $\mu' \in \mathbb{N}$ means that the variable μ nondeterministically assumes a natural value after the transition.

Given an SDMTA $\mathcal{A} = (C, \Omega, A_1, \ldots, A_m)$, states of $\mathcal{A}$ are represented by tuples $(q_1, \ldots, q_m)$, where $q_i \in A_i$.

6.2.1 Semantics

Given two valuations v_1, v_2 over X, with $v_1 \oplus v_2$ we denote the valuation on $X \cup X'$ such that for any $x \in X$, $(v_1 \oplus v_2)(x) = v_1(x)$ and $(v_1 \oplus v_2)(x') = v_2(x)$. Given two instances I_1, I_2 over Υ, with $I_1 \oplus I_2$ we denote the instance over $\Upsilon \cup \Upsilon'$ such that for any $\mu \in \Upsilon$, $(I_1 \oplus I_2)(\mu) = I_1(\mu)$ and $(I_1 \oplus I_2)(\mu') = I_2(\mu)$.

A configuration of an SDMTA $\mathcal{A} = (C, \Omega, A_1, \ldots, A_m)$ with $A_i = (\Sigma^i, X^i, \Upsilon^i, Q^i, q_0^i, \phi_0^i, \mathcal{K}_0^i, \delta^i)$, is a tuple $(s_1, \ldots, s_m)$ such that $s_i = (q, v, I, \mathcal{K})$ is a configuration of the DMTA A_i with $q \in Q^i$ a state of A_i, v a valuation over X^i, I an instance over Υ^i and $\mathcal{K}$ a knowledge.

Given two configurations $s = (s_1, \ldots, s_m)$ and $s' = (s'_1, \ldots, s'_m)$ such that $s_i = (q_i, v_i, I_i, \mathcal{K}_i)$ and $s'_i = (q'_i, v'_i, I'_i, \mathcal{K}'_i)$, we have that:

- there is a *ϵ-transition step* from s to s' (denoted $s \rightarrow_\tau s'$) if there exist an index i and $e = (q_i, \epsilon(\tau_1), \phi, q'_i) \in \delta^i$ such that $I_i(\tau_1) = \tau$, $(I_i \oplus I'_i), (v_i \oplus v'_i), \mathcal{K}_i \models \phi$, $\mathcal{K}'_i = \mathcal{K}_i \cup \{\tau\}$ and, for all $j \neq i$, $s'_j = s_j$;
- there is a *communication transition step* from s to s' with the term τ (denoted $s \rightarrow_{a(\tau)} s'$) if there exist two different indexes i and j, $(q_i, a!(\tau_1), \phi_1, q'_i) \in \delta^i$ and $(q_j, a?(\tau_2), \phi_2, q'_j) \in \delta^j$ such that:
  - $I_i(\tau_1) = I_j(\tau_2) = \tau$;
  - $(I_i \oplus I'_i), (v_i \oplus v'_i), \mathcal{K}_i \models \phi_1$ and $(I_j \oplus I'_j), (v_j \oplus v'_j), \mathcal{K}_j \models \phi_2$,
  - $\mathcal{K}'_i = \mathcal{K}_i \cup \{\tau\}$ and $\mathcal{K}'_j = \mathcal{K}_j \cup \{\tau\}$;
  - for all $k \notin \{i, j\}$, it holds that $s'_k = s_k$;
- there is a *time step* from s to s' through time $t \in \mathbb{R}^{>0}$, written $s \rightarrow_t s'$, if, for any i, $q'_i = q_i$, $v'_i = (v_i + t)$, $I'_i = I_i$, and $\mathcal{K}'_i = \mathcal{K}_i$.

With the ϵ-transition step we model the internal data manipulation executed by the DMTA without communication. For example, given a configuration $(q, v, I, \mathcal{K})$, transition $(q, \epsilon(\mu), \phi, q')$, where $\phi = Enc(\mu, k) \in \mathcal{K} \wedge k \in \mathcal{K}$, states that if $(I \oplus I'), (v \oplus v'), \mathcal{K} \models \phi$ the DMTA may decipher a ciphertext $Enc(\mu, k)$ contained in its knowledge if also the key k is known, and then enrich the knowledge with the instantiation $I(\mu)$.

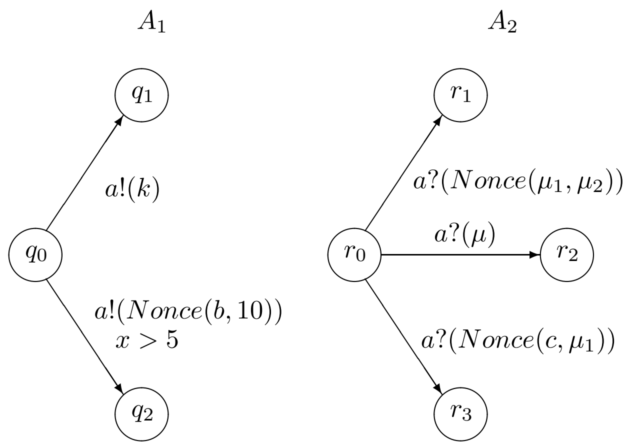

Figure 6.1: A System of Cryptographic Timed Automata.

A communication transition step $\rightarrow_{a(\tau)}$ models the communication of the term τ through the channel a. There is a synchronization between two DMTAs, and they both change their configuration by following the formula in the transition and by augmenting their knowledge. If variables appear in the output and input terms (τ_1 and τ_2, respectively), they should be *reducible* for transmission. In particular, we require that $I_i(\tau_1) = I_j(\tau_2)$ (also see Example 6.3). The other DMTAs of the system non involved in the communication remain in their original configuration.

Finally, a time step models time elapsing. When time elapses, we reasonably assume that each DMTA A_i in the system performs a time step by changing its valuation v_i.

Example 6.3 *Consider the DMTAs A_1 and A_2 in Figure 6.1, where q_0 and r_0 are the initial states of A_1 and A_2, respectively. If $\mathcal{K}_0$ is the initial knowledge of A_1 and we assume that $k \in \mathcal{K}_0$, the only communications that may happen between A_1 and A_2 through synchronization steps of matching terms are either $a(Nonce(b, 10))$ (when x has value greater than 5) or $a(k)$. In both cases, the knowledge of A_2 is augmented with the term received by A_1.*

Given a DMTA $A = (\Sigma, X, \Upsilon, Q, q_0, \phi_0, \mathcal{K}_0, \delta)$, a configuration $s = (q_0, v, I, \mathcal{K}_0)$ of A is *initial* if $I, v, \mathcal{K}_0 \models \phi_0$. Given an SDMTA $\mathcal{A} = (C, \Omega, A_1, \ldots, A_m)$, we say that the configuration $s = (s_1, \ldots, s_m)$ of $\mathcal{A}$ is initial iff s_i is an initial configuration of the DMTA A_i for all $i \in [1, m]$.

A *run* of an SDMTA $\mathcal{A}$ is a finite sequence of steps $\sigma = s_0 \rightarrow_{\alpha_1} s_1 \rightarrow_{\alpha_2} \ldots \rightarrow_{\alpha_l} s_l$ where s_0 is an initial configuration of $\mathcal{A}$, s_j is a configuration of $\mathcal{A}$ and $\alpha_j \in \{\tau, a(\tau)\} \cup \mathbb{R}^{>0}$ for each $j \in [1, l]$.

A state $\bar{q} = (q_1, \ldots, q_m)$ of an SDMTA $\mathcal{A}$ is reachable iff there is a run $\sigma = (s_0^1, \ldots, s_0^m) \rightarrow_{\alpha_1} \ldots \rightarrow_{\alpha_l} (s_l^1, \ldots, s_l^m)$ of $\mathcal{A}$ such that, for some j, $s_j^i = (q_i, v, I, \mathcal{K})$ for all $i \in [1, m]$. A state q of a DMTA A_i in the SDMTA $\mathcal{A}$ is reachable iff there is a run $\sigma = (s_0^1, \ldots, s_0^m) \rightarrow_{\alpha_1} \ldots \rightarrow_{\alpha_l} (s_l^1, \ldots, s_l^m)$ of $\mathcal{A}$ such that, for some j, $s_j^i = (q, v, I, \mathcal{K})$.

6.2.2 Expressiveness

In this section we discuss the expressive power of SDMTAs. To this purpose we introduce a notion of accepted run with respect to a set of states and a notion of language. We suppose that $\mathcal{A}$ has associated a set $F \subseteq \bigcup_{i \in [1,m]} Q_i$ of *final states.*

Given a run $r = s_0 \rightarrow_{\alpha_1} s_1 \rightarrow_{\alpha_2} \ldots \rightarrow_{\alpha_l} s_l$, with *word*$(r)$ we denote the string $a_1 \cdot \ldots \cdot a_l$ such that, for any i, either $\alpha_i = a_i$ and $\alpha_i \in \mathbb{R}^{\geq 0}$ or there exists τ such that $\alpha_i = a_i(\tau)$. We say that r *is accepted by* $\mathcal{A}$ if the states appearing in s_l are contained in F. With $\mathcal{L}(\mathcal{A})$ we denote the set $\{word(r) \,|\, r \text{ is accepted by } \mathcal{A}\}$.

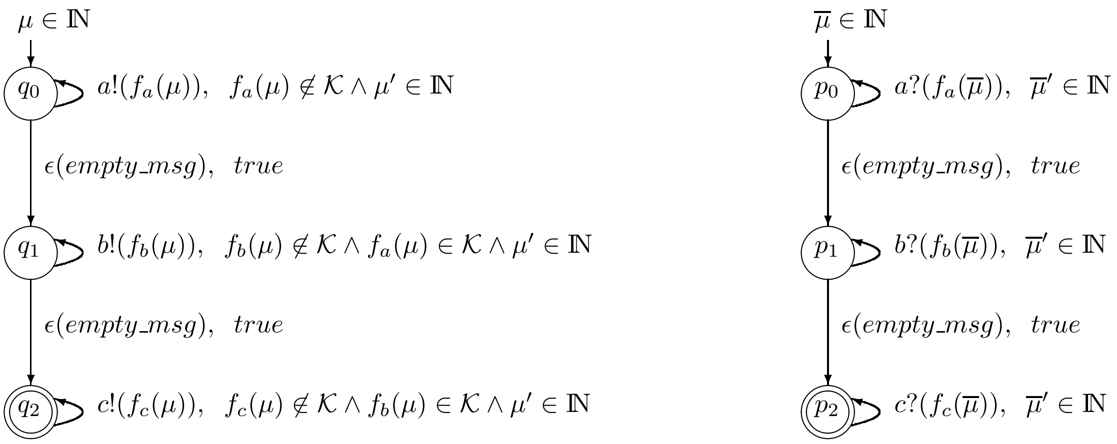

Figure 6.2: SDMTA for $\{a^k b^h c^m | k \geq h \geq m\}$.

Given a string $z = a_1 \cdot \ldots \cdot a_m$, with $untime(z)$ we denote the string $a'_1 \cdot \ldots \cdot a'_m$ such that $a'_i = \epsilon$ if $a_i \in \mathbb{R}^{\geq 0}$ and $a'_i = a_i$ otherwise.

Given a language L, with $Untime(L)$ we denote the set $\{untime(z) \,|\, z \in L\}$.

Proposition 6.1 *There exists an SDMTA $\mathcal{A}$ such that $Untime(\mathcal{L}(\mathcal{A})) = \{a^k b^h c^m \,|\, k \geq h \geq m\}$.*

Proof. In figure 6.2 we give $\mathcal{A}$ such that $Untime(\mathcal{L}(\mathcal{A})) = \{a^k b^h c^m \,|\, k \geq h \geq m\}$.

The second component just receives the messages sent by the first one.

In state q_0 of the first component, $\mathcal{A}$ generates a^k. After the performance of the ϵ transition from q_0 to q_1 the knowledge is equal to $\{f_a(c_1), \ldots, f_a(c_k)\}$ for some natural number c_i such that $c_i \neq c_j$ if $i \neq j$.

In state q_1 of the first component, $\mathcal{A}$ generates b^h. We have that $h \leq k$ since we require that $f_a(\mu) \in \mathcal{K}$. After the performance of the ϵ transition from q_1 to q_2 the knowledge contains $\{f_b(c'_1), \ldots, f_b(c'_h)\}$ for some natural number c'_i such that $c'_i \neq c'_j$ if $i \neq j$.

Finally, in state q_2 of the first component, $\mathcal{A}$ generates c^m. We have that $m \leq h$ since we require that $f_b(\mu) \in \mathcal{K}$. □

Now, language $\{a^k b^h c^m \,|\, k \geq h \geq m\}$ is not regular, while in [9] it is proven that, if L is a language recognized by a Timed Automaton, then $Untime(L)$ is a regular language. Therefore, by Proposition 6.1, and since a Timed Automaton can be easily translated into a SDMTA, we have the following result.

Corollary 6.1 *SMDTAs are more expressive than Timed Automata.*

As a consequence, also the model presented in [117], equivalent to the class of Timed Automata, is less expressive than SDMTAs.

6.2.3 Decidability of Reachability

We recall the definitions of clock equivalence for DMTAs. Clock equivalence (see also Section 1.3.2) is a finite index equivalence relation permitting to group sets of valuations and to have decidability results.

Let A be a DMTA; with C_A we denote the greatest constant that appears in A.

Let us consider the equivalence relation $\approx$ over clock valuations containing precisely the pairs (v, v') such that:

- for each clock x, either $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, or both $v(x)$ and $v'(x)$ are greater than C_A, with C_A the largest integer appearing in clock constraints over x;
- for each pair of clocks x and y with $v(x) \leq C_A$ and $v(y) \leq C_A$ it holds that $fract(v(x)) \leq fract(v(y))$ iff $fract(v'(x)) \leq fract(v'(y))$ (where $fract(\cdot)$ is the fractional part);
- for each clock x with $v(x) \leq C_A$, $fract(v(x)) = 0$ iff $fract(v'(x)) = 0$.

Let $[v]$ denote the equivalence class $\{v' \mid v \approx v'\}$. The set of equivalence classes given by $\{[v] \mid v \text{ is a valuation}\}$ is finite, and with V we denote its cardinality.

Known properties of the equivalence classes are summarized in the following theorem [9].

Theorem 6.1 *Given a DMTA A, let $c < C_A$ and v and v' be two valuations; $v \in [v']$ implies that $v \models x \sim c$ iff $v' \models x \sim c$. Moreover, given a class $[v]$, the set $\{[v + t] \mid t \in \mathbb{R}^{\geq 0}\}$ is computable.*

For simplicity, we prove that we can always translate an SDMTA into an SDMTA composed by only one DMTA.

Proposition 6.2 *Given an SDMTA $\mathcal{A}$ and a state $\bar{q} = (q_1, \ldots, q_m)$ of $\mathcal{A}$, an SDMTA $\mathcal{A}'$ composed by only one DMTA with a state $\hat{q}$ can be constructed such that $\bar{q}$ is reachable by $\mathcal{A}$ iff $\hat{q}$ is reachable by $\mathcal{A}'$.*

Proof. It is sufficient to consider the cartesian product of the sequential components of $\mathcal{A}$ and introduce two functions *pair* and *triple* in Ω. Therefore, given $\mathcal{A} = (C, \Omega, A_1, \ldots, A_m)$, where $A_i = (\Sigma^i, X^i, \Upsilon^i, Q^i, q_0^i, \phi_0^i, \mathcal{K}_0^i, \delta^i)$ for $i = 1, \ldots, m$, we construct $\mathcal{A}' = (C, \Omega', A)$, where $\Omega' = \Omega \cup \{pair, triple\}$ and $A = (\Sigma, X, \Upsilon, Q, q_0, \phi_0, \mathcal{K}_0, \delta)$. The set $\Sigma = \emptyset$ does not contain any channel label, and the sets $X = \bigcup_i X^i$, $\Upsilon = \bigcup_i \Upsilon^i$ and $\mathcal{K}_0 = \bigcup_i \mathcal{K}_0^i$ are given by the union of the respective sets of each component A_i. The set of states $Q = Q^1 \times \ldots \times Q^m$ is given by the cartesian product of the set of states of each A^i with $q_0 = (q_0^1, \ldots, q_0^m)$ the initial state. The initial condition of A is $\phi_0 = \bigwedge_i \phi_0^i$, and the set of transitions δ is defined as follows.

Terms of the form $pair(i, \tau)$ are used to represent the fact that τ is in the knowledge of the i^{th} component thanks to an ϵ transition step. Terms of the form $triple(i, j, \tau)$ are used to represent the fact that τ is in the knowledge of the i^{th} and j^{th} components thanks to a communication between i and j.

Given a formula ϕ, with ϕ^i we denote the formula ϕ where each formula of the form $\tau \in \mathcal{K}$ is replaced with $pair(i, \tau) \in \mathcal{K} \vee \bigvee_{j \in [1,m]} (triple(i, j, \tau) \in \mathcal{K} \vee triple(j, i, \tau) \in \mathcal{K})$.

In $\mathcal{A}'$ we introduce a transition

$$((q_1, \ldots, q, \ldots q_m), \epsilon(pair(i, \tau)), \phi^i, (q_1, \ldots, q', \ldots q_m))$$

if there exists a transition $(q, \epsilon(\tau), \phi, q')$ of the i^{th} component of $\mathcal{A}$.

In $\mathcal{A}'$ we introduce a transition

$$((q_1, \ldots, q, \ldots, q'', \ldots q_m), \epsilon(triple(i, j, \tau)), \phi^i \wedge (\phi')^j \wedge \bar{\phi}, (q_1, \ldots, q', \ldots, q''', \ldots, q_m))$$

if there exist two transitions (q, α_1, ϕ, q') and $(q'', \alpha_2, \phi', q''')$ of the i^{th} and j^{th} components, respectively, such that $\{\alpha_1, \alpha_2\} = \{a!(\tau), a?(\tau')\}$ and $\bar{\phi}$ expresses the set of instances I and I' such that $I(\tau) = I'(\tau')$. $\square$

Hence, from now on we suppose to have SDMTAs composed by only one sequential DMTA performing only ϵ transitions.

With $Form(\mathcal{A})$ we denote the set of formulas ϕ appearing in the transitions and initial conditions of $\mathcal{A}$.

A formula ϕ *is simple for a set of message variables* Υ if and only if for each τ appearing in ϕ and for each instance I such that $I \models \phi$, $|\{\mu \,|\, \mu \in Var(\tau) \wedge I(\mu) \in \mathbb{N}\}| \leq 1$. Intuitively, a formula is simple if for each term appearing in the formula, at most one of its variables may assume natural values.

With $Term(\mathcal{A})$ we denote the set of terms τ appearing in the transitions of $\mathcal{A}$.

Given a term τ, with $Nat(\tau)$ we denote the set of natural numbers appearing in τ. If $\mathcal{K}$ is a knowledge, with $Nat(\mathcal{K})$ we denote the set $\bigcup_{\tau \in \mathcal{K}} Nat(\tau)$. Moreover, given an SDMTA $\mathcal{A}$, with $Nat(\mathcal{A})$ we denote the set $\bigcup_{\tau \in Term(\mathcal{A})} Nat(\tau)$. As an example, if the SDMTA $\mathcal{A}$ has the transition $(q, a!(Nonce(\mu, 10)), Nonce(A, 11) \in \mathcal{K}, q')$, then $10, 11 \in Nat(\mathcal{A})$.

A term τ *is simple for a set of message variables* Υ if it is equal to $f(\mu_1, \ldots, \mu_n)$ with $\mu_i \in \Upsilon$. As an example, $f(\mu_1, \mu_2)$ and $f(\mu)$ are simple, while $f(\mu_1, f(\mu_2, \mu))$ and $f(1, \mu_2)$ are not.

An SDMTA $\mathcal{A} = (C, \Omega, (\Sigma, X, \Upsilon, Q, q_0, \phi_0, \mathcal{K}_0, \delta))$ *is simple* if τ and ϕ are simple for Υ for all $\tau \in Term(\mathcal{A})$ and $\phi \in Form(\mathcal{A})$.

With $\Psi(\Upsilon)$ we denote the set of formulae of the form $\bigwedge_{\mu} \mu \in \mathbb{N} \wedge \bigwedge_{\overline{\mu} \in \Upsilon} \mu \sim_{\mu,\overline{\mu}} \overline{\mu}$ with $\sim_{\mu,\overline{\mu}} \in \{=, \neq\}$. As an example, $\psi = (\mu_1, \mu_2, \mu_3 \in \mathbb{N} \wedge \mu_1 = \mu_2 = \mu_3)$ is in $\Psi(\{\mu_1, \mu_2, \mu_3\})$.

Now, we show that a simple SDMTA can always be transformed into a standard form that preserves reachability of states.

Definition 6.1 *A simple SDMTA* $\mathcal{A} = (C, \Omega, (\Sigma, X, \Upsilon, Q, q_0, \phi_0, \mathcal{K}_0, \delta))$ *is in* natural standard form *iff:*

- $C = \emptyset$;
- $\phi_0 = \bigwedge_{x \in X} x = 0 \wedge \bigwedge_{\mu \in \Upsilon} \mu \in \mathbb{N} \wedge \bigwedge_{\overline{\mu} \in \Upsilon} \mu = \overline{\mu}$;
- *for any* $(q, \epsilon(\tau), \phi, q') \in \delta$ τ *is simple for* Υ *and* ϕ *is of the form* $\phi_1 \wedge \phi_2 \wedge \phi_3$ *where* ϕ_1 *is equal to* $\bigwedge_{i=1}^{n} \tau_i \in \mathcal{K} \wedge \bigwedge_{j=1}^{m} \tau'_j \notin \mathcal{K}$, *for some simple terms* $\tau_1, \ldots, \tau_n, \tau'_1, \ldots, \tau'_m$ *on* $\Upsilon \cup \Upsilon'$, $\phi_2 \in \Psi(\Upsilon \cup \Upsilon')$ *and* ϕ_3 *is a formula on clocks* $X \cup X'$.

For an SDMTA in *natural standard form*, message variables can assume only natural values ($C = \emptyset$). The initial condition of an SDMTA in natural standard form guarantees that clocks are set to 0 at the beginning and all message variables assume the same natural value. Finally, the condition on the formulas and on the terms appearing in the transitions guarantees that an SDMTA in natural standard form only operates on simple terms, and that each transition expresses the relation between message variables. Thus, every term appearing in an SDMTA in natural standard form is either a constant, or it can be represented with $f(\mu)$ for some $f \in \Omega$ and $\mu \in \Upsilon$, with $I(\mu) \in \mathbb{N}$ for all I.

Proposition 6.3 *Given a simple SDMTA* $\mathcal{A} = (C, \Omega, (\Sigma, X, \Upsilon, Q, q_0, \phi_0, \mathcal{K}_0, \delta))$ *we can construct an SDMTA* $\mathcal{A}' = (C', \Omega', (\Sigma, X, \Upsilon, Q', \overline{q}_0, \phi'_0, \emptyset, \delta'))$ *in natural standard form, with* $Q \subseteq Q'$, *such that for any* $q \in Q$, q *is reachable by* $\mathcal{A}$ *iff* q *is reachable by* $\mathcal{A}'$.

Proof. We consider $C' = \emptyset$ and $\phi'_0 = \bigwedge_{x \in X} x = 0 \wedge \bigwedge_{\mu \in \Upsilon} \mu \in \mathbb{N} \wedge \bigwedge_{\overline{\mu} \in \Upsilon} \mu = \overline{\mu}$ (as for Definition 6.1). We also consider $Q' = Q \cup \{\overline{q}_0, \ldots, \overline{q}_{|\mathcal{K}_0|}\}$. The set of transition δ' is obtained from the set of transitions δ as follows.

If $\mathcal{K}_0 = \{\tau_1, \ldots, \tau_n\}$, transitions $(\overline{q}_n, \epsilon(), \phi_0[\mu'/\mu]_{\mu \in \Upsilon}[x'/x]_{x \in X}, q_0)$ and $(\overline{q}_i, \epsilon(\tau_{i+1}), \bigwedge_{x \in X} x = 0, \overline{q}_{i+1})$ are added for all $i \in [0, n-1]$. These transitions initialize the values of the clocks, of the message variables and of the knowledge to the initial values expressed by ϕ_0 and $\mathcal{K}_0$.

Now, we delete the natural numbers in $Nat(\mathcal{A})$. If $Nat(\mathcal{A}) = \{n_1, \ldots, n_k\}$, then we consider $c_1, \ldots, c_k$ new constants and we substitute each n_i with c_i. Moreover, we replace each $\mu = n_i$ with $\mu = c_i$, and, $\mu \in \mathbb{N}$ with $\mu \in \mathbb{N} \vee \mu \in \{c_1, \ldots, c_k\}$, for any $\mu \in \Upsilon \cup \Upsilon'$. Finally, since the SDMTA is simple, for all terms in $Term(\mathcal{A})$, all variables occurring in a term are replaced with a unique variable assuming only natural values. The same is done for all formulas in $Form(\mathcal{A})$. This can be easily done since for simple SDMTAs there is at most one variable per formula assuming natural values.

Let T be the set of terms appearing in the SDMTA after these operations. We can partition T in $T_1, \ldots, T_h$ such that, for any $\tau, \tau' \in T$, it holds that $\tau \simeq \tau'$ iff $\tau, \tau' \in T_i$ for some i.

We then take $\Omega' = \{f_1, \ldots, f_h\}$, and, for any $\tau \in T_i$, we replace τ with the simple term $f_i(\mu_i)$, where μ_i is the unique variable appearing in τ_i.

Now, we recursively replace each transition $(q, \epsilon(\tau), \phi_1 \vee \phi_2, q')$ with two transitions $(q, \epsilon(\tau), \phi_1, q')$ and $(q, \epsilon(\tau), \phi_2, q')$.

It is easy to see that such a construction of $\mathcal{A}'$ guarantees that the states in Q reached by $\mathcal{A}$ are the same that are reached by $\mathcal{A}'$. $\square$

From now on, we suppose that SDMTAs are simple SDMTAs in natural standard form.

For checking reachability for Timed Automata, in [9] these are reduced to finite state machines. We note that an unbound knowledge cannot be simulated with a finite state machine. Hence we prove the decidability of reachability of SDMTAs by reducing the problem to the reachability problem for Vector Addition Systems (VASs).

We recall now the model of VASs. Given two vectors $w, w' \in \mathbb{Z}^n$, with $\pi_i(w)$ we denote the i^{th} component of w and with $w + w'$ we denote the vector w'' such that $\pi_i(w'') = \pi_i(w) + \pi_i(w')$, for any $i \in [1, n]$.

Definition 6.2 *A* Vector Addition System *(VAS)* of dimension n *is a tuple* $\mathcal{S} = (Q, \delta)$ *such that* Q *is a finite set of states and* $\delta \subseteq Q \times \mathbb{Z}^n \times Q$*. A* configuration *is a pair* (q, w) *with* $q \in Q$ *and* $w \in \mathbb{N}^n$*. There exists a step from configuration* (q, w) *to configuration* (q', w') *(denoted with* $(q, w) \rightarrow (q', w')$*) if* $(q, w'', q') \in \delta$ *such that* $w' = w + w''$*.*

The following theorem is proved in [80].

Theorem 6.2 *Given two configurations* (q, w) *and* (q', w') *of a VAS* $\mathcal{S}$*, the problem of checking whether there exists a sequence of steps* $(q, w) \rightarrow \ldots \rightarrow (q', w')$ *is decidable and EXP-SPACE hard.*

We now prove the reduction result from VASs to SDMTAs.

Theorem 6.3 *Given an SDMTA* $\mathcal{A}$ *and a state* $\overline{q}$ *of* $\mathcal{A}$*, there exists a VAS* $\mathcal{S}$ *and two states* q' *and* q'' *of* $\mathcal{S}$ *such that* $\mathcal{A}$ *reaches* $\overline{q}$ *iff there exists a sequence of steps* $(q', (0, \ldots, 0)) \rightarrow \ldots \rightarrow (q'', (0, \ldots, 0))$ *of* $\mathcal{S}$*.*

Proof. First of all, we consider an extension of VASs. If $\mathcal{S} = (Q, \delta)$ we assume $\delta \subseteq Q \times 2^{[1,n]} \times \mathbb{Z}^n \times Q$. There exists a step from the configuration (q, w) to the configuration (q', w') if $(q, In, w'', q') \in \delta$ such that $w' = w + w''$ and $\pi_i(w) > 0$ for any $i \in In$.

It is obvious that, given an *extended*–VAS we can construct an equivalent VAS by replacing any transition (q, In, w, q') with two transitions $(q, w', \hat{q}), (\hat{q}, w - w', q')$ where $\hat{q}$ is a new state and $\pi_i(w') = -1$ if $i \in In$ and $\pi_i(w') = 0$ otherwise. Note that, by definition, the values of w, for any configuration (q, w), could not be smaller than 0. Therefore, the two transitions introduced above are enough to check whether the values corresponding to indexes in In are greater than 0.

Let $\Omega = \{f_1, \ldots, f_k\}$, Q, Υ and X be the set of function symbols, the set of states, the set of message variables and the set of clocks of $\mathcal{A}$, respectively.

With $\overline{\mathcal{T}(\Upsilon)}$ we denote the set of terms of the form $f(A)$ such that $f \in \Omega$ and $\emptyset \neq A \subseteq \Upsilon$. Given a set $\tilde{\Upsilon} \subseteq \Upsilon$ and $T \subseteq \overline{\mathcal{T}(\Upsilon)}$, let $T_{\tilde{\Upsilon}}$ denote the set $\{f(A \cap \tilde{\Upsilon}) \mid f(A) \in T\}$.

A set $T \subseteq \overline{\mathcal{T}(\Upsilon)}$ is *coherent* if, for any $f(A), f'(A') \in T$, it holds that either $A = A'$ or $A \cap A' = \emptyset$.

The idea is that $f(A)$ represents the instance I and the term $f(c)$ such that $c = I(\mu)$, if $\mu \in A$. Hence, if $A = \emptyset$, then c is different from $I(\mu)$, for any μ. Therefore, $\mu_1, \mu_2 \in A$, for some i, iff $I(\mu_1) = I(\mu_2)$. Intuitively, a set T is coherent if it defines an instance I.

A set $T \subseteq \overline{\mathcal{T}(\Upsilon)}$ is *connected* if it is coherent and, for any $\tau, \tau' \in T$, $Var(\tau) = Var(\tau') \neq \emptyset$. The idea is that a set is connected if the values appearing in two terms are related.

Given a connected set $T \subseteq \overline{\mathcal{T}(\Upsilon)}$, with $[T]$ we denote the set of connected sets T' in T such that, given the set of variables A appearing in T, there exist sets of variables A' such that $T' = T[A'/A]$. Intuitively, $[T]$ expresses the knowledge of T abstracting from instances.

A state of the VAS $\mathcal{S}$ is constructed as a tuple $(q, [v], (A_1, T_1), \ldots, (A_p, T_p))$ such that $q \in Q$, v is a valuation on X, and $\emptyset \neq A_i \subseteq \Upsilon$ and T_i is a connected set in $\mathcal{T}(A_i)$, for any i, and $\{A_1, \ldots, A_p\}$ is a partition of Υ.

The dimension of $\mathcal{S}$, denoted with $U = 2^{|\overline{\mathcal{T}(\Upsilon)}|}$, is given by the number of subsets of $\overline{\mathcal{T}(\Upsilon)}$. Let $g : 2^{\overline{\mathcal{T}(\Upsilon)}} \to [1, U]$ be a surjective function assigning a coordinate of the vector to each subset of $\overline{\mathcal{T}(\Upsilon)}$.

The configuration $((q, [v], (A_1, T_1), \ldots, (A_p, T_p)), w)$ of $\mathcal{S}$ expresses the configuration $(q', v', I, \mathcal{K})$ of $\mathcal{A}$ such that $q = q'$, $v' \in [v]$, $(A_1, T_1), \ldots, (A_p, T_p)$ define the instance I and $w \in \mathbb{N}^U$ represents the knowledge $\mathcal{K}$ that we can refer when we have the instance I. Note that the subsets of $\overline{\mathcal{T}(\Upsilon)}$ referred by g and w contain terms with variables, those variables should be instantiated according to I in order to get the knowledge $\mathcal{K}$.

The set of transitions of $\mathcal{S}$ is constructed as follows.

To model time steps of $\mathcal{A}$, we add in $\mathcal{S}$ transitions of the form $(\overline{q}, In, w, \overline{q}')$ where $In = \emptyset$, $w = (0, \ldots, 0)$, $\overline{q} = (q, [v], (A_1, T_1), \ldots, (A_p, T_p))$, for some q, $[v]$, A_i and T_i, and $\overline{q}' = (q, [v + t], (A_1, T_1), \ldots, (A_p, T_p))$ for some $t \in \mathbb{R}^{\geq 0}$.

Moreover, there is a transition in $\mathcal{S}$ from the state $(q, [v], (A_1, T_1), \ldots, (A_p, T_p))$ to the state $(q', [v'], (A'_1, T'_1), \ldots, (A'_r, T'_r))$, with label (In, w) if there exists a transition $(q, \epsilon(\tau), \phi_1 \wedge \phi_2 \wedge \phi_3, q')$ of $\mathcal{A}$ with $\phi_1 = \bigwedge_{i=1}^{n} \tau_i \in \mathcal{K} \wedge \bigwedge_{j=1}^{m} \tau'_j \notin \mathcal{K}$, $\phi_2 \in \Psi(\Upsilon \cup \Upsilon')$ and ϕ_3 a formula on clocks $X \cup X'$, such that:

- $v \oplus v'$ satisfies ϕ_3;
- there exists a coherent set $T \subseteq \overline{\mathcal{T}(\Upsilon \cup \Upsilon')}$ and a term $\tau' = f(\overline{A})$ such that $\tau = f(\mu)$, $\overline{A}_i$ is the set in which μ_i appears in T and it holds that:
  - $\bigcup_{i=1}^{p} T_i = T_\Upsilon$ and $\bigcup_{i=1}^{r} T'_i = (T_{\Upsilon'})\,[\mu/\mu']_{\mu' \in \Upsilon'}$;
  - for any i, if $\tau_i = \overline{f}(\mu)$, then there exists $\overline{f}(\overline{A}) \in T$ with $\mu \in \overline{A}$;
  - for any j, if $\tau'_j = \overline{f}(\mu)$, then for any $\overline{f}(\overline{A}) \in T$ it holds that $\mu \notin \overline{A}$;
  - for any μ_1, μ_2, it holds that $\phi_2 \Rightarrow (\mu_1 = \mu_2)$ iff either $\mu_1, \mu_2 \in A_j$ for some j, or, for some i and j, there exists $\overline{f}(\overline{A}) \in T_i$ such that $\mu_1, \mu_2 \in \overline{A}$;
  - for any μ_1, μ_2, it holds that $\phi_2 \Rightarrow (\mu_1 \neq \mu_2)$ iff either $\mu_1 \in A_h$ and $\mu_2 \in A_k$ with $h \neq k$, or, there exists no $\overline{f}(\overline{A}) \in T_i$ such that $\mu_1, \mu_2 \in \overline{A}$ for any i;
- $In = \{g(T_i) \mid i = 1, \ldots, p\}$;
- if $\tau' \in \bigcup_{i=1}^{m} T_i$, then $w = (0, \ldots, 0)$ and otherwise, if $H = \{i \mid Var(T_i) \cap Var(\tau) \neq \emptyset\}$, then
$$\pi_j(w) = \begin{cases} -1 & \text{if } g^{-1}(j) \in [T_i] \text{ with } T_i \neq \emptyset \text{ and } i \in H \\ 1 & \text{if } g^{-1}(j) \in [\{\tau'\} \cup \bigcup_{i \in H} T_i] \\ 0 & \text{otherwise} \end{cases}.$$

The last condition models the fact that, if τ is in the knowledge (namely, $\tau' \in \bigcup_{i=1}^{m} T_i$), then the knowledge does not change (namely, $w = (0, \ldots, 0)$). Otherwise, the insertion of τ creates new relations among the terms in the knowledge. Actually, the natural numbers appearing in τ

can appear in other terms in the knowledge. The terms that become related when τ is inserted are those with indices in H. Hence, the case $\pi_j(w) = -1$ means that the set of connected term T_i represented by coordinate j (namely, $g^{-1}(j) \in [T_i]$) that has relations with τ still cannot be activated by a certain instance. Actually, it does not express the new relations introduced by τ. The case $\pi_j(w) = 1$ means that the set of terms $\{\tau'\} \cup \bigcup_{i \in H} T_i$, represented by coordinate j and expressing the new relations, can be activated by a certain instance.

Hence, we have a transition in $\mathcal{S}$ for any step of $\mathcal{A}$. Each transition updates the values of the vector in the configuration by following the changes of the knowledge due to the step of $\mathcal{A}$. Note that, if a transition of $\mathcal{S}$ models a time step of $\mathcal{A}$, the knowledge is left unchanged.

Finally, we add a new state q_T to $\mathcal{S}$ and transitions with label $(In, (0, \ldots, 0))$ and target q_T from each state $((\overline{q}, [v], (A_1, T_1), \ldots, (A_m, T_m))$ with $In = \{g(T_i) \mid i = 1, \ldots, m\}$. Moreover, we add $e_1, \ldots, e_U$ transitions such that, for each i, $e_i = (q_T, \emptyset, w_i, q_T)$, where $\pi_j(w_i) = -1$ if $j = i$ and $\pi_j(w_i) = 0$ otherwise. The state q_T represents the state $\overline{q}$ of $\mathcal{A}$, and these new transitions have the purpose to set the vector to $(0, \ldots, 0)$.

Since the construction of $\mathcal{S}$ reflects precisely the behavior of each single step of $\mathcal{A}$, $\mathcal{A}$ reaches $\overline{q}$ iff there exists a sequence of steps in $\mathcal{S}$ from the configuration $((q_0, [\bigwedge_{x \in X} x = 0], \bigwedge_{\mu, \overline{\mu} \in \Upsilon} \mu = \overline{\mu}, \emptyset), (0, \ldots, 0))$ to the configuration $(q_T, (0, \ldots, 0))$. This is easily proven by induction on the length of the sequence $((q_0, [\bigwedge_{x \in X} x = 0], \bigwedge_{\mu, \overline{\mu} \in \Upsilon} \mu = \overline{\mu}, \emptyset), (0, \ldots, 0)) \to \ldots \to (q_T, (0, \ldots, 0))$, by considering, at the inductive step, the different cases that arise when constructing the set of transitions of $\mathcal{S}$. $\square$

Hence, we have the following result.

Corollary 6.2 *Given an SDMTA $\mathcal{A}$, it is decidable whether a state q is reachable.*

Example 6.4 *Consider a simple SDMTA composed by a DMTA $\mathcal{A}$ with initial condition $\phi_0 = (x = 0 \wedge \mu_1 = \mu_2)$ and a transition $e = (q, \epsilon(f(\mu_1)), x < 1 \wedge f(\mu_1) \notin \mathcal{K} \wedge \mu_1' \neq \mu_1 = \mu_2 = \mu_2', q)$.*

The set of equivalence classes is $\{x = 0, 0 < x < 1, x = 1, x > 1\}$.

As an example, from configuration $((q, x = 0, (\{\mu_1, \mu_2\}, \emptyset)), (0, \ldots, 0))$ we have a step with label $(0, \ldots, 0)$ to the configuration $((q, v, (\{\mu_1, \mu_2\}, \emptyset)), (0, \ldots, 0))$ with $v \in \{x = 0, 0 < x < 1, x = 1, x > 1\}$ representing a possible time step. The pair $(\{\mu_1, \mu_2\}, \emptyset)$ means that $\mu_1 = \mu_2$ and $f(\mu_1), f(\mu_2) \notin \mathcal{K}$.

By transition e we can reach the configuration $((q, x = 0, (\{\mu_1\}, \emptyset)(\{\mu_2\}, \{f(\mu_2)\})), w)$ such that w is one in position $g(f(\{\mu_1\})), g(f(\{\mu_2\})), g(f(\{\mu_1, \mu_2\}))$, and is zero otherwise. Actually, after the step we have that $\mu_1 \neq \mu_2$, $f(\mu_2) \in \mathcal{K}$ and in the future it is possible to create three different kind of instances such that:

1. *$f(\mu_1) \in \mathcal{K} \wedge f(\mu_2) \notin \mathcal{K}$ is true (since $\pi_{g(f(\{\mu_1\}))}(w) = 1$);*
2. *$f(\mu_1) \notin \mathcal{K} \wedge f(\mu_2) \in \mathcal{K}$ is true (since $\pi_{g(f(\{\mu_2\}))}(w) = 1$);*
3. *$f(\mu_1) \in \mathcal{K} \wedge \mu_1 = \mu_2$ is true (since $\pi_{g(f(\{\mu_1, \mu_2\}))}(w) = 1$).*

Let us consider now the transition $e' = (q, \epsilon(h(\mu_1)), x = 0 \wedge f(\mu_1) \in \mathcal{K} \wedge \mu_1' = \mu_1 = \mu_2 = \mu_2', q)$ which inserts the term $h(\mu_1)$ in the knowledge.

Let us consider also the state $((q, x = 0, (\{\mu_1, \mu_2\}, f(\{\mu_1, \mu_2\}))), w)$, where w is 1 in position $f(\{\mu_1, \mu_2\})$ and 0 otherwise. This state expresses that $\mu_1 = \mu_2$ and $f(\mu_1) = f(\mu_2)$ is in the knowledge (note that $h(\mu_1)$ is not in the knowledge).

The transition e' decrements the coordinate of $f(\{\mu_1, \mu_2\})$ (that becomes 0) and increments that of $\{h(\{\mu_1, \mu_2\}), f(\{\mu_1, \mu_2\})\}$ (that becomes 1). Actually, before the step we can create an instance for which $f(\mu_1)$ is in the knowledge and $h(\mu_1)$ is not, but after the step this is not possible.

This is necessary to deal with conditions of the form $f(\mu_1) \in \mathcal{K} \wedge g(\mu_1) \notin \mathcal{K}$.

6.3 Modeling Cryptographic Protocols with SDMTAs

Security protocols are sensitive to the passage of time since time aspects can influence the flow of messages during the execution of a protocol.

On the one hand, timeouts may be integrated within the protocol and retransmissions or other behaviour can be considered. In this case, the description of the protocol should model these implementation details. On the other hand, time information can be used within a protocol in order to enrich the information contained in a message. This kind of application may be very useful, for example, when generating *timestamps*. Finally, one should also be aware that the timing of the message flow may be exploited by an adversary to violate the security of the protocol.

In this section we define an instantiation of SDMTAs to the case of cryptographic protocols (CSDMTAs). We formalize intruder's capabilities and we a give the definitions of secrecy and authentication properties for cryptographic protocols within this framework. Then we study the Yahalom protocol by modeling each principal of the protocol and the intruder with DMTAs. The execution of the protocol is modeled by the resulting CSDMTA.

Definition 6.3 *A Cryptographic SDMTA (CSDMTA) is an SDMTA $\mathcal{AC} = (C, \Omega, A_1, \ldots, A_m, I)$, where C and Ω are defined as in Example 6.1, DMTAs A_i model the principals in the protocol, and I is the DMTA modeling the intruder.*

In the following, we give a formalization of a classical Dolev–Yao style intruder [46].

Definition 6.4 *Given a CSDMTA $\mathcal{AC} = (C, \Omega, A_1, \ldots, A_m, I)$, where each DMTA A_i, modelling a principal in the cryptographic protocol, is in the form $A_i = (\Sigma^i, X^i, \Upsilon^i, Q^i, q_0^i, \phi_0^i, \mathcal{K}_0^i, \delta^i)$, we say that $I = (\Sigma, X, \Upsilon, Q, q_0, \phi_0, \mathcal{K}_0, \delta)$ is the intruder for $\mathcal{AC}$ iff:*

- $\Sigma = \cup_{i=1}^m \Sigma^i$;
- $X = \{x\}$;
- $Q = \{q_0\}$;
- *$\mathcal{K}_0$ is given by the initial public information necessary for the execution of the protocol;*
- $(q_0, \epsilon(), x' = 0, q_0) \in \delta'$;
- *for all $i \in [1, m]$, if either $(q, a!(\tau), \phi, q') \in \delta^i$ or $(q, a?(\tau), \phi, q') \in \delta^i$, then $(q_0, a?(\tau'), x' = 0, q_0)$ and $(q_0, a!(\tau'), \tau' \in \mathcal{K} \wedge x' = 0, q_0)$ are in δ, where $\tau \simeq \tau'$;*
- *for all $i \in [1, m]$, if $Pair(\tau_1, \tau_2)$ is a subformula of τ with $(q, a!(\tau), \phi, q') \in \delta^i$, then both $(q_0, \epsilon(\tau_1'), Pair(\tau_1', \tau_2') \in \mathcal{K} \wedge x \geq t_u \wedge x' = 0, q_0)$ and $(q_0, \epsilon(\tau_2'), Pair(\tau_1', \tau_2') \in \mathcal{K} \wedge x \geq t_u \wedge x' = 0, q_0)$ are in δ, where $\tau_1 \simeq \tau_1'$, $\tau_2 \simeq \tau_2'$ and t_u is a constant;*
- *for all $i \in [1, m]$, if $Pair(\tau_1, \tau_2)$ is a subformula of τ with $(q, a?(\tau), \phi, q') \in \delta^i$, then $(q_0, \epsilon(Pair(\tau_1', \tau_2')), \tau_1' \in \mathcal{K} \wedge \tau_2' \in \mathcal{K} \wedge x \geq t_p \wedge x' = 0, q_0) \in \delta$, where $\tau_1 \simeq \tau_1' \wedge \tau_2 \simeq \tau_2'$ and t_p is a constant;*
- *for all $i \in [1, m]$, if $Enc(\tau_1, \tau_2)$ is a subformula of τ with $(q, a!(\tau), \phi, q') \in \delta^i$, then $(q_0, \epsilon(\tau_1'), Enc(\tau_1', \tau_2') \in \mathcal{K} \wedge \tau_2' \in \mathcal{K} \wedge x \geq t_d \wedge x' = 0, q_0) \in \delta$, where $\tau_1 \simeq \tau_1'$, $\tau_2 \simeq \tau_2'$ and t_d is a constant;*
- *for all $i \in [1, m]$, if $Enc(\tau_1, \tau_2)$ is a subformula of τ with $(q, a?(\tau), \phi, q') \in \delta^i$, then $(q_0, \epsilon(Enc(\tau_1', \tau_2')), \tau_1' \in \mathcal{K} \wedge \tau_2' \in \mathcal{K} \wedge x \geq t_e \wedge x' = 0, q_0) \in \delta$, where $\tau_1 \simeq \tau_1'$, $\tau_2 \simeq \tau_2'$ and t_e is a constant;*

With such a definition of a single state intruder, we assume that principals of the protocol use only public channels, therefore the intruder may intercept each message exchanged within the network and spread any information. This is modeled by taking Σ as the set containing all channel names appearing in the automata A_i and by adding an input transition to the intruder for any output transitions of any A_i. Viceversa, an output transition is added to the intruder for any input transition of any A_i. Moreover, we give the intruder classical Dolev-Yao capabilities such as extraction of a term from a pair, pairing of two terms, and encryption and decryption of terms, assuming that the key is known. Note that, according to some given algorithms, this operations may require some amount of time to be performed. Therefore, for each of those transitions, we require that the value of the clock x is equal or greater than the constants t_u, t_p, t_d and t_e representing, respectively, the time needed for extracting a term from a pair, for pairing two terms, for deciphering an encrypted term and for encrypting a term. Finally, with the empty ϵ-transition the intruder may nondeterministically instantiate variables by modifying its actual instance. Note that by performing such a transition no messages are added to the knowledge.

6.3.1 Security Properties

Many security properties may be defined in order to analyze cryptographic protocols (among them, *secrecy*, *authentication*, *integrity*, *fairness*, *anonymity*, *non-repudiation*, etc.).

In this chapter we focus on the definition of secrecy and authentication properties for our framework. Intuitively, a term is secret to a principal if it never appears within its knowledge (see [1, 53]), while a principal A truly authenticates to a principal B if whenever B thinks to communicate with A, B is really communicating with A.

A Secrecy Property

For specifying and verifying the secrecy property, we require that a term τ, that should be kept secret to a set of principals P, does never appear within the knowledge of a principal in the set P.

Definition 6.5 *Given a CSDMTA $\mathcal{AC} = (C, \Omega, A_1, \ldots, A_m, I)$, a term τ with $Var(\tau) = \emptyset$ and a set of principals $P \subseteq \{A_1, \ldots, A_m, I\}$, we say that τ is* P-secret *for any principal in P iff there exists no run $\sigma = s^0 \to_{\alpha_1} s^1 \to_{\alpha_2} \ldots \to_{\alpha_l} s^l$ of $\mathcal{AC}$, with $s^i = (s_{A_1}, \ldots, s_{A_m}, s_I)$ for some $i \in [0, l]$ and $s_j = (q, I, v, \mathcal{K})$ for some $j \in P$ such that $\tau \in \mathcal{K}$.*

Proposition 6.4 *Given a CSDMTA $\mathcal{AC} = (C, \Omega, A_1, \ldots, A_m, I)$, it is decidable whether a term τ with $Var(\tau) = \emptyset$ is* P-secret *for a set of principals $P \subseteq \{A_1, \ldots, A_m, I\}$.*

Proof. We build the CSDMTA $\mathcal{AC}' = (C, \Omega, A'_1, \ldots, A'_m, I')$ such that for each DMTA $A \notin P$, $A' = A$ and for each $A = (\Sigma, X, \Upsilon, Q, q_0, \phi_0, \mathcal{K}_0, \delta) \in P$, $A' = (\Sigma, X, \Upsilon, Q \cup q_s, q_0, \phi_0, \mathcal{K}_0, \delta \cup \delta_s)$, where $q_s \notin Q$ and $\delta_s = \delta \cup \{(q, \epsilon(\tau), \tau \in \mathcal{K}, q_s) \,|\, q \in Q\}$. Now, for each DMTA in P, we have a special transition reaching the special state q_s when the term to be kept secret appears in its knowledge. Therefore, we can say that the term τ is P-secret for $\mathcal{AC}$ if no q_s can be reached in the CSDMTA $\mathcal{AC}'$. Thus, by the decidability of state reachability given in Corollary 6.2, also P-secrecy is decidable. □

An Authentication Property

In the literature several authentication flaws are defined (see, for example, [36, 1, 102]). The authentication failure that we will define in our framework is inspired by Lowe [102]. For specifying and verifying the authentication property, we require that whenever the principal B ends a communication protocol with the principal A, then A had previously started the communication protocol.

In order to formally define the authentication property for the model of CSDMTAs, we may assume that, given a CSDMTA $\mathcal{AC} = (C, \Omega, A_1, \ldots, A_m, I)$ and two principals A_i, A_j, whenever A_i starts a communication protocol with A_j, it reaches the state $q_{S_{ij}}$, and that whenever A_j ends the protocol with A_i, it reaches the state $q_{E_{ij}}$. This property can be defined very easily within our model by requiring that for each run of a CSDMTA, whenever the state $q_{E_{ij}}$ is reached, then the state $q_{S_{ij}}$ was previously crossed. Intuitively, this means that any closing event $q_{E_{ij}}$ is preceded by a starting event $q_{S_{ij}}$. In fact, if A_j reaches the state $q_{E_{ij}}$ without A_i reaching state $q_{S_{ij}}$, then it means that another principal, simulating of being A_i, started and performed a complete session of the communication protocol with A_j.

Definition 6.6 *Given a CSDMTA $\mathcal{AC} = (C, \Omega, A_1, \ldots, A_m, I)$, and two principals A_i, A_j, we say that the principal A_i* truly authenticates *to principal A_j, if there exists no run $\sigma = s^0 \rightarrow_{\alpha_1} s^1 \rightarrow_{\alpha_2} \ldots \rightarrow_{\alpha_l} s^l$ of $\mathcal{AC}$, with configurations $s^k = (s^k_{A_1}, \ldots, s^k_{A_m}, s^k_I)$ for $k \in [0, l]$, such that $s^p_{A_j} = (q_{E_{ij}}, I^p, v^p, \mathcal{K}^p)$ and $s^r_{A_j} = s^0_{A_j}$ or $s^r_{A_j} = (q_{E_{ij}}, I^r, v^r, \mathcal{K}^r)$ for some p, r with $r < p$ and $s^q_{A_i} \neq (q_{S_{ij}}, I^q, v^q, \mathcal{K}^q)$ for all $q \in [r, p]$.*

Proposition 6.5 *Given a CSDMTA $\mathcal{AC} = (C, \Omega, A_1, \ldots, A_m, I)$, it is decidable whether a principal A_i truly authenticates to principal A_j.*

Proof. We build the CSDMTA $\mathcal{AC}' = (C, \Omega, A_1, \ldots, A'_i, \ldots, A'_j, \ldots, A_m, O, I)$. On the one hand, given $A_i = (\Sigma, X, \Upsilon, Q, q_0, \phi_0, \mathcal{K}_0, \delta)$, we have $A'_i = (\Sigma, X, \Upsilon, Q \cup q_{run}, q_0, \phi_0, \mathcal{K}_0, \delta')$, where $q_{run} \notin Q$ and δ' is obtained by replacing each transition $(q_{S_{ij}}, \alpha, \phi, q) \in \delta$ with the two transitions $(q_{S_{ij}}, run!(), \bigwedge_{x \in X} x' = x \wedge \bigwedge_{\mu \in \Upsilon} \mu' = \mu, q_{run})$ and $(q_{run}, \alpha, \phi, q)$. On the other hand, given $A_j = (\Sigma, X, \Upsilon, Q, q_0, \phi_0, \mathcal{K}_0, \delta)$, we have $A'_j = (\Sigma, X, \Upsilon, Q \cup q_{com}, q_0, \phi_0, \mathcal{K}_0, \delta')$, where $q_{com} \notin Q$ and δ' is obtained by replacing each transition $(q_{E_{ij}}, \alpha, \phi, q) \in \delta$ with the two transitions $(q_{E_{ij}}, commit!(), \bigwedge_{x \in X} x' = x \wedge \bigwedge_{\mu \in \Upsilon} \mu' = \mu, q_{com})$ and $(q_{com}, \alpha, \phi, q)$. We assume that *run* and *commit* are special unused labels. Therefore, every time the DMTA A_i starts a run of the communication protocol it performs a transition with label *run*, while every time the DMTA A_j commits the ending of the communication it performs a transition with label *commit*.

Now, the new principal O has the task to observe the *run* and the *commit* transitions performed by A_i and A_j. Formally, $O = (\{com, run\}, \emptyset, \emptyset, \{o_0, o_1, o_{AutF}\}, o_0, true, \emptyset, \delta_0)$ where δ_o is composed by the four transitions $(o_0, run?(), true, o_1)$, $(o_0, commit?(), true, o_{AutF})$, $(o_1, run?(), true, o_1)$ and $(o_1, commit?(), true, o_0)$.

Hence, if a *commit* transition is performed without being preceded by a *run* transition (thus violating the authentication property), the observer O will reach state o_{AutF}.

Therefore, we can say that, given the protocol modeled by $\mathcal{AC}$, the principal A_i truly authenticates to the principal A_j if the state o_{AutF} can not be reached in the CSDMTA $\mathcal{AC}'$. Thus, by the decidability of state reachability given in Corollary 6.2, also the authentication property is decidable. □

Now, we can use the framework of CSDMTAs to model and analyze a real cryptographic protocol in a timed setting.

6.3.2 The Yahalom Protocol

The Yahalom protocol [26] is designed for the distribution of a fresh symmetric key shared between two users. The protocol resorts to a trusted server, and each user is assumed to share a symmetric key with the server. Here we consider a strengthened version of the Yahalom protocol proposed by Paulson in [120]. In the standard protocol notation, we denote with $A \rightarrow B : Msg$ a message Msg sent by A and received by B, with $\{Msg\}_K$ we denote the encryption of the message Msg under the key K. If A, B and S are the principals of the protocols (with S the trusted server), Na and Nb are fresh nonces and Kas, Kbs and Kab are symmetric keys shared by the principals in the subscript, the Yahalom protocol can be described as follows:

$$\begin{array}{l} 1.\ A \rightarrow B\ :\ A, Na \\ 2.\ B \rightarrow S\ :\ B, Nb, \{A, Na\}_{Kbs} \\ 3.\ S \rightarrow A\ :\ Nb, \{B, Kab, Na\}_{Kas}, \{A, B, Kab, Nb\}_{Kbs} \\ 4.\ A \rightarrow B\ :\ \{A, B, Kab, Nb\}_{Kbs}, \{Nb\}_{Kab} \end{array}$$

It is also assumed that principal A only knows elements in the set $\{A, B, S, Kas\}$, the knowledge of principal B is given by $\{B, S, Kbs\}$, the trusted server S knows $\{A, B, Kas, Kbs\}$.

User A starts the protocol by communicating to B its intention to share a new session key with it (step 1). User B generates a fresh nonce Nb and creates a new term containing the identity of A and its nonce Na encrypted with the key Kbs shared with the trusted server. User B sends its identity, the fresh nonce Nb and the encrypted term to the server (step 2). Server S deciphers the encrypted term, obtains the identity of A and generates a new fresh key Kab. It also builds two terms encrypted with Kas and Kbs, and sends the whole message to A (step 3). Finally, A deciphers the first encryption and checks whether it contains nonce Na. If this is the case, A sends to B the second term encrypted with Kbs and mutually authenticates to B by sending nonce Nb encrypted with the fresh session key (step 4).

The basic requirements that this protocol must satisfy are the secrecy of the key Kab (in every session, the value of Kab must be known only by the participants playing the roles of A, B and S) and a proper authentication of the principal A to the principal B.

In a timed setting, the protocol can be adapted to take timeouts and retransmissions into account. In step 1, after sending its message, A may start a timer while waiting for the message of step 3. If the timeout occurs, A may retransmit its message (even the same nonce Na can be resent as it was already sent in clear). As some amount of time must pass while B and S receive and elaborate messages, if A receives an answer too early, this might signal some misbehaviour as, for example, the reception of a faked message from the intruder. Therefore, if we suppose that A knows the encryption and decryption times of B and S, we might adapt the Yahalom protocol to take time into account. In a formalism where time aspects are not considered this would not be possible.

6.3.3 Specification of the Yahalom Protocol through SDMTAs

We model the execution of the Yahalom protocol through a CSDMTA $\mathcal{AC} = (C, \Omega, A, B, S, I)$, where A, B and S are the DMTAs representing the principal A, the principal B and the trusted server S, respectively, while I is the intruder as specified in Definition 6.4.

The principals in the protocol are specified by the following DMTAs, graphically represented in Figure 6.3:

$$\begin{array}{l} A = (\{c\}, \{x, t\}, \Upsilon^A, Q^A, q_0, \phi_0^A, \{a, b, s, k_{as}\}, \delta^A); \\ B = (\{c\}, \{y\}, \Upsilon^B, Q^B, r_0, \phi_0^B, \{b, s, k_{bs}\}, \delta^B); \\ S = (\{c\}, \{z\}, \Upsilon^S, Q^S, u_0, \phi_0^S, \{a, b, s, k_{as}, k_{bs}\}, \delta^S). \end{array}$$

We assume that all communications happen on the only public channel c. All initial formulas ϕ_0^i reset to 0 the clocks of the DMTA i and set message variables to a random value. All principals $i \in \{A, B, S\}$ have the set of variable messages $\Upsilon^i = \{\mu_1^i, \dots, \mu_6^i\}$. Note that, for simplicity, we assumed more than a single variable modelling a natural value. Even if this is not in accordance with the hypothesis of Corollary 6.2, DMTAs with a single natural variable can be obtained by adding some transitions in each step a natural value is received within a nonce; the additional transition is used to instantiate a special nonce variable with the value of the received nonce (treating it as a single constant element instead of treating it as a function composed by an integer variable).

For readability, in Figure 6.3 we use the notation $N_\mu^{\mu'}$ for $Nonce(\mu, \mu')$; τ, τ' for $Pair(\tau, \tau')$ and $\{\tau\}_\mu$ for $Enc(\tau, \mu)$. Moreover, we suppose that when a message $(\tau_1, \dots, \tau_n)$ is send/received, $\tau_1, \dots, \tau_n$ fall in the knowledge (this is simply implementable by a finite sequence of ϵ transitions).

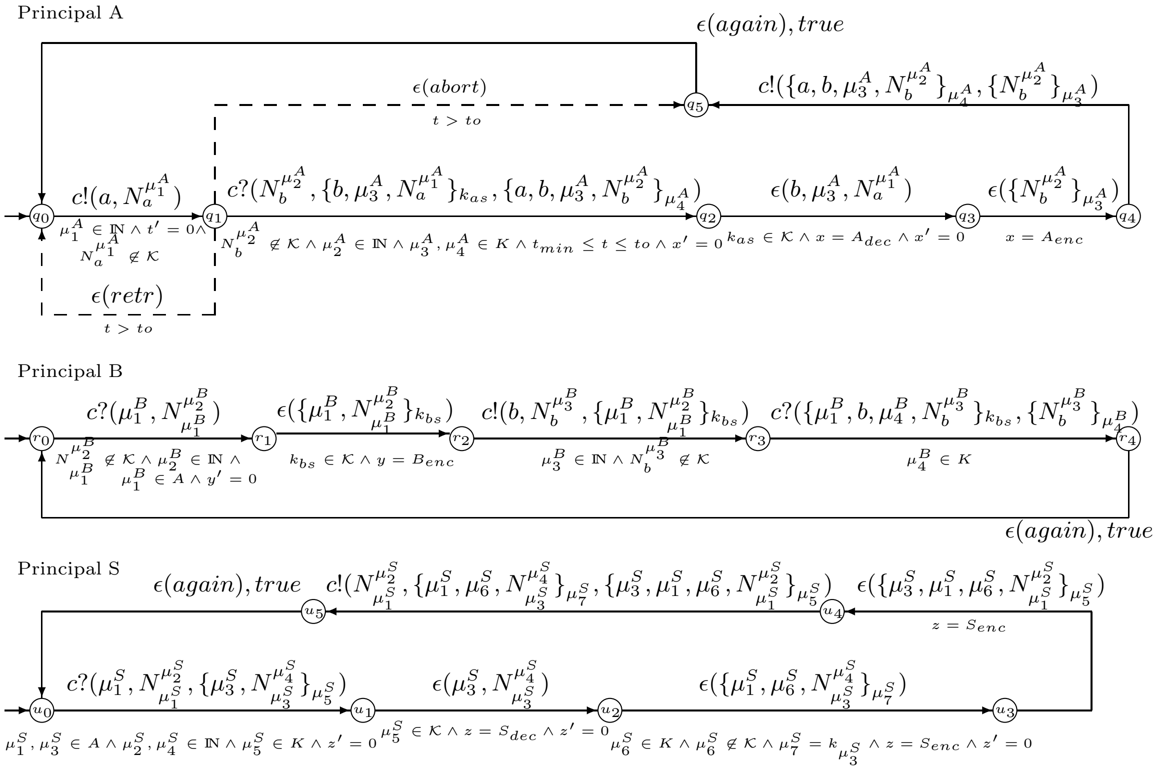

Figure 6.3: CSDMTA descriptions of the principals of the Yahalom protocol.

In the figure, for all principals, we omit the condition $\forall \mu \in \Upsilon^i\, \mu' = \mu$ from transition guards, but we assume that it holds for all transitions but the transitions with label $\epsilon(again), true$, where the guard $true$ means that message variables assume a value nondeterministically.

Note that $\mathcal{K}$ on a transition is intended to refer to the knowledge in the configuration from which the transition is performed.

If we do not consider the intruder, which is always capable of intercepting all messages, the guarantee that each message sent within the network will be received by the right participant is given by the reducibility of the terms sent.

The protocol is started by A that sends a message $a, N_a^{\mu_1^A}$, where $N_a^{\mu_1^A}$ is a fresh unused nonce, and starts waiting for a reply from S. In the formula on this transition, $N_a^{\mu_1^A}$ should not be in the knowledge of A. When A sends the message to B it also resets a timer t to 0, and a timeout can be fixed (in the figure it is fixed to the value to). Moreover, we assume that A knows the encryption/decryption time of B and S, and hence it does not expect to receive a message before time t_{min}.

After B receives the message from A, it checks whether the nonce is unused, generates a fresh nonce, and builds the encrypted terms it will send to S. Note that B_{enc} is a constant representing the time needed by B for encrypting a term.

When S receives the term from B it deciphers the encrypted term and builds the ciphered terms it should send to A. Again, S_{dec} and S_{enc} are constant values representing the time needed by S to decipher and cipher a term.

Note that the key chosen by S through the instantiation of the variable μ_6^S is fresh (since it does not appear in the knowledge of S, it was not chosen before).

When S finishes to elaborate the messages, it sends to A the message containing the new fresh key in the two subterms ciphered with k_{as} and k_{bs}, respectively.

If A receives such a message before the timeout has expired, it extracts the new fresh key and uses it to cipher the nonce sent by B to S, and then sends to B the ciphered nonce together with the term containing the fresh key built by S for B.

Some policies can be implemented when A does not receive an answer before the timeout expires. In Figure 6.3, we considered a couple of them (see the dashed transitions from q_1). In one case, A may think that B or S are "down" and abort the execution of the protocol (dashed transition to state q_5). In the other case, A may think that its first message was lost, and retransmits its first message (dashed transition to state q_0). With our methodology, both policies are proved to be secure.

Given the description of the Yahalom protocol through the model of CSDMTA, we can, in fact, prove the secrecy for the fresh generated key Kab (the intruder is not be able to deduce any information about the new session key). In particular we require that after the generation of the key (state u_3 of S) the key $I(\mu_6^S)$ is $\{I\}$-secret for the CSDMTA $\mathcal{AC}$ modeling the execution of the protocol.

We can also prove that the principal A truly authenticates to the principal B (during the execution of the communication protocol no other principal is able to simulate of being A).

These properties can be easily verified through the reachability proof method, as we have shown in Example 6.4.

Let us consider a variant of the protocol in which a second timer is set by B after sending its message at step 2. In this case, if a timeout occurs, the retransmission decision is more delicate. It is not clear whether B should resend the original message, should send a new nonce or should abort. Intuitively, the nonce Nb could be reused. In fact, if we implement such a retransmission policy and we check again the secrecy and authentication properties, we obtain that the protocol it still secure, thus confirming that, in this case, retransmission of the same message by B is secure.

Part III

Model Checking Security

Chapter 7

Automatic Analysis of a Non-Repudiation Protocol

Repudiation is defined as denial by one of the entities involved in a communication of having participated in all or part of the communication. One speaks of repudiation of the origin if the originator of a message denies having sent the message, and of repudiation of receipt if the recipient of the message denies having received the message.

Protocols have been defined which ensure non-repudiation by making use of a trusted third party in the communication. Protocols involving no third party have been proposed in faultless scenarios. Markowitch and Roggeman [118] give the probabilistic protocol (introduced in Section 5.5.1) which achieves fair non-repudiation services without the need of a third party and without further assumptions. In particular, the probabilistic protocol is fair up to a given tolerance ε. This tolerance depends on the values chosen for various parameters of the protocol.

In [7], the protocol is described by means of a probabilistic process algebra and analyzed, in an untimed setting, through a notion of probabilistic weak bisimulation. In this chapter we translate into PRISM descriptions [125] the Probabilistic Timed Systems modeling the protocol. We shall estimate how the tolerance varies when the parameters of the protocol are varied.

In Section 7.1 we recall Probabilistic Timed Systems. In Section 7.2 we use the Probabilistic Timed Systems to model the protocol. In Section 7.3 we translate our model into PRISM descriptions. In Section 7.4 we show the results obtained by running the PRISM model checker.

7.1 Probabilistic Timed Systems

We give a definition of Probabilistic Timed Systems that is a subclass of Probabilistic Timed Automata (see Section 2.1) assuming discrete time domain. Let us assume a set X of integer variables, with a subset Y of variables called *clocks*. A *valuation* over X is a mapping $v : X \rightarrow \mathbb{Z}$ assigning natural values to clocks and integer values to variables in $X \setminus Y$. For a valuation v and a time value $t \in \mathbb{N}$, let $v + t$ denote the valuation such that $(v + t)(x) = v(x)$, for each integer variable $x \in X \setminus Y$, and $(v + t)(y) = v(y) + t$, for each clock $y \in Y$. The set of *constraints* over X, denoted $\Phi(X)$, is defined by the following grammar, where ϕ ranges over $\Phi(X)$, $x \in X$, $c \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \neq, >, \geq\}$:

$$\phi ::= x \sim c \,|\, \phi \wedge \phi \,|\, \neg\phi \,|\, \phi \vee \phi \,|\, true$$

We write $v \models \phi$ when *the valuation v satisfies the constraint ϕ*. Formally, $v \models x \sim c$ iff $v(x) \sim c$, $v \models \phi_1 \wedge \phi_2$ iff $v \models \phi_1$ and $v \models \phi_2$, $v \models \neg\phi$ iff $v \not\models \phi$, $v \models \phi_1 \vee \phi_2$ iff $v \models \phi_1$ or $v \models \phi_2$, and $v \models true$.

An *assignment* over X is a set of expressions either of the form $x' = c$ or of the form $x' = y + c$, where $x, y \in X$ and $c \in \mathbb{Z}$.

With $Ass(X)$ we denote the set of sets of assignments $\{x'_1 = \rho_1, \ldots, x'_n = \rho_n\}$ such that $x_i \in X$ and $x_i \neq x_j$, for any $i \neq j$.

Let $B \in Ass(X)$; with $v[B]$ we denote the valuation resulting after the assignments in B. More precisely, $v[B](x) = c$ if $x' = c$ is in B, $v[B](x) = v(y) + c$ if $x' = y + c$ is in B, and $v[B](x) = v(x)$, otherwise.

A *Probabilistic Timed System* A is a tuple $(\Sigma, X, Q, q_0, \delta, \gamma, \pi)$, where:

- Σ is a finite alphabet of *actions*. With $\tau \in \Sigma$ we denote the *silent* or *internal* action.
- X is a finite set of variables with a subset Y of clocks.
- Q is a finite set of *states* and $q_0 \in Q$ is the *initial state*.
- $\delta \subseteq Q \times \Sigma \times \Phi(X) \times Ass(X) \times Q$ is a finite set of *transitions*. With $\delta(q)$, we denote the set of transitions starting from state q. More precisely, $\delta(q) = \{(q_1, \alpha, \phi, B, q_2) \in \delta \,|\, q_1 = q\}$.
- γ *is an interval function* such that for each $e = (q_1, \alpha, \phi, B, q_2) \in \delta$, it holds that $\gamma(e)$ is a closed interval of naturals. With $|\gamma(e)|$ we denote the natural value $u - l + 1$, where $\gamma(e) = [l, u]$. Intuitively, $\gamma(e)$ represents an interval of time within which the time t when the transition e will be performed is probabilistically chosen. It must also hold that the constraint ϕ is true during the interval $\gamma(e)$.
- π is a *probability function* such that for each state q and transition $e \in \delta(q)$, it holds that $\pi(e) \cdot \frac{1}{|\gamma(e)|}$ is the probability of performing the transition e from state q in a generic time t in $\gamma(e)$. Hence, we have a uniform distribution for time t, since for each time $t \in \gamma(e)$ the probability is fixed to $\pi(e) \cdot \frac{1}{|\gamma(e)|}$. Therefore we require that for each state q it holds that $\sum_{e \in \delta(q)} \pi(e) \in \{0, 1\}$.

A *configuration* of A is a pair (q, v), where $q \in Q$ is a state of A, and v is a valuation over X. The set of all the configurations of A is denoted with $\mathcal{S}_A$.

There is a *step* from a configuration $s_1 = (q_1, v_1)$ to a configuration $s_2 = (q_2, v_2)$ through action $a \in \Sigma$, after $t \in \mathbb{N}$ time units, written $s_1 \stackrel{(a,t)}{\longrightarrow} s_2$, if there is a transition $e = (q_1, a, \phi, B, q_2) \in \delta$ such that $(v_1 + t) \models \phi$, $v_2 = (v_1 + t)[B]$ and $t \in \gamma(e)$. With $prob(s_1 \stackrel{(a,t)}{\longrightarrow} s_2)$ we denote the probability $\sum_{e \in E} \pi(e) \cdot \frac{1}{|\gamma(e)|}$, where E is the set of transitions that could have triggered the step $s_1 \stackrel{(a,t)}{\longrightarrow} s_2$, namely the set $\{e = (q_1, a, \phi, B, q_2) \in \delta \mid (v_1 + t) \models \phi \wedge v_2 = (v_1 + t)[B] \wedge t \in \gamma(e)\}$.

If s is a configuration, then with $Adm(s)$ we denote the set of steps $s \stackrel{(a,t)}{\longrightarrow} s'$. The configuration s is called *terminal* iff $Adm(s) = \emptyset$.

An *execution fragment* starting from s_0 is a finite sequence of steps $\sigma = s_0 \stackrel{(a_1,t_1)}{\longrightarrow} s_1 \stackrel{(a_2,t_2)}{\longrightarrow} s_2 \stackrel{(a_3,t_3)}{\longrightarrow} \ldots \stackrel{(a_k,t_k)}{\longrightarrow} s_k$ such that $s_0, s_1, \ldots, s_k \in \mathcal{S}_A$, $a_1, a_2, \ldots, a_k \in \Sigma$. We define $last(\sigma) = s_k$, $|\sigma| = k$ and σ^j the sequence of steps $s_0 \stackrel{(a_1,t_1)}{\longrightarrow} s_1 \stackrel{(a_2,t_2)}{\longrightarrow} \ldots \stackrel{(a_j,t_j)}{\longrightarrow} s_j$, where $j \leq k$. Moreover we define the following probability

$$P(\sigma) = \begin{cases} 1 & \text{if } k = 0 \\ P(\sigma^{k-1}) \cdot \frac{prob(s_{k-1} \stackrel{(a_k,t_k)}{\longrightarrow} s_k)}{\sum_{g \in Adm(s_{k-1})} prob(g)} & \text{if } k > 0 \end{cases}.$$

The execution fragment σ is called *maximal* iff $last(\sigma)$ is terminal. We denote with $ExecFrag(s)$ the set of execution fragments starting from s.

An *execution* is either a maximal execution fragment or an infinite sequence $s_0 \stackrel{(a_1,t_1)}{\longrightarrow} s_1 \stackrel{(a_2,t_2)}{\longrightarrow} \ldots$, where $s_0, s_1 \ldots \in \mathcal{S}_A$, $a_1, a_2, \ldots \in \Sigma$. We denote with $Exec(s)$ the set of executions starting from s. Finally, let $\sigma\uparrow$ denote the set of executions σ' such that $\sigma \leq_{prefix} \sigma'$, where *prefix* is the usual prefix relation over sequences. Assuming the basic notions of probability theory (see e.g. [59]) we define the probability space on the executions starting in a given configuration $s \in \mathcal{S}_A$ as follows. Let $Exec(s)$ be the set of executions starting in s, $ExecFrag(s)$ be the set of execution fragments starting in s, and $\Sigma_F(s)$ be the smallest sigma field on $Exec(S)$ that contains the basic cylinders $\sigma \uparrow$, where $\sigma \in ExecFrag(s)$. The probability measure $Prob$ is the unique measure on $\Sigma_F(s)$ such that $Prob(\sigma \uparrow) = P(\sigma)$.

7.1.1 Parallel composition

Given two probabilistic timed systems A_1 and A_2, and given the set $L = \Sigma_{A_1} \bigcap \Sigma_{A_2}$, where Σ_{A_i} is the set of actions of the system A_i, we define the parallel composition of A_1 and A_2, denoted $A_1||^p A_2$, where $p \in [0, 1]$. Intuitively p represents the different advancing speeds for the two systems. The set of states of $A_1||^p A_2$ is given by the cartesian product of the states of the two systems A_1 and A_2. Given a state (r, q) of $A_1||^p A_2$, the set of transitions starting from (r, q) is obtained by the following rules:

- If from state r the system A_1 has a transition $e = (r, a, \phi, B, r')$ with action $a \notin L$ and probability p', $A_1||^p A_2$ has a transition $((r, q), a, \phi, B, (r', q))$ with probability $p{\cdot}p'$ and interval $\gamma(e)$.

- If from state q the system A_2 has a transition $e = (q, a, \phi, B, q')$ with action $a \notin L$ and probability p', $A_1||^p A_2$ has a transition $((r, q), a, \phi, B, (r, q'))$ with probability $(1 - p) \cdot p'$ and interval $\gamma(e)$.

- If from state r the system A_1 has a transition $e = (r, a, \phi_1, B_1, r')$ with action $a \in L$ and probability p', and from state q the system A_2 has a transition $e' = (q, a, \phi_2, B_2, q')$ with probability p'' and $B_1 \cup B_2 \in Ass(X)$, then A_1 and A_2 synchronize and $A_1||^p A_2$ has a transition $((r, q), a, \phi_1 \wedge \phi_2, B_1 \cup B_2, (r', q'))$ with probability $p \cdot p' + (1 - p) \cdot p''$ and interval $\gamma(e) \cap \gamma(e')$.

When we omit parameter p from the composition operator we assume the two systems to have the same advancing speeds, and hence p equal to $\frac{1}{2}$.

7.2 Modeling the Non-Repudiation Protocol with PTSs

In this section we use the model of Probabilistic Timed Systems to formally describe the protocol seen in Section 5.5.1. When no constraint is put on a transition we assume it will be taken instantly. Moreover, if for the transition starting from a certain state q we omit probabilities, then the transitions with source q are equiprobable. Hence, if a state has only one transition e, then $\pi(e) = 1$, and, if a state has two transitions e and e', then $\pi(e) = \pi(e') = \frac{1}{2}$. We start with introducing the Probabilistic Timed Systems modeling an originator and a recipient behaving correctly. The originator (Figure 7.1) is always ready to start a communication by accepting a request, sending the first message containing M encrypted with K (action $firstmes$) and receiving the first ack (see steps 1, 2 and 3 of the protocol in Section 5.5.1). Then, in state q_3, with probability $1 - p$, it sends a random message reaching state q_2 and, with probability p, sends the last message containing K, sets the variable l to 1, and reaches state q_4 (step 4 of the protocol). We do not model value passing, hence we simply call all these actions *message*. In state q_2 the reception of the ack message is modeled by the input action *ack*, while the expiration of the deadline (represented by the constant AD) is modeled by the action *stop* executed when the clock x assumes a value greater than AD. The fair termination of the protocol is reached when the originator receives the last ack (step 5 of the protocol) and performs the action *correctstop*. The protocol terminates in an unfair way if and only if the originator does not receive the ack related to the message containing K, and in such a case it executes the action $unfair$. The constants ad and AD used in the constraints of the *ack* transitions, represent an estimation of the minimum and maximum transmission delay of

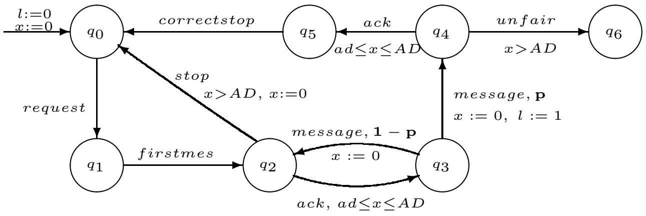

Figure 7.1: Representation of *Orig*.

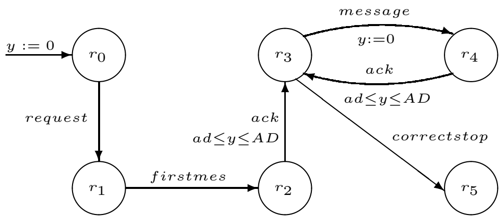

Figure 7.2: Representation of *HRecip*.

an ack message, respectively. In particular, we assume that an ack, sent within the net, will always arrive at destination in time t such that $ad \leq t \leq AD$.

We modeled the actions *request*, $firstmes$, and *message* as instant actions, since the time needed for the transmission of such messages in the network is not interesting in our analysis.

In Figure 7.2, we show the system representing a recipient that behaves correctly. The recipient starts the protocol by sending a request, receives the first message, sends the first ack and reaches state r_3, from where, whenever it receives a message, it sends an ack back. The protocol terminates when the input action *correctstop* is executed.

The whole system representing the protocol is defined as $Orig||HRecip$, where originator and recipient synchronize through actions in the set $L = \{request, firstmes, ack, message, correctstop\}$.

The protocol ends in a fair correct way when the action *correctstop* is performed, i.e. when the system reaches the state (q_0, r_5). In particular, if both participants behave correctly, the unfair behavior cannot be executed; instead, it is possible to find a malicious recipient that receives the expected information and denies sending the final ack.

In Figure 7.3 we show the system representing a malicious recipient that maximizes the probability of guessing the last message of the protocol (we assume that it knows the probability distribution chosen by the originator). It follows a Bernoulli distribution with parameter q to decide either to send the ack message (transition τ from state r_4 to state r_6) or to try to compute M by employing the last received message (transition from state r_4 to state r_5). We assume that the time necessary to decipher the message is within the interval $[dd, DD]$. Note that if $ad + dd < AD$ the recipient can send an ack even after failing to decipher the message (see transition from r_5 to r_6). So the originator should take care of the protocol parameters ad, AD, dd and DD. State r_7 represents, instead, the state reached by the malicious recipient when correctly guessing the last message. Since we set the variable l to 1 when the originator sends the last message, the malicious recipient succeeds in its strategy when in state r_5 such a variable has value 1 reaching the final state r_7. On the other hand, if variable l has value 0, the malicious recipient goes to state r_6, from which it tries to send the ack back.

The probability of executing the action $unfair$ for the system $Orig||HRecip$ is equal to 0, while

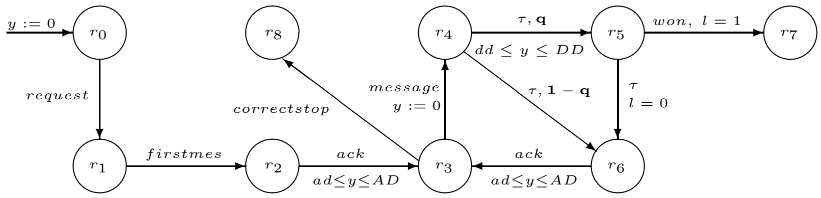

Figure 7.3: Representation of *MRecip*.

the probability of executing it for the system $Orig||MRecip$ is (as $ad + dd > AD$):

$$z = p \cdot q \cdot \sum_{i=0}^{\infty}((1-p)\cdot(1-q))^i = \frac{p \cdot q}{1-(1-p)\cdot(1-q)}.$$

Given $0 < p \leq 1$ chosen by the originator and $0 < q \leq 1$, the maximum value for z is p, obtained by taking $q = 1$. The recipient model, which optimizes the probability of violating the fairness condition, is obtained by removing the transition labeled with τ from state r_4 to state r_6.

7.2.1 The case of slow networks

As we have seen the probability for the malicious user to break the protocol is always smaller than p, which is a parameter decided by the designer of the protocol. This happens, obviously, when the condition $ad+dd > AD$ holds. In such a case, in fact, the malicious user could only try to decrypt the first message with the last received one as key (risking in this case to stop the protocol), or send an ack to the originator.

On the other hand, the condition above holds only if the time needed to send/receive an ack within the network is smaller than the time needed to decrypt a message within a given cryptosystem. Could we still use the protocol with a reasonable margin of risk in a network where the maximum acknowledgement delay time is bigger than the maximum decryption time, or in a network that is frequently subject to congestion? If the condition $ad + dd > AD$ does not hold, in fact, the malicious user could try to send an ack to the originator even after trying to decrypt the last received message.

We are interested in analyzing how the probability of breaking the protocol fairness increases when operating with a network with a long round trip time (high values for parameter AD), and in networks that are frequently subject to congestions (high values for the length of the interval $[ad, AD]$).

7.3 The PRISM Verifier

PRISM [125] is a probabilistic model checker that allows modeling and analyzing systems which exhibit a probabilistic behavior. Given a description of the system to be modeled, PRISM constructs a probabilistic model that can be either a *discrete-time Markov chain* (DTMC), a *Markov decision process* (MDP), or a *continuous-time Markov chain*(CTMC) [83]. On the constructed model PRISM can check properties specified by using a temporal logic (PCTL for DTMCs and MDPs, and CSL for CTMCs).

A system in PRISM is composed of modules and variables. A module has local variables and its behavior is expressed by commands of the form:

$$[sym]\ g \rightarrow \lambda_1 : u_1 + \ldots + \lambda_n : u_n$$

where sym is the synchronization symbol, g is a guard on all the variables in the system and u_i is a set of updates on local variables. The constant λ_i is the probability of performing the update u_i.

Constraints can be of the form $x \sim c$, where c is a natural and x a variable of the whole system. Updates are expressed by using primed variables, as an example $x' = 0$ represents the reset to 0 of variable x.

Since we used the model of DTMC, we use the PCTL specification language [61, 22, 16] to specify properties of systems.

The syntax of PCTL is given by the following grammar:

$$\phi ::= true \mid false \mid a \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathcal{P}_{\sim p}[\psi]$$
$$\psi ::= X\phi \mid \phi\mathcal{U}^{\leq k}\phi \mid \phi\mathcal{U}\phi$$

where a is an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$ is a relational operator, $p \in [0, 1]$ is a probability, and k is an integer.

An atomic proposition a is satisfied or not by a given state of a Probabilistic Timed System. Symbol X denotes the "next state operator", symbol $\mathcal{U}$ denotes the "until" operator, and $\mathcal{U}^{\leq k}$ denotes the "bounded until" (i.e. within k steps) operator. Intuitively, $\phi_1\mathcal{U}\phi_2$ is satisfied when the formula ϕ_1 holds until ϕ_2 holds; $\phi_1\mathcal{U}^{\leq k}\phi_2$ is satisfied if ϕ_2 becomes true within k steps. Moreover, $\mathcal{P}_{\sim p}[\psi]$ is satisfied by a given set of computations iff the overall probability p' of the computations satisfying ψ is such that $p' \sim p$.

7.3.1 Modeling the Non-Repudiation Protocol in PRISM

In this section we show the PRISM description of the systems introduced in Section 7.2, and the results of some properties verified on such models[1].

Honest Recipient

In Figure 7.4 we show the PRISM code of the modules representing the originator (`Orig`) and an honest recipient (`HRecip`). A third module (`AckDel`) is used for modeling the probability distribution of the variable `t` representing the acknowledgement delay. We decided to use a uniform probability distribution for the variable `t`. So, given the random variable `t` that can range in the interval [`ad`,`AD`], we have that:

$$\forall k \in [\mathtt{ad}, \mathtt{AD}] \qquad p(\mathtt{t} = k) = \frac{1}{\mathtt{AD} - \mathtt{ad} + 1},$$

where $p(\mathtt{t} = k)$ represents the probability that variable `t` assumes value k.

The constant `p1` used in the `Orig` module represents the probability p used in the Probabilistic Timed System *Orig* seen in Section 7.2.

Given the above participants the protocol always ends in a fair correct way. When the protocol ends in a fair way, the recipient state variable `r` assumes value 5. As a consequence, we can verify that the protocol always ends in a fair way by simply checking the property:

$$\mathcal{P}_{\geq 1}[true\,\mathcal{U}\ (r = 5)].$$

The model checking for the above property took 0,004 seconds, giving as result that the property is true in all the states of the model.

Malicious Recipient

In Figure 7.5 we show the PRISM code of the modules representing the originator (`Orig`) and a malicious recipient (`MRecip`). As in the previous case, a third module (`AckDel`) is used for modeling the uniform probability distribution of the variable `t` representing the acknowledgement delay. Also the random variable `x`, used in the module `MRecip` and representing the time needed to decrypt a message, follows a uniform distribution with range [`dd`,`DD`]. The constant `q1` used in the `MRecip` module represents the probability q used in the Probabilistic Timed System *MRecip* seen in Section 7.2.

[1]Tests are done on a 1,15GHz AMD Athlon PC with Linux OS and 512 MB of RAM.

```
probabilistic

const AD = 3;
const ad = 1;

rate p1 = 0.001;

rate u1 = 1/(1+AD-ad);

module Orig
    q : [0..6];
    l : [0..1];

    [request]     (q=0)&(r=0)          ->  (q'=1);
    [firstmes]    (q=1)&(r=1)          ->  (q'=2);
    [ack_r]       (q=2)&(a=2)&(t<=AD)  ->  (q'=3);
    [stop]        (q=2)&(t>AD)         ->  (q'=0);
    [message]     (q=3)&(r=3)          ->  p1:(q'=4)&(l'=1) + (1-p1):(q'=2);
    [ack_r]       (q=4)&(a=2)&(t<=AD)  ->  (q'=5);
    [unfair]      (q=4)&(t>AD)         ->  (q'=6);
    [correctstop] (q=5)&(r=3)          ->  (q'=0);
endmodule

module HRecip
    r : [0..5];

    [request]     (r=0)&(q=0)  ->  (r'=1);
    [firstmes]    (r=1)&(q=1)  ->  (r'=2);
    [ack_s]       (r=2)&(a=0)  ->  (r'=3);
    [message]     (r=3)&(q=3)  ->  (r'=4);
    [ack_s]       (r=4)&(a=0)  ->  (r'=3);
    [correctstop] (r=3)&(q=5)  ->  (r'=5);
endmodule

module AckDel
    a: [0..2];
    t: [0..AD];

    [ack_s]  (a=0)&((r=2)|(r=4))  ->  (a'=1);
    []       (a=1)                ->  u1:(a'=2)&(t'=1) + u1:(a'=2)&(t'=2) + u1:(a'=2)&(t'=3);
    [ack_r]  (a=2)&((q=2)|(q=4))  ->  (a'=0);
endmodule
```

Figure 7.4: `Orig` + `HRecip`.

7.4 Experimental Results

Setting :	set_a	set_b	set_c	set_d
`ad`	10	1	1	1
`AD`	13	4	12	12
`dd`	14	1	1	1
`DD`	15	5	2	2
`p1`	0.1	0.1	0.1	0.001

Table 7.1: Protocol Settings.

We studied the probability for a malicious user of breaking the fairness of the non-repudiation for several settings of our model parameters. More precisely in the settings set_a, set_ b, set_c and set_d, we studied the probability of breaking the protocol fairness as a parameter of the value of `q1`, assuming various values for the constants `ad`, `AD`, `dd`, `DD` and `p1` (see table 7.1). Since the malicious user's state variable `r` assumes value 7 when it successfully gets its information without sending the last ack, the properties we used in order to estimate the probability of breaking the protocol fairness are of the form:

$$\mathcal{P}_{\geq v}[true\, \mathcal{U}\ (r = 7)].$$

```
probabilistic

const AD = 3;
const ad = 1;
const DD = 2;
const dd = 1;

rate p1 = 0.001;
rate q1 = 1;

rate u1 = 1/(1+AD-ad);
rate v1 = 1/(1+DD-dd);

module Orig
    q : [0..6];
    l : [0..1];

    [request]     (q=0)&(r=0)            ->  (q'=1);
    [firstmes]    (q=1)&(r=1)            ->  (q'=2);
    [ack_r]       (q=2)&(a=2)&(t+x<=AD)  ->  (q'=3);
    [stop]        (q=2)&(t+x>AD)         ->  (q'=0);
    [message]     (q=3)&(r=3)            ->  p1:(q'=4)&(l'=1) + (1-p1):(q'=2);
    [ack_r]       (q=4)&(a=2)&(t+x<=AD)  ->  (q'=5);
    [unfair]      (q=4)&(r=7)            ->  (q'=6);
    [correctstop] (q=5)&(r=3)            ->  (q'=0);
endmodule

module MRecip
    r : [0..8];
    x : [0..DD];

    [request]     (r=0)&(q=0)  ->  (r'=1);
    [firstmes]    (r=1)&(q=1)  ->  (r'=2);
    [ack_s]       (r=2)&(a=0)  ->  (r'=3);
    [message]     (r=3)&(q=3)  ->  (r'=4);
    []            (r=4)        ->  q1:(r'=5) + (1-q1):(r'=6)&(x'=0);
    []            (r=5)&(l=0)  ->  v1:(r'=6)&(x'=1) + v1:(r'=6)&(x'=2);
    [won]         (r=5)&(l=1)  ->  (r'=7);
    [ack_s]       (r=6)&(a=0)  ->  (r'=3);
    [correctstop] (r=3)&(q=5)  ->  (r'=8);
endmodule

module AckDel
    a : [0..2];
    t : [ad..AD];

    [ack_s]  (a=0)&((r=2)|(r=6))  ->  (a'=1);
    []       (a=1)                ->  u1:(a'=2)&(t'=1) + u1:(a'=2)&(t'=2) + u1:(a'=2)&(t'=3);
    [ack_r]  (a=2)&((q=2)|(q=4))  ->  (a'=0);
endmodule
```

Figure 7.5: `Orig` + `MRecip`.

Thus, we can approximate to v the probability of breaking the protocol when the property $\mathcal{P}_{\geq v}[true\,\mathcal{U}\ (r = 7)]$ is satisfied by the initial state of the model, and the property $\mathcal{P}_{\geq v+\varepsilon}[true\,\mathcal{U}\ (r = 7)]$ is not, with ε a small value. The checking procedure of a property of the above type always took less then half a second.

Figures 7.6, 7.7 and 7.8 show the probability of breaking the protocol fairness as a function of `q1` for the parameter settings in table 7.1.

In the set_a setting, we have that $\mathtt{ad} + \mathtt{dd} > \mathtt{AD}$. In such a case the malicious recipient could not send an ack after trying to decrypt the first message using the last received one as key, and so the probability of breaking the protocol fairness, shown in Figure 7.6, is given by the formula $\frac{\mathtt{p1}\cdot\mathtt{q1}}{1-(1-\mathtt{p1})\cdot(1-\mathtt{q1})}$. As we have seen, the maximum value for this probability is represented by the parameter `p1`.

In the set_b, set_c and set_d settings we have, instead, that $\mathtt{ad} + \mathtt{dd} \leq \mathtt{AD}$. With these three settings we propose a first analysis of the case of slow networks described in Section 7.2.1. In such a case, as can be seen from Figures 7.7 and 7.8, we have that the probability of breaking the

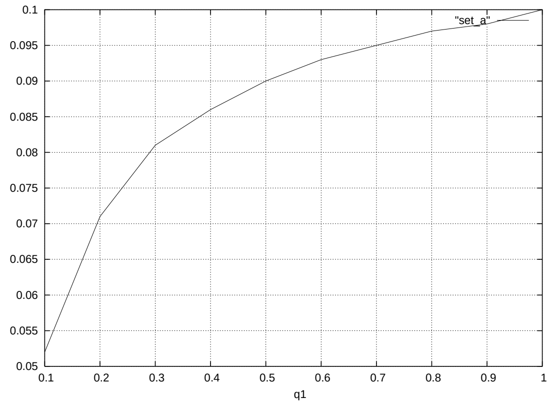

Figure 7.6: Varying `q1` in an ideal network.

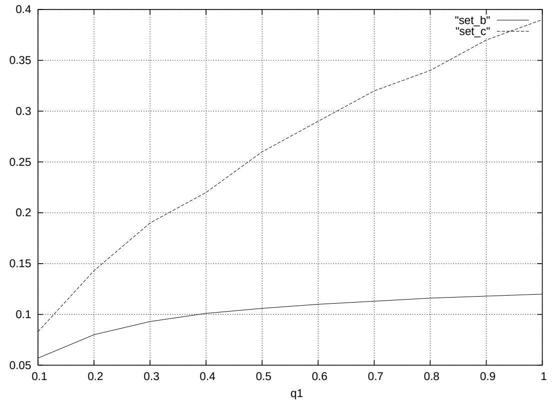

Figure 7.7: Varying `q1` in slow networks.

protocol fairness increases as `q1` goes to 1 and reaches also values that are bigger than `p1`.

In the settings `e`,`f`,`g`, we studied the probability of breaking the protocol fairness as a parameter of the value of `AD`, assuming various values for the constants `ad`, `dd`, `DD` and `p1` (see Table 7.2).

Figure 7.9 shows the probability of breaking the protocol fairness as a function of `AD` for the parameter settings in Table 7.2. In particular those settings model the case of "slow and congestioned network" described in Section 7.2.1.

Obviously, as `AD` increases, also the probability of breaking the non-repudiation protocol increases. But, as it can be seen from Figure 7.9, with a small value of `p1`, the protocol could be reasonably used also in the case when the time needed to send an ack is bigger than the time needed to decrypt a message.

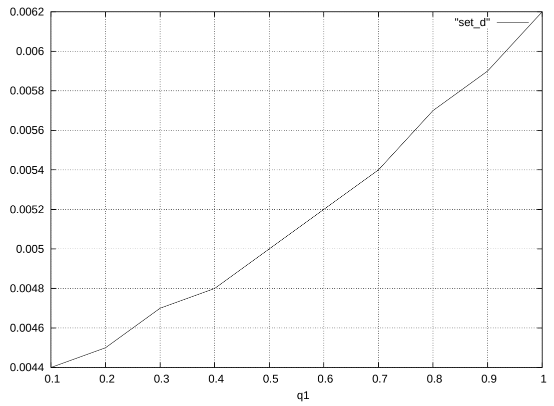

Figure 7.8: Varying `q1` in a slow network, given a low `p1`.

Setting :	set_e	set_f	set_g
`ad`	1	1	1
`dd`	1	1	1
`DD`	4	4	4
`p1`	0.001	0.01	0.1
`q1`	1	1	1

Table 7.2: Protocol Settings.

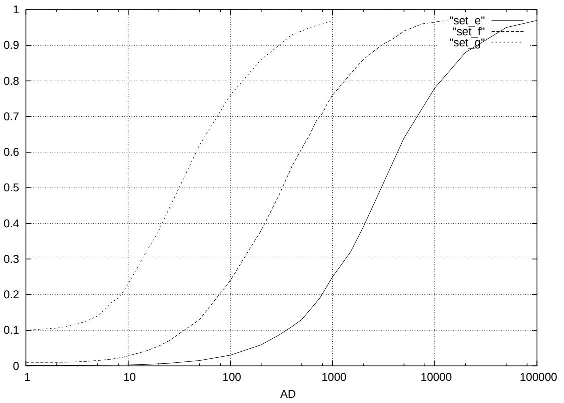

Figure 7.9: Analysis of congestioned networks.

Chapter 8

Automatic Analysis of the NRL Pump

A computer system may store and process information with a range of classification levels and provide services to users with a range of clearances. The system is *multilevel secure* [19] if users have access only to information classified at or below their clearance. In a distributed framework, multilevel security can be achieved by using trusted secure components, called *Multilevel Secure Components* (MSCs) [74, 75], to connect single-level systems bounded at different security levels, thus creating a *multiple single-level security* architecture [74, 75].

Many applications must satisfy *time performance* as well as *security* constraints which, as well known, are conflicting requirements (e.g. see [139]). This has motivated research into probabilistic protocols that can trade-off between security and time performance requirements. MSCs are an example of such protocols. Their role is to minimize leaks of high level information from high level systems to lower level systems without degrading average time performances.

An MSC proposal is the *Naval Research Laboratory's Pump* (*NRL Pump*) [114, 76, 77]. It is a trusted device that acts as a router forwarding messages from a low-level system to one at higher level. Now, acks are needed for reliable communication. If the high system passed acks directly to the low one, then it could pass high information by altering ack delays. To minimize such *covert channel*, the pump decouples the acks stream by inserting *random delays*. To avoid time performance degradation, the long-term high-system-to-pump time behavior should be reflected in the long-term pump-to-low-system time behavior. The NRL pump achieves this result by statistically modulating acks: the pump-to-low-system ack delay is probabilistic based on the moving average of the high-system-to-pump ack delays. This approach guarantees that the average time performances of the secure system (*with* the NRL pump) are the same as those of the insecure one (*without* the NRL pump).

Analysis of *information leakage* of protocols is usually carried out by estimating covert channel capacity [4, 37, 58, 139], which can be estimated by simulation, since an analytical approach usually cannot handle a full model of the protocol at hand. Using the NRL Pump case study, we show how *probabilistic model checking* can be exploited to estimate covert channel capacity for various system configurations. This allows a formal as well as automatic security analysis (see [107] for a survey on this subject) of the probabilistic protocol.

In [114] a NRL Pump assurance strategy is proposed. A logical view of the pump is refined using a combination of Statemate activity charts and state charts. Activities and behavior of the logical design are mapped to an implementation described in terms of Statemate module charts. A Verdi [35] specification expresses the requirements to which the implementation must be shown to conform. Verification is done through proof using the EVES [81, 82] verification system, and by testing using WhiteBox DeepCover tool [132].

In this chapter we model the NRL Pump by Probabilistic Automata enriched with variables. The advantage of using these model as representation formalism is twofold. Firstly, on such systems state-space reduction algorithms can be used. Second, classic automaton-theoretic means for analysis of properties of the described system can be employed. These systems and requirements on their behavior are then translated into specifications for the FHP-Murφ tool [121, 122, 27].

We show how FHP-Murφ, a probabilistic version of Murφ [45], can compute the probability of security violation of the NRL Pump protocol as a function of (discrete) time, for various configurations of the system parameters (e.g. buffer sizes, moving average size, etc). This allows us to estimate the capacity of the probabilistic covert channel left by decoupling the acks stream. Notwithstanding the huge amount of system states, we are able to complete our analysis and to compute covert channel capacity for various NRL pump parameters settings.

To the best of our knowledge, this is the first time that probabilistic model checking is used for a quantitative analysis of the covert channel capacity. Symbolic model checking, based on PRISM [125, 84, 84], has already been widely used for verification of probabilistic protocols. However, for protocols involving arithmetical computations or many FIFO queues, PRISM tends to fill up the RAM [121]. This is due to OBDDs troubles in representing arithmetical operations and FIFO queues. Since probabilistic protocols involving arithmetics or FIFO queues can be often conveniently analyzed using an explicit approach, we use FHP-Murφ to carry out our analysis. Note that indeed the Murφ verifier has already been widely used for security verification, e.g. see [45, 112, 113, 108]. We use FHP-Murφ instead of Murφ since FHP-Murφ extends Murφ capabilities to a probabilistic setting.

We note that an approximate analysis of the covert channel studied here is presented in [115]. However, because of model complexity, only bounds to covert channel capacity can be obtained in [115]. In such cases probabilistic model checking complements the analytical approach by allowing an *exact* analysis on some aspects (i.e., security) of models that are out of reach for the analytical approach.

As for simulation based results (see [77]), the interesting case is when covert channel capacity, as well as the probability of a security violation, is small. Hence, estimating such probabilities by a simulator can be quite hard. In fact, such analysis is not pursued in [77]. In such cases a model checker can be used as an efficient *exhaustive simulator*. Of course a model checker may have to handle a *scaled down* model (with model parameters instanced to *small enough* values) w.r.t. the model handled by a simulator (e.g. w.r.t. the model considered in [77]).

Summing up, we show how probabilistic model checking can complement the covert channel approximate analysis of [115] and the simulation results of [77].

In Section 8.1 we report a brief description of the NRL pump device. In Section 8.2 we define the formal framework of Probabilistic Systems that we use in Section 8.3 in order to formally describe the device. The FHP-Murφ specifications of the pump are reported in Section 8.4. In Section 8.5 we show the results of our analysis technique by estimating the covert channel capacity.

8.1 The NRL Pump

The NRL Pump is a special purpose trusted device that acts as a router forwarding messages from low level agents to high level ones by monitoring the timing of the acks in the opposite way. As shown in Figure 8.1, a low agent sends a message to some high agent through the pump. In order to make the communication reliable, the pump stores the message in a buffer and sends an ack to the low agent (in Figure 8.2, we formalize the communication steps between the agents). The delay of the ack is probabilistically based on a moving average of ack delays from the high agents to the pump. This is an attempt to prevent the high agent to alter the ack timing in order to send information to the low agent. Moreover, the long-term high-agents-to-pump behavior is reflected by the long-term pump-to-low-agents behavior, so that performance is not degraded. It is also assumed that the low level agent cannot send any new message until the acknowledgment of the previous message is received.

The pump keeps the message in the buffer until the high agent is able to receive it. When the high agent receives the message, it sends an ack to the pump. If the high agent does not

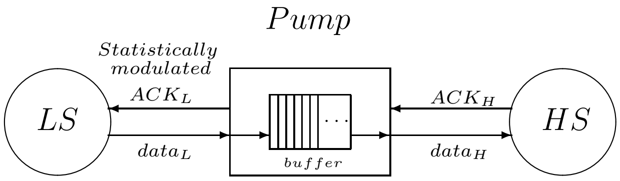

Figure 8.1: The NRL Pump

$LS \to P$	: $data_L$	low system sends to pump data to deliver to high system
$P \to LS$	: ACK_L	pump acknowledges to low system with a probabilistic delay
$P \to HS$	: $data_H$	the pump sends the data to the high system
$HS \to P$	: ACK_H	the high system acknowledges to the pump

Figure 8.2: Data communication protocol

acknowledge a received message before a timeout fixed by the pump-administrator expires, the pump stops the communication.

8.1.1 NRL Pump probabilistic ack delay modeling

Let $x_1, \ldots, x_n$ denote the delays of the last n acks sent by the high system to the pump, and $\overline{x}$ denote their average $\sum_{i=1}^{n} x_i/n$. We denote with $p(l, \overline{x})$ the probability that the pump sends an ack to the low system with a delay l when $\overline{x}$ is the average of received acks. Now, $p(l, \overline{x})$ can be defined in many ways each of which yields different *probabilistic ack schema* with security performances. Let us consider two possible scenarios.

In the first one, $l = \overline{x} + d$, with d a uniformly distributed random variable taking integer values in range $[-\Lambda, +\Lambda]$. Since the expected value of d (notation $E(d)$) is 0, we have $E(l) = \overline{x} + E(d) = \overline{x}$, as required by the NRL Pump specification. We have $p(l, \overline{x}) =$ **if** $(l \in [\overline{x} - \Lambda, \overline{x} + \Lambda])$ **then** $1/(2\Lambda + 1)$ **else** 0.

The drawback of this approach is that, if the schema is known to the low and high systems, then the following *deterministic covert channel* can be established. To transmit bit b $(b = 0, 1)$ to the low system, the high system sends h consecutive acks to the pump with a given delay H_b. If h is large enough, from the law of large numbers we will have $\overline{x} \sim H_b$, and $l \in [H_b - \Lambda, H_b + \Lambda]$. Without loss of generality let us assume $H_0 < H_1$. Whenever the low system *sees* an ack time $l \in [H_0 - \Lambda, H_1 - \Lambda)$ (resp. $l \in (H_0 + \Lambda, H_1 + \Lambda]$), it knows with certainty that a bit 0 (1) has been sent from the high system. Of course, if the ack time is in the interval $[H_1 - \Lambda, H_0 + \Lambda]$ the low system does not know which bit is being sent from the high system. However, if the high system is sending acks with delay H_0 (H_1) and h is large enough, then we know that (with high probability) the low system will observe an ack delay in $[H_0 - \Lambda, H_1 - \Lambda)$ (resp. $(H_0 + \Lambda, H_1 + \Lambda]$). Note that once the low system receives an ack with delay in $[H_0 - \Lambda, H_1 - \Lambda)$ (resp. $(H_0 + \Lambda, H_1 + \Lambda]$) then it is sure that the high system has sent bit 0 (1).

Note that the deterministic covert channel arises since the range of l depends on $\overline{x}$. To solve the problem, in the second scenario we use a *binomial distribution*.

Let $p \in [0, 1]$ and Δ be an integer. Let d be a random variable taking integer values in $[0, \Delta]$ with probabilities: $P(d = k) = \begin{pmatrix} \Delta \\ k \end{pmatrix} p^k (1-p)^{\Delta - k}$. The range of d does not depend on p. Let p be $(\overline{x} - 1)/\Delta$. Since d has a binomial distribution we have $E(d) = \Delta \cdot p = \Delta \cdot \frac{\overline{x}-1}{\Delta} = (\overline{x} - 1)$. We define the pump ack delay l as follows: $l = d + 1$. Then, $E(l) = \overline{x}$, as required from the NRL Pump specification.

Since the range of l does not depend on $\overline{x}$, the covert channel of the first scenario does not exist. However the high system can send information to the low one as follows. To transmit bit b $(b = 0, 1)$, the high system sends h consecutive acks to the pump with a given delay H_b. If h is large enough, from the law of large numbers we know that we will have $\overline{x} \sim H_b$. The low system can compute a moving average $\overline{y}$ of the last m ack delays from the NRL Pump. If m is large enough we have $\overline{x} \sim \overline{y}$. Then, by comparing $\overline{y}$ with H_0 and H_1, the low system can estimate (with arbitrarily low error probability) the bit b.

Now, the low system knows the bit b only in a probabilistic sense. In Section 8.5 we will show that the error probability depends on many parameters, and, by modeling the NRL Pump with FHP-Murφ [121], we will compute such error probability. This, in turn, allows us to estimate the covert channel capacity.

8.2 Probabilistic Systems with Variables

In this section we define a model of Probabilistic Systems with variables (PSs).

Let us assume a set X of real variables, a *valuation* over X is a mapping $v : X \to \mathbb{R}$ assigning real values to variables in X. We also assume a set of function symbols $\mathcal{F}$.

The set of *constraints* over X, denoted $\Phi(X)$, is defined by the following grammar, where ϕ ranges over $\Phi(X)$, $x_1, \ldots, x_n, y_1, \ldots, y_m \in X$, $f, q \in \mathcal{F}$ and $\sim \in \{<, \leq, =, \neq, >, \geq\}$:

$$\phi ::= f(x_1, \ldots, x_n) \sim g(y_1, \ldots, y_m) \,|\, \phi \wedge \phi \,|\, \neg\phi \,|\, \phi \vee \phi \,|\, true$$

We suppose a unique interpretation I for function symbols in $\mathcal{F}$ such that $I(f) : \mathbb{R}^n \to \mathbb{R}$ where n is the arity of the function f. Since the interpretation I is unique, when it is clear from the context, we may write f instead of $I(f)$.

We write $v \models \phi$ when *the valuation v satisfies the constraint ϕ*. Formally, $v \models f(x_1, \ldots, x_n) \sim g(y_1, \ldots, y_m)$ iff $I(f)(v(x_1), \ldots, v(x_n)) \sim I(g)(v(y_1), \ldots, v(y_m))$, $v \models \phi_1 \wedge \phi_2$ iff $v \models \phi_1$ and $v \models \phi_2$, $v \models \neg\phi$ iff $v \not\models \phi$, $v \models \phi_1 \vee \phi_2$ iff $v \models \phi_1$ or $v \models \phi_2$, and $v \models true$.

With X' we denote the set of variables $\{x' \,|\, x \in X\}$, where variable x' is the variable representing a new value to assign to x.

An *assignment* is a formula in $\Phi(X \cup X')$ of the form $x' = f(y_1, \ldots, y_m)$, where $x' \in X'$ and $\{y_1, \ldots, y_m\} \subseteq X$. As an example, the constraint $x' = x + 1$ updates x with its successor.

A set of assignments S is *complete* with respect to X if, for any $x \in X$, there exists a unique assignment $x' = f(y_1, \ldots, y_m)$ in S. The set of complete assignments on X is denoted by $\mathcal{S}(X)$.

We define an operator of valuation merging $\oplus$ such that given a valuation v, v' of variables in X, $v \oplus v'$ is a valuation on variables in $X \cup X'$ such that $v \oplus v'(x)$ returns $v(x)$ if $x \in X$ and $v'(y)$ if $x \in X'$ and $x = y'$.

Definition 8.1 *A* Probabilistic System *(PS for short) is a tuple $A = (\Sigma, X, Q, q_0, \phi_0, \delta, \pi)$, where:*

- *Σ is a finite alphabet of* actions*.*
- *X is a finite set of* variables*.*
- *Q is a finite set of* states *and $q_0 \in Q$ is the* initial state*.*
- *$\phi_0 \in \Phi(X)$ is the initial condition. We require that $|\{v | v \models \phi_0\}| = 1$.*
- *$\delta \subseteq Q \times \Sigma \times \Phi(X) \times \mathcal{S}(X) \times Q$ is a finite set of* transitions*. For a state q, we denote with $start(q)$ the set of transitions with q as source state, i.e. the set $\{(q_1, a, \phi, S, q_2) \in \delta \,|\, q_1 = q\}$; and we denote with $start(q, a)$ the set of transitions with q as source state and a as action, i.e. the set $\{(q_1, \alpha, \phi, S, q_2) \in \delta \,|\, q_1 = q$ and $\alpha = a\}$.*
- *$\pi : \delta \to]0, 1]$ is a probability function such that $\pi(e)$ is the probability of performing the transition e. We require that for each state q, $\sum_{e \in start(q)} \pi(e) \in \{0, 1\}$.*

In Figure 8.3 we show an example of Probabilistic System. When no condition is put on a transition we assume that it is the condition *true* and all assignments are $x' = x$ for every $x \in X$. We represent transition probabilities with bold numbers and we assume the probability is equal to 1 when omitted.

$$
\begin{array}{ll}
e_1 = (q_0, b, y \leq 5 \wedge \forall x \in X.x' = x, q_1) & \pi(e_1) = \frac{2}{3} \\
e_2 = (q_0, a, y > 5 \wedge y' = 0 \wedge \forall x \neq y \in X.x' = x, q_2) & \pi(e_2) = 1 \\
e_3 = (q_0, a, y \leq 5 \wedge \forall x \in X.x' = x, q_3) & \pi(e_3) = \frac{1}{3}
\end{array}
$$

$\forall (q, v)\ \sum_{e \in Adm(q,v)} \pi(e) \in \{0, 1\}$

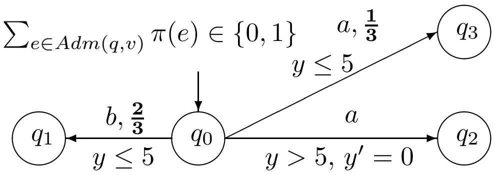

Figure 8.3: Example of Probabilistic System with Variables.

8.2.1 Semantics of PSs

A *configuration* of A is a pair $s = (q, v)$, where $q \in Q$ is a state of A, and v is a valuation over X. The initial configuration of A is represented by (q_0, v), where $v \models \phi_0$. The set of all the configurations of A is denoted with $\mathcal{S}_A$. We assumed a single initial configuration. This is not a limitation, since one can deal with a set of initial configurations by assuming a fictitious initial configuration with equiprobable transitions to all the configurations of the set.

There is a *transition step* from a configuration $s_i = (q_i, v_i)$ to a configuration $s_j = (q_j, v_j)$ through action $a \in \Sigma$, written $s_i \xrightarrow{a} s_j$, if there is a transition $e = (q_i, a, \phi, S, q_j) \in \delta$ such that $v_i \oplus v_j \models \phi \wedge \bigwedge_{\phi' \in S} \phi'$.

We note that, for any transition (q_i, a, ϕ, S, q_j) and valuation v_i, there exists a unique v_j such that $v_i \oplus v_j \models \phi \wedge \bigwedge_{\phi' \in S} \phi'$, since S is complete.

Given a configuration $s = (q_i, v_i)$, $Adm(s)$ represents the set of transitions that a system could execute from configuration s, and we say that a transition in $Adm(s)$ is *enabled* in s. Given $s_i = (q_i, v_i)$ and $s_j = (q_j, v_j)$, $Adm(s_i, a, s_j) = \{e = (q_i, a, \phi, S, q_j) \in start(q_i) \mid v_i \oplus v_j \models \phi \wedge \bigwedge_{\phi' \in S} \phi'\}$ denotes the set of transitions that lead from configuration s_i to configuration s_j through a transition labeled with a.

A configuration $s = (q_i, v_i)$ is called *terminal* iff $Adm(s') = \emptyset$, and we denote with S_T the set of terminal configurations. In order to deal with infinite runs we add self-loop steps for any terminal configuration. Namely, there is a self-loop step $(q_i, v_i) \xrightarrow{SL} (q_i, v_i)$ for any $(q_i, v_i) \in S_T$.

For configurations $s_i = (q_i, v_i)$, $s_j = (q_j, v_j)$ and $a \in \Sigma$, we define with $P(s_i, a, s_j)$ the probability of reaching configuration s_j from configuration s_i through a step $s_i \xrightarrow{a} s_j$ labeled with a. Formally, we have that $P(s_i, a, s_j) = \frac{\sum_{e \in Adm(s_i, a, s_j)} \pi(e)}{\sum_{e' \in Adm(s_i)} \pi(e')}$.

The probability of executing a transition step from a configuration s is chosen according to the values returned by the function π among all the transitions enabled in s. The probability of executing a self loop step $\xrightarrow{SL}$ is assumed to be equal to 1.

An *execution fragment* starting from s_0 is a finite sequence of transition steps $\sigma = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots \xrightarrow{a_k} s_k$, where $s_0, s_1, \ldots, s_k \in \mathcal{S}_A$ and $a_i \in \Sigma$. With $ExecFrag$ we denote the set of execution fragments and with $ExecFrag(s)$ the set of execution fragments starting from s. We define $last(\sigma) = s_k$ and $|\sigma| = k$.

For any $j < k$, with σ^j we define the sequence of steps $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_j} s_j$.

If $|\sigma| = 0$ we put $P(\sigma) = 1$, else, if $|\sigma| = k \geq 1$, we define $P(\sigma) = P(s_0, a_1, s_1) \cdot \ldots \cdot P(s_{k-1}, a_k, s_k)$.

An *execution* is an infinite sequence $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots$, where $s_0, s_1 \ldots \in \mathcal{S}_A$ and $a_1, a_2, \ldots \in \Sigma \cup \mathbb{N}^{>0} \cup \{SL\}$. We denote with $Exec$ the set of executions and with $Exec(s)$ the set of executions starting from s. Finally, with $\sigma \uparrow$ we denote the set of executions σ' such that $\sigma \leq_{prefix} \sigma'$, where *prefix* is the usual prefix relation over sequences.

Assuming the basic notions of probability theory (see e.g. [59]), we define the probability space on the executions starting in a given configuration $s \in \mathcal{S}_A$ as follows. Let $Exec(s)$ be the set of executions starting in s, $ExecFrag(s)$ be the set of execution fragments starting in s, and $\Sigma_{Field}(s)$ be the smallest sigma field on $Exec(s)$ that contains the basic cylinders $\sigma \uparrow$, where $\sigma \in ExecFrag(s)$. The probability measure $Prob$ is the unique measure on $\Sigma_{Field}(s)$ such that $Prob(\sigma \uparrow) = P(\sigma)$.

A Probabilistic Systems may have an infinite number of configuration even though the number of transitions exiting a state is finite. Therefore, Probabilistic Systems are equivalent to infinite-state Markov Chains. Infinite-state Markov Chains have been studied, for instance, in [103, 13].

8.3 Modeling the NRL Pump with PSs

In this section we show how the NRL Pump presented in Section 8.1 can be modeled by a PS. The PS NRL_Pump is shown in Figure 8.4.

We restrict our attention to the NRL Pump features involved in the probabilistic covert channel described before.

Our goal is to compute the error probability of the low system when it estimates the bit value sent from the high system. This probability depends on the number of consecutive high system acks to the pump with delay H_b and on several parameters, which are defined as constants. Without loss of generality, we consider the case $b = 0$.

The system representing the pump is specified by using the set of constants and variables shown in Table 8.1.

Cosntants :	Value
`BUFFER_SIZE`	5
`NRL_WINDOW_SIZE`	5
`LS_WINDOW_SIZE`	5
`INIT_NRL_AVG`	4
`INIT_LS_AVG`	0
`DELTA`	10
`HS_DELAY`	4
`DECISION_THR`	1
Variables :	Value assigned by *init* condition
`b`	0
`nrl_avg`	`INIT_NRL_AVG`
`nrl_delays[i]`	`INIT_NRL_AVG` ∀ i∈ [0,`NRL_WINDOW_SIZE`-1]
`nrl_first_index`	0
`ls_avg`	`INIT_LS_AVG`
`ls_delays[i]`	`INIT_LS_AVG` ∀ i∈ [0,`LS_WINDOW_SIZE`-1]
`ls_first_index`	0
`nrl_ack`	0
`ls_decision_state`	0
`ls_decision`	0

Table 8.1: Pump Constants and Variables

`BUFFER_SIZE` is the size of the buffer used by the pump to store the messages sent from the low system to the high one. `NRL_WINDOW_SIZE` is the size of the sliding window used by the pump to keep track of the last ack delays from the high system. These delays are used to compute the high-system-to-pump moving average. `LS_WINDOW_SIZE` is the size of the sliding window used by the low system to keep track of the last ack delays from the pump. These delays are used to compute the pump-to-low-system moving average. `INIT_NRL_AVG` is the initial value of high-system-to-pump moving average. `INIT_LS_AVG` is the initial value of pump-to-low-system moving average. `DELTA` is the maximal pump-to-low-system ack delay: each of these delays ranges in $[0, \mathtt{DELTA}]$ and is computed probabilistically. `HS_DELAY` is such that the delay used by the high system to transmit bit 0 is $H_0 = \mathtt{HS\_DELAY}$, and the delay used by the high system to transmit bit

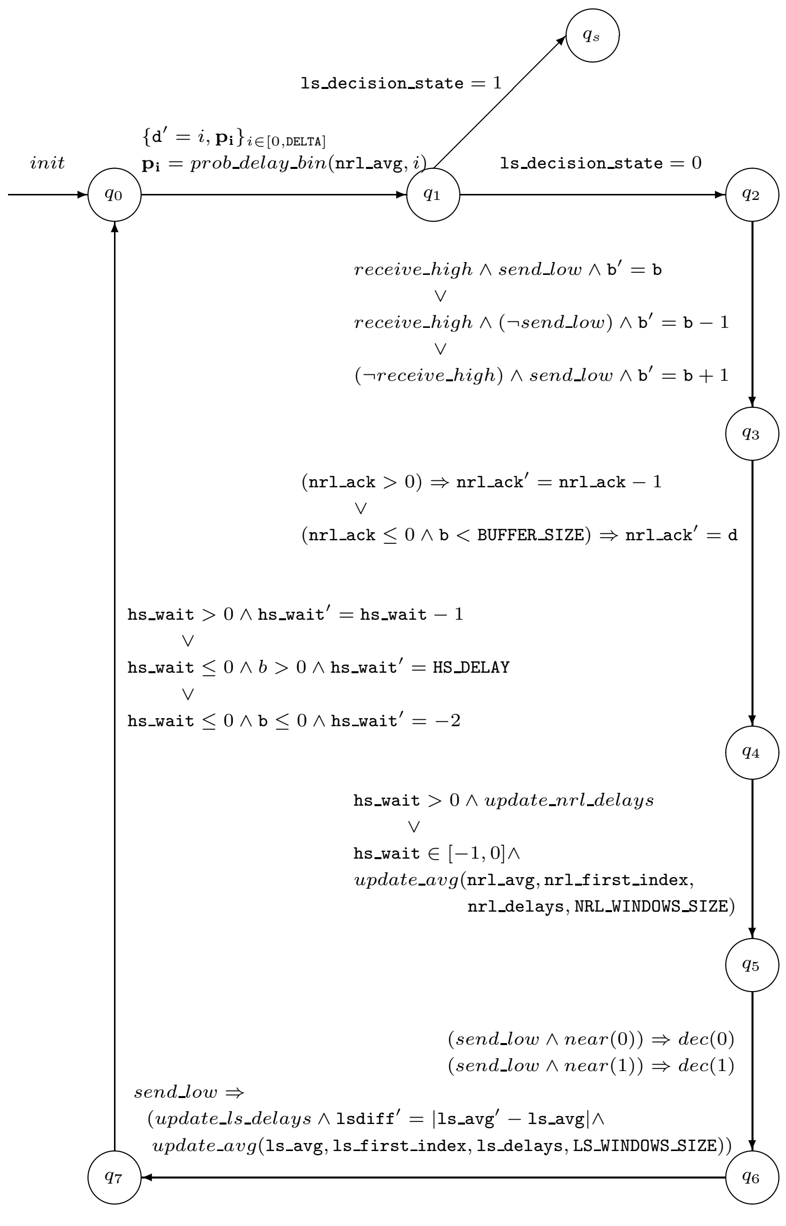

Figure 8.4: *NRL_pump*: PS modeling the NRL Pump

1 is $H_1 = \texttt{HS\_DELAY}+2.0$. DECISION_THR is such that the low system decides that the bit sent by the high system is b, for $b \in \{0,1\}$, when it receives an ack from the pump and the difference between the new and the old pump-to-low-system moving average is below DECISION_THR, i.e. when the pump-to-low-system moving average is stable enough. Now, waiting for a long time before making a decision (e.g. by making DECISION_THR small), the low system can be quite sure of making a correct decision, but the more the low system waits for making a decision, the smaller the *bit-rate* of this *covert channel*.

Valuations of global variables define the configuration state of our model. For simplicity we use arrays. They can be easily modeled in PS by using a number of variables equal to the size of the array. Variable b represents the number of messages in the pump buffer. Variable nrl_avg is the average of the last delays for the acks received by the pump from the high system. These delays are saved in array nrl_delays, where index nrl_first_index points to the eldest one. Variable ls_avg is the average of the last delays of the acks sent by the pump and received by the low system. These delays are saved in array ls_delays, where index ls_first_index points to the eldest one. Variable nrl_ack is the timer used by the pump for sending the next ack to the low system. Once initialized, at each step nrl_ack is decremented by 1. The pump sends the ack when $\texttt{nrl\_ack} \leq 0$. Similarly, hs_wait is the timer used by the high system for sending acks to the pump. Variable ls_decision_state is 0 if the low system has not yet decided the value of the bit sent by the high system, 1 if the low system has already decided, 2 if the whole system must be restarted after a decision. If ls_decision_state is 1 or 2, ls_decision takes the value of the bit.

In Table 8.2 and Table 8.3 we report the descriptions of formulas and functions referred to in Figure 8.4.

For readability, we have not written constraints to describe variables that do not change their value after the firing of a certain transition. Hence, we suppose that, if a variable x does not appear in a condition of a transition, then the condition contains also the formula $x' = x$.

Label :	$ConstraintDescription$
$send\_low$	$\texttt{nrl\_ack} \leq 0 \wedge \texttt{b} < \texttt{BUFFER\_SIZE}$
$receive\_high$	$\texttt{hs\_wait} \leq 0 \wedge \texttt{b} > 0$
$update\_nrl\_delays$	$\texttt{nrl\_delays}[\texttt{last}(\texttt{nrl\_first\_index})]' = \texttt{nrl\_delays}[\texttt{last}(\texttt{nrl\_first\_index})] + 1$
$update\_ls\_delays$	$\texttt{ls\_delays}[\texttt{last}(\texttt{ls\_first\_index})]' = d$
$update\_avg(avg, first\_index, delays, SIZE)$	$\texttt{avg}' = compute\_avg(\texttt{avg}, \texttt{delays}, \texttt{first\_index}, \texttt{SIZE}) \wedge \texttt{first\_index}' = (\texttt{first\_index} + 1)\ mod\ \texttt{SIZE} \wedge \texttt{delays}'[\texttt{last}(\texttt{first\_index}, \texttt{SIZE})] = 0$
$near(C)$	$\texttt{lsdiff} < \texttt{DECISION\_THR} \wedge \texttt{ls\_avg} \in]\texttt{HS\_DELAY} - 1 + 2 * C, \texttt{HS\_DELAY} + 1 + 2 * C[$
$dec(C)$	$\texttt{ls\_decision\_state} = 1 \wedge \texttt{ls\_decision} = C$

Table 8.2: Pump Constraits

Label :	$FunctionDescription$
$prob\_delay\_bin(\texttt{avg}, i)$	$\binom{\texttt{DELTA}}{i} * \left(\frac{\texttt{avg}}{\texttt{DELTA}}\right)^i * \left(\frac{\texttt{avg}}{\texttt{DELTA}}\right)^{\texttt{DELTA}-i}$
$last(first\_index, SIZE)$	$(\texttt{first\_index} + \texttt{SIZE} - 1)\ mod\ \texttt{SIZE}$
$compute\_avg(\texttt{avg}, \texttt{delays}, \texttt{first\_index}, \texttt{SIZE})$	$avg + \frac{delays[last(first\_index)] - delays[first\_index]}{\texttt{SIZE}}$

Table 8.3: Pump Functions

The initial condition *init* defines the initial configuration for our model. In this configuration variables assume the values given in Table 8.1.

The PS has a transition from state q_0 to state q_1 for every $i \in [0, \texttt{DELTA}]$ with a formula d'$= i$ and with a binomial probability distribution. In particular, the parameter d is computed by the function $prob\_delay\_bin(avg, i)$ (see Table 8.3) which returns the probability that the pump ack time is i when the pump moving average is avg. This function implements a binomial distribution with average value avg on the interval `[0,DELTA]`.

Let us consider the PS NRL_pump.

From state q_1, if no decision has been taken yet (when `ls_decision`=0) the system can reach state q_2, otherwise it goes to the stop state q_s.

Transition from state q_2 to state q_3 defines the transition relation for the pump buffer. We have three cases: 1) `hs_wait` and `nrl_ack` are ≤ 0 (pump received ack from the high system and sent ack to the low one), the pump can send a message to the high system and receive a message from the low one; 2) only `hs_wait` is ≤ 0 (pump received ack from the high system), the pump can only send; 3) only `nrl_ack` is ≤ 0 (pump sent an ack to the low system), the pump can only receive. These cases are modeled with the constraints *send_low* and *receive_high* in Table 8.2.

Transition from state q_3 to state q_4 defines the transitions for the pump-to-low-system ack timer (`nrl_ack`). When it reaches 0, the low system gets an ack from the pump. If the pump buffer is not full, the low system sends a new message, and the pump picks the value `d` as the delay to ack to such message.

Transition from state q_4 to state q_5 updates the moving average of the delays from the high system ack to the pump. When the pump waits for the ack (`hs_wait` > 0), it updates the value of the last ack delay (see *update_nrl_delays* in Table 8.2). When the `hs_wait` timer expires ($-1 \leq$ `hs_wait` ≤ 0), the pump updates the new average for the acks delays and its auxiliary variables (see $update\_avg(avg, first\_index, delays, SIZE)$ in Table 8.2).

Transitions from state q_5 to state q_6 and the q_7 define the low system estimation of the high system ack delay. Again, `d` is the value of the last ack delay. Initially, the low system updates its information about the last ack delay received, as done by the pump in the previous transition (see again *update_ls_delays* and $update\_avg(avg, first\_index, delays, SIZE)$ in Table 8.2). If the difference between the new and old pump-to-low-system moving average (`lsdiff`) is below `DECISION_THR`, the pump-to-low-system moving average is stable enough and the low system decides that the high system sent either 0, if `HS_DELAY - 1.0` $<$ `ls_avg` $<$ `HS_DELAY` $+ 1.0$, or 1, if `HS_DELAY` $+ 1.0 <$ `ls_avg` $<$ `HS_DELAY` 3.0 (see $near(C)$ in Table 8.2). So, the low system may or not take a decision that, in turn, may or not be correct (see $dec(C)$ in Table 8.2).

Transition from state q_7 to state q_0 defines the transitions for the high-system-to-pump ack timer (`hs_wait`). As we have seen, the initial value is `HS_DELAY`, since we are in the case $b = 0$.

8.4 The FHP-Murφ Verifier

FHP-Murφ (*Finite Horizon Probabilistic Murφ*) [121, 122, 27] is a modified version of the Murφ verifier [45, 116]. FHP-Murφ allows us to define *Finite State/Discrete Time Markov Chains* and to automatically verify that the probability of reaching a given error state in *at most k* steps is below a given *threshold*.

Translations of PSs into FHP-Murφ specifications are almost immediate.

8.4.1 FHP-Murφ model of the NRL Pump

We give an immediate translation of the NRL_pump in Figure 8.4 into a specification for the FHP-Murφ model checker.

The system representing the pump is specified by using the set of constants and variables shown in Figure 8.5 and Figure 8.6 (see also Table 8.1).

Global variables define the state of our model and, hence, the number of bytes needed to represent it (76 in our case).

FHP-Murφ allows definition of functions/procedures. Parameters are passed by reference. Function and procedures can modify only those declared "var".

Procedure `main` (Figure 8.8) triggers the next state computation of all subsystems. Parameter `d`, which is passed to `obs` and `nrlpump_ack`, is computed in Figure 8.9, where `prob_delay_bin(m,d)` returns the probability that the pump ack time is `d` when the pump moving average is `m`. As we

```
const   -- constant declarations (i.e. system parameters)
BUFFER_SIZE :    5;   -- size of pump buffer
NRL_WINDOW_SIZE: 5;   -- size of nrl pump sliding window
LS_WINDOW_SIZE:  5;   -- size of low system sliding window
INIT_NRL_AVG :   4.0; -- init. value of high-sys-to-pump moving average
INIT_LS_AVG :    0.0; -- init. value of pump-to-low-syst moving average
DELTA :          10;  -- maximal pump-to-low-system ack delay
HS_DELAY :       4.0; -- H_0 = HS_DELAY; H_1 = HS_DELAY + 2.0
DECISION_THR :   1.0; -- decision threshold
type  -- type definitions (i.e. data structures)
real_type : real(6,99); -- 6 decimal digits, |mantissa| <= 99
BufSizeType : 0 .. BUFFER_SIZE; -- interval
NrlWindow : 0 .. (NRL_WINDOW_SIZE - 1); -- interval
LSWindow : 0 .. (LS_WINDOW_SIZE - 1); -- interval
AckDelay : 0 .. DELTA;  -- range of pump-to-low-system ack delay
```

Figure 8.5: Constants (system parameters) and types (data structures)

```
var -- declarations of global variables (i.e. model state variables)
b : BufSizeType;        -- number of msgs in pump buffer
nrl_avg : real_type;     -- high-system-to-pump moving average
ls_avg : real_type;    -- pump-to-low-system moving average
nrl_delays : array[NrlWindow] of real_type;
      -- pump sliding window: last high-system-to-pump ack delays
ls_delays : array[LSWindow] of real_type;
      -- low system sliding window: last pump-to-low-system ack delays
nrl_first_index : NrlWindow;     -- cursor nrl sliding window
ls_first_index : LSWindow; -- cursor low system sliding window
nrl_ack : real_type;   -- pump-to-low-system ack timer
hs_wait : real_type;   -- high-system-to-pump ack timer
ls_decision: 0 .. 1; -- 0: hs sent 0, 1: hs sent 1
ls_decision_state: 0 .. 2; -- 0: to be decided;
     -- 1: decision taken; 2: decision taken and state reset done
```

Figure 8.6: Global variables (state variables)

```
-- reset all, but obs_decision_state
procedure goto_stop_state(); begin  init();
ls_decision_state := 2; end;     -- decision taken and reset done
```

Figure 8.7: Procedure `goto_stop_state` resets system states

have seen (see Table 8.3), this function implements a binomial distribution with average value `m` on the interval `[0,DELTA]`.

Figure 8.10 shows the definition of the initial state and of the probabilistic transition rules for our model of the NRL Pump.

```
procedure main(d : AckDelay);
Var b_new : BufSizeType;      Var nrl_ack_new : real_type;
Var hs_wait_new : real_type;  Var avg_new : real_type;
begin  -- decision taken and state reset already done
if (ls_decision_state = 2) then return; endif;
if (ls_decision_state = 0) then -- decision not taken yet
 b_new := b;                  nrl_ack_new := nrl_ack;
 hs_wait_new := hs_wait;      avg_new := nrl_avg;
 buffer(b_new);               nrlpump_ack(nrl_ack_new, d);
 obs(d);                      nrlpump(avg_new);
 hs(hs_wait_new);             b := b_new;
 nrl_ack := nrl_ack_new;      hs_wait := hs_wait_new;
 nrl_avg := avg_new;
else -- decision taken but state reset to be done
 goto_stop_state();  endif;  end;
```

Figure 8.8: Procedure `main` updates system state

```
function binomial(n : real_type; k : real_type) : real_type;
var result : real_type;      var nn, kk : real_type;
begin   result := 1; nn := n; kk := k;
while (kk >= 1) do
   result := result*(nn/kk);  nn := nn - 1; kk := kk - 1;  endwhile;
return (result);  end;
function prob_delay_bin(m : real_type; d : AckDelay) : real_type;
var p : real_type;
begin   p := m/DELTA;
return ( binomial(DELTA, d)*pow(p, d)*pow(1 - p, DELTA - d) );  end;
```

Figure 8.9: Function `prob_delay_bin()` updates high sys ack timer

Procedure `init` (Figure 8.11) defines the initial state for our model (condition *init* and state q_0 in Figure 8.4).

Procedure `main` and `init`, together with the `startstate` implement the transitions from state q_0 to state q_1 and q_2 in Figure 8.4.

Procedure `buffer` (Figure 8.12) implements the transition from state q_2 to state q_3.

Procedure `nrlpump_ack` (Figure 8.13) implements the transition from state q_3 to state q_4 of the PS in Figure 8.4

Procedure `nrlpump` (Figure 8.14) implements the transition from state q_4 to state q_5.

Procedure `obs` (Figure 8.15) implements the transition from state q_5 to state q_6 and q_7.

Procedure `hs` (Figure 8.16) implements the transition from state q_7 to state q_0.

Procedure `goto_stop_state` (Figure 8.7) resets the NRL Pump (by calling `init`) after the decision has been made by the low system. This procedure has nothing to do with the pump working. It is only used to easy our covert channel measures (see transition to state q_s in Figure 8.4)

```
startstate "initstate" init(); end;  -- define init state
ruleset d : AckDelay do   -- define transition rules
 rule "time step" prob_delay_bin(nrlavg, d) ==> begin main(d) end; end;
```

Figure 8.10: Startstate and transition rules for NRL pump model

```
procedure init();  begin
 b := 0; nrl_avg := INIT_NRL_AVG;  ls_avg := INIT_LS_AVG;
 for i : NrlWindow do nrl_delays[i] := INIT_NRL_AVG; end;
 for i : LSWindow do ls_delays[i] := INIT_LS_AVG; end;
 nrl_first_index := 0; ls_first_index := 0;  nrl_ack := 0;
 hs_wait := 0;  ls_decision := 0; ls_decision_state := 0;  end;
```

Figure 8.11: Procedure `init` defines the initial state

```
procedure buffer(Var b_new : BufSizeType);
begin  -- send and get at the same time, b does not change
if (((hs_wait <= 0.0)&(b > 0)) & ((nrl_ack <= 0.0)&(b < BUFFER_SIZE)))
then  return; endif;
-- high system gets msg from buffer
if ((hs_wait <= 0.0)&(b > 0)) then b_new := b - 1; return; endif;
-- low system sends msg to buffer
if ((nrl_ack <= 0.0)&(b < BUFFER_SIZE)) then b_new := b+1;return;endif;
end;
```

Figure 8.12: Procedure `buffer` models the pump buffer

```
procedure nrlpump_ack(Var nrl_ack_new : real_type; d : AckDelay);
begin  -- ack timer expired, low system sends new msg, timer takes d
if ((nrl_ack <= 0.0) & (b < BUFFER_SIZE))
   then nrl_ack_new := d; return; endif;
-- ack timer not expired, waiting ack, timer decremented
if (nrl_ack > 0.0) then nrl_ack_new := nrl_ack - 1; return; endif; end;
```

Figure 8.13: Procedure `nrlpump_ack` defines pump-to-low-system ack timer

```
procedure nrlpump(Var avg_new : real_type); var last_index : NrlWindow;
begin
last_index := (nrl_first_index + NRL_WINDOW_SIZE - 1)%NRL_WINDOW_SIZE;
-- high system processing message
if (hs_wait >  0)
then nrl_delays[last_index] := nrl_delays[last_index] + 1.0; endif;
-- high system sends ack to the pump
if ((hs_wait >=  -1) & (hs_wait <= 0.0)) then
avg_new := nrl_avg +
((nrl_delays[last_index]-nrl_delays[nrl_first_index])/NRL_WINDOW_SIZE);
nrl_first_index := (nrl_first_index + 1)%NRL_WINDOW_SIZE;
nrl_delays[(nrl_first_index+NRL_WINDOW_SIZE-1)%NRL_WINDOW_SIZE] := 0;
endif;  end;
```

Figure 8.14: Procedure `nrlpump` updates value of moving average of high syst ack times

```
procedure obs(d : AckDelay);
var ls_last_index : LSWindow;     var ackval : real_type;
var ls_old : real_type;           var lsdiff : real_type;
begin
if ((nrl_ack <= 0.0) & (b < BUFFER_SIZE)) then  ackval := d;
 ls_last_index := (ls_first_index + LS_WINDOW_SIZE - 1)%LS_WINDOW_SIZE;
 ls_delays[ls_last_index] := ackval;    ls_old := ls_avg;
 ls_avg := ls_avg + ((ls_delays[ls_last_index] -
                     ls_delays[ls_first_index])/LS_WINDOW_SIZE);
 ls_first_index := (ls_first_index + 1)%LS_WINDOW_SIZE;
 ls_delays[(ls_first_index + LS_WINDOW_SIZE - 1)%LS_WINDOW_SIZE] := 0;
 -- make decision
 if (ls_decision_state = 0) then -- decision has not been taken yet
  -- make decision only when ls_avg stable (i.e. lsdiff small)
  lsdiff := fabs(ls_avg - ls_old);
   if ((lsdiff < DECISION_THR) & (HS_DELAY - 1.0 < ls_avg) &
       (ls_avg < HS_DELAY + 1.0))
       then  ls_decision := 0; ls_decision_state := 1; return; endif;
   if ((lsdiff < DECISION_THR) & (HS_DELAY + 1.0 < ls_avg) &
       (ls_avg < HS_DELAY + 3.0))
     then ls_decision := 1; ls_decision_state := 1; return; endif;
  endif; endif; end;
```

Figure 8.15: Procedure `obs` computes the low syst estimate of the high syst ack time

FHP-Murφ returns the probability of reaching a state in which a given invariant fails. Invariant "`no_decision_taken`" in Figure 8.17 states that no decision has been taken, and allows us to compute $P_{\text{dec}}(h)$. Invariant "`no-dec_or_right-dec`" in Figure 8.17 states that no decision or the correct decision has been taken, and allows us to compute $P_{\text{wrong}}(h)$.

8.5 Experimental Results

Let us study the probabilities of making a *decision*, the *wrong decision*, or the *right decision* within h time units, denoted $P_{\text{dec}}(h)$, $P_{\text{wrong}}(h)$, and $P_{\text{right}}(h)$ (= $P_{\text{dec}}(h)$ (1-$P_{\text{wrong}}(h)$)). FHP-Murφ returns the probability of reaching a state in which a given invariant fails, allowing us to compute $P_{\text{dec}}(h)$and $P_{\text{wrong}}(h)$.

We studied how $P_{\text{dec}}(h)$, $P_{\text{wrong}}(h)$ and $P_{\text{right}}(h)$ depend on `BUFFER_SIZE`, `NRL_WINDOW_SIZE`, `LS_WINDOW_SIZE`. Figure 8.18 shows $P_{\text{dec}}(h)$ and $P_{\text{wrong}}(h)$, and Figure 8.19 shows $P_{\text{right}}(h)$, as functions of h for some parameter settings. We label `A-B-C-d` (resp. `A-B-C-w`, `A-B-C-r`) the experiments referring to $P_{\text{dec}}(h)$ (resp. $P_{\text{wrong}}(h)$, $P_{\text{right}}(h)$) with settings `BUFFER_SIZE = A`, `NRL_WINDOW_SIZE = B`, `LS_WINDOW_SIZE = C`. (Each of the curves in Figs. 8.18,8.19 required about

```
procedure hs(Var hs_wait_new : real_type);
var last_index : NrlWindow;
begin
-- ack timer not expired, waiting ack, timer decremented
if (hs_wait > 0) then hs_wait_new := hs_wait - 1; return; endif;
-- ack timer expired, high syst receives new msg, timer takes HS_DELAY
if ((hs_wait <= 0.0)&(b > 0)) then hs_wait_new := HS_DELAY;return;endif;
-- ack timer expired, high system waiting for new msg
if ((hs_wait <= 0.0) & (b <= 0)) then hs_wait_new := -2.0; return;endif;
end;  -- hs()
```

Figure 8.16: Procedure `hs` defines high-system-to-pump ack timer

```
invariant "no_decision_taken" 0.0   (ls_decision_state = 0);
invariant "no-dec_or_right-dec" 0.0
(ls_decision_state = 0) | ((ls_decision_state>0) & (ls_decision=0));
```

Figure 8.17: Invariants

2 days of computation on a 2 GHz Pentium PC with 512 MB of RAM.)

Figure 8.18 shows that the low system with probability almost 1 decides correctly the value of the bit sent by the high system within 100 time units. Our time unit is about the time needed to transfer messages from/to the pump. Thus we may reasonably assume that a time unit is about 1ms. Then Figure 8.19 tells us that the high system can send bits to the low system at a rate of about 1 bit every 0.1 seconds, i.e. 10 bits/sec. Hence, for many applications the NRL Pump can be considered *secure*, i.e. the covert channel has such a low capacity that it would take too long to the high system to send interesting messages to the low system. On the other hand, it is not hard to conceive scenarios where also a few bits sent from the high system to the low system are a security threat.

Notice that, for the parameter settings that we have considered, the most important parameter is `LS_WINDOW_SIZE`, i.e., different parameter settings with the same value of `LS_WINDOW_SIZE` yield the same curves in Figs. 8.18, 8.19. Moreover, the larger is `LS_WINDOW_SIZE` the later and more precise is the decision of the low system. This is quite natural since `LS_WINDOW_SIZE` is the width of the moving average used by the low system to estimate the average ack delays from the high system. The more samples we use in such a moving average the better is the estimation, event though the longer is the waiting to get it.

Figure 8.19 shows that the transition from the situation in which no covert channel exists ($P_{\text{right}}(h)$ close to 0) to that in which a covert channel exists ($P_{\text{right}}(h)$ close to 1) is steep. Hence, relatively small changes in system parameters (namely `LS_WINDOW_SIZE`) may have a dramatic effect on security.

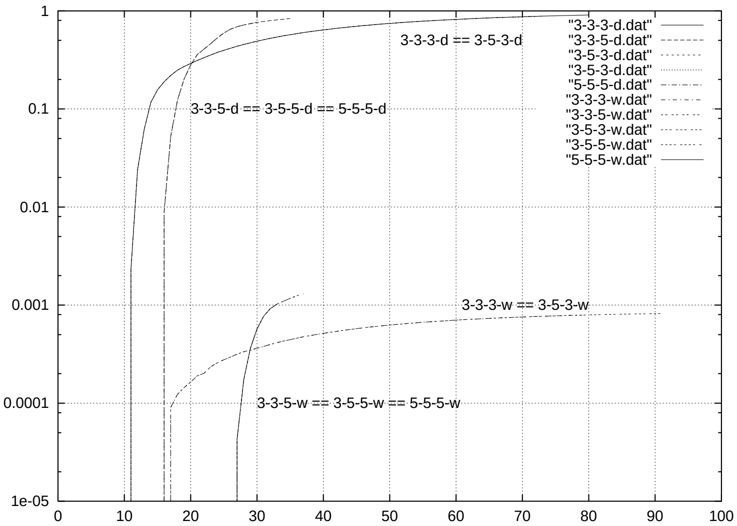

Figure 8.18: Probabilities of taking a decision and a wrong decision within h time units

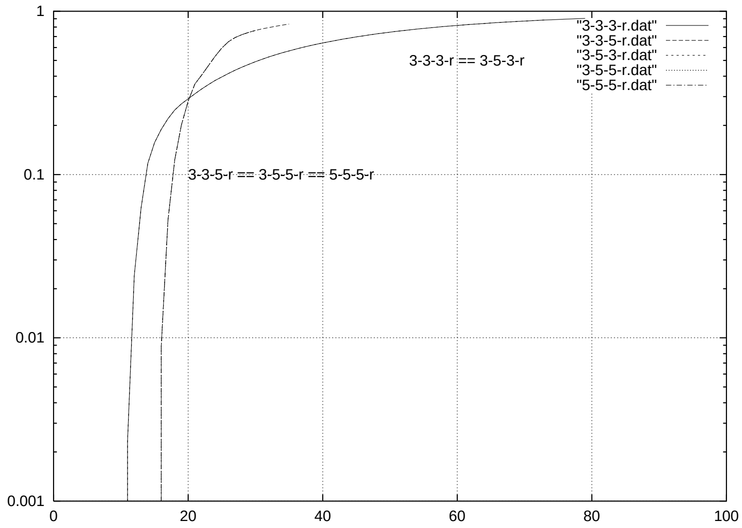

Figure 8.19: Probability of taking the right decision within h time units

Conclusions

Our society is becoming more and more dependent on computer networks. Unsafe behaviors must be avoided, and data which are elaborated, transmitted or stored need some form of protection. Cryptographic algorithms have been proposed to protect information which is transmitted on communication networks. However, cryptography is mainly used as a building block of many complex applications where the correctness of the cryptographic algorithm is not a guarantee, per se, of the correctness of the applications.

Only the development and use of formal tools may allow analysts and designers to describe faithfully, to analyze in detail and to prove the correctness of systems. Formal specifications and application of verification techniques may significantly reduce the risk of errors in the development of systems, compared with traditional informal specification and testing.

The application of formal methods to the development of validated systems mainly concentrates on specification languages, with semantics adequate to support implementation and verification, and methods and tools for verifying specifications with respect to properties expressed in suitable formalisms.

Models based on the concept of states and transitions, also called automata, have turned out to be particularly intuitive. States of the automaton represent snapshots of the described system, while transitions represent state changes.

The nondeterministic approximation however makes it impossible to capture the leakage of probabilistic and time information and prevent possible attacks based, for instance, on statistical or timing analysis. For this reason, theoretical foundations of security properties should take probability and time into account.

Approximating Weak Bisimulation

The correctness of a system with respect to a desired behavior is verified by checking whether a structure that models the system satisfies a formula describing that behavior.

The fact that the system cannot reach unwanted states (safety) and that there is no possibility of leakage of information (security and privacy) are among the most interesting properties.

In general, behavioural equivalences may be used to verify a property of a system by assessing the equivalence of the system considered, with a system one knows to enjoy the property. The problem of formalising the notion of confidentiality boils down to that of formalising equivalence of processes. This is a central and difficult question at the heart of computer science to which there is no unique answer. Which notion of equivalence is appropriate depends on the context and application. Consequently, one should not be surprised that the information security community has failed to come up with a consensus on what constitutes confidentiality. However, weak bisimulation equivalence has been successfully used for the analysis of information flow security properties in multilevel systems. Fine enough to capture any unsecure behavior that can give rise to information flow, but not so strict to classify as unsecure behaviours which are instead correct. Decidability of weak bisimulation for a class of Probabilistic Timed Automata has been proved. This result has been used for the analysis of Information Flow security in a context where both time and probability are taken into account.

In systems that model quantitative processes, steps are associated with a given quantity, such as the probability that the step will happen or the resources (e.g. time or cost) needed to perform that step. The standard notion of bisimulation can be adapted to these systems by treating the quantities as labels, but this does not provide a robust relation, since quantities are matched only when they are identical. Processes that differ for a very small probability, for instance, would be considered just as different as processes that perform completely different actions. This is particularly relevant to security systems where specifications can be given as perfect, but impractical when processes may be considered safe if they only differ from the specification with a negligible probability.

To find a more flexible way to differentiate processes, one may consider the notion of metric, which is a function that associates a real number (distance) with a pair of elements.

In [44, 43, 41] metrics are introduced in order to quantify the similarity of the behavior of probabilistic transitions systems that are not strictly bisimilar.

While the notion of weak bisimulation introduced in this thesis is quite strict, a notion of approximate weak bisimulation could be extremely useful when analyzing security aspects of probabilistic systems.

In fact, new aspects in the study of insecure behaviors can be analyzed by modeling the probabilistic behavior of systems. More precisely, within a purely nondeterministic qualitative setting we can just establish whether a system is secure or not, while in a probabilistic model we can also verify the security level of a system by establishing with which probability a certain insecure behavior might arise.

Moreover, many problems solved by using deterministic algorithms turn out to be secure, but require exponential time. The same problems can be solved by resorting to probabilistic algorithms running in polynomial time, but allowing the execution of potential insecure behaviors. In the classical nondeterministic approach to security analysis, such probabilistic algorithms turn out to be insecure even though the probability of executing an insecure behavior is close to 0. As a consequence, a quantitative study of the unwanted behaviors is crucial for the evaluation of the security level of probabilistic systems.

In order to introduce a quantitative measure for insecure behavior (by defining quantitative security properties) and to estimate the probability that a certain insecure behavior arises, one can resort to a quantitative notion of weak bisimulation for deciding whether two systems behave almost in the same way (up to a given threshold).

As a future work, one may think of introducing a notion of approximate weak bisimulation with epsilon-precision within the model of Probabilistic Timed Automata. Such a notion should be able to tolerate fluctuations making the security conditions less restrictive and relating systems that may have largely different possible behaviors under the condition that such behaviors are observable with a negligible probability. This will allow assessing bisimilarity of automata for which the difference between the quantitative behaviors is within a small distance.

A Unified Cryptographic Model

A recent but well-known formal view of cryptography has been introduced by Abadi and Rogaway in [3], which describes formal algebraic expressions on which encryption operates and defines what it means for two encrypted expressions to be equivalent. As a novelty, [3] relates the formal view and the classical computational model of cryptography, by proving as the main result the soundness of the formal world with respect to the computational world.

The treatment of cryptographic operations within formal models is covered by a large and well-established body of literature, but most of these efforts do not consider cryptographic operations in an imperfect cryptography scenario. The classical Dolev-Yao model, which is based on the perfect cryptography assumption and the restricted expressive power of the adversary, favours a convenient application of formal methods that treat cryptographic operations as purely formal. On the basis of such considerations, we aim at relaxing the strict requirements of the Dolev-Yao approach to cryptography. In order to overcome the limitations of such requirements, we may take into account the probability for a polynomial time adversary of attacking with success a message encrypted with a secret key. While in a Dolev-Yao setting such a possibility is simply disregarded (a message encrypted with an unknown key is a black box) in a real scenario an adversary with a suitable knowledge may have a good chance of obtaining useful information from a ciphertext that, from a purely formal standpoint, is considered to be a black box.

In [145, 146] we tried to fill this gap. A technique is shown for verifying whether the privacy of the ciphertexts exchanged during a protocol can be guaranteed at a reasonable level. The model is based on the analysis of the structure of the messages, by considering all possible ways an attacker may guess some keys contained in the messages at his disposal. For this purpose, a simple formal model of cryptography is extended with probabilistic information which represents an estimation of the robustness of the encryption algorithm and of the used key.

This technique can be used to extend any quantitative formal model used for the analysis of probabilistic non-interference. In this setting, one may think at providing a probabilistic formal model capable of estimating the information leakage due to both the information flows which can derive from the protocol (probabilistic or time) behavior and the weaknesses of the encryption algorithms and secret keys.

A step in this direction can be done by enriching a probabilistic and timed model for systems description with notions of cryptographic operations (encryption, decryption, hashing, etc.) in order to deal with cryptographic systems (e.g. by enriching with probabilities the model of SDMTAs introduced in Chapter 6).

Development of Automatic Tools

In the long term, we expect our results to lay the foundations for automated verification techniques of infomation flow security properties. A good starting point could be the implementation of the notions of weak bisimulation and of approximate weak bisimulation.

Bibliography

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. of the 4th ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[2] M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Software Eng.*, 22(1):6–15, 1996.

[3] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proc. of 1st IFIP Int. Conf. on Theoretical Computer Science*, volume 1872 of *Springer LNCS*, pages 3–22, 2000.

[4] R. K. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. *ACM Trans. Database Syst.*, 17:513–560, 1992.

[5] A. Aldini. *Security and Performance Analyses in Distributed Systems*. PhD thesis, University of Bologna, 2003.

[6] A. Aldini, M. Bravetti, and R. Gorrieri. A process-algebraic approach for the analysis of probabilistic non interference. *Journal of Computer Security*, 12:191–245, 2004.

[7] A. Aldini and R. Gorrieri. Security analysis of a probabilistic non-repudiation protocol. In *Proc. of PAPM-PROBMIV*, volume 2399, pages 17–36. Springer LNCS, 2002.

[8] R. Alur, C. Courcoubetis, and D. L. Dill. Verifying automata specifications of probabilistic real-time systems. In *Proc. of REX Workshop*, volume 600, pages 28–44. Springer LNCS, 1992.

[9] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[10] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211:253–273, 1999.

[11] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proc. of STOC'93*, pages 592–601. ACM Press, 1993.

[12] T. Amnell, G. Behrmann, J. Bengtsson, P. R. D'Argenio, A. David, A. Fehnker, T. Hune, B. Jeannet, K. G. Larsen, M. O. Moeller, P. Pettersson, C. Weise, and W. Yi. Uppaal-now, next and future. volume 2067, pages 99–124. Springer LNCS, 2000.

[13] J. C. M. Baeten, J. A. Bergstra, and S. A. Smolka. Axiomatizing probabilistic processes: Acp with generative probabilities. *Information and Computation*, 121:234–255, 1995.

[14] C. Baier. *On Algorithmic Verification methods for Probabilistic Systems*. Habilitation thesis, University of Mannheim, 1998.

[15] C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. In *CAV*, volume 1254, pages 119–130. Springer LNCS, 1997.

[16] C. Baier and M. Kwiatowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[17] R. Barbuti and L. Tesei. A decidable notion of timed non-interference. *Fundam. Inform.*, 54(2-3):137–150, 2003.

[18] D. Beauquier. On probabilistic timed automata. *Theor. Comput. Sci.*, 292(1):65–84, 2003.

[19] D. Bell and L. J. La Padula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-301, MITRE MTR-2997, 1975.

[20] R. E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[21] M. Bhargava and C. Palamidessi. Probabilistic anonymity. In *Proc. of CONCUR'05*, volume 3653 of *Springer LNCS*, pages 171–185, 2005.

[22] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. of Int. Conference on Foundation of Software Technologies and Theoretical Computer Science*, number 1026 in Springer LNCS, pages 499–513, 1995.

[23] H. Bohnenkamp, P. van der Stok, H. Hermanns, and F. Vaandrager. Costoptimization of the ipv4 zeroconf protocol. In *Proc. of DSN'03*, IEEE CS Press, pages 531–540, 2003.

[24] P. Bouyer. Timed automata may cause some troubles. Technical Report BRICS RS-02-35, Aalborg University, 2002.

[25] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. of International Congress on Logic, Methodology and Philosophy Science*, pages 1–11. Stanford university Press, 1960.

[26] M. Burrows, M. Abadi, and R. Needham. A logic for authentication. Technical Report 39, Digital Systems Research Center, 1989.

[27] Cached Murphi. Web site: http://www.dsi.uniroma1.it/∼tronci/cached.murphi.html.

[28] S. Cattani and R. Segala. Decision algorithm for probabilistic bisimulation. In *Proc. of CONCUR'02*, Springer LNCS, pages 371–385, 2002.

[29] K. Cerans. Decidability of bisimulation equivalences for parallel timer processes. In *Proc. of CAV'92*, number 663 in Springer LNCS, pages 302–315, 1992.

[30] D. L. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[31] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proc. of the Workshop on Logic of Programs*, number 131 in Springer LNCS, pages 52–71, 1981.

[32] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking.* MIT Press, 1999.

[33] R. Corin, S. Etalle, P. H. Hartel, and A. Mader. Timed analysis of security protocols. Technical report, Univiversity of Twente, Centre for Telematics and Information Technology, 2005.

[34] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite state probabilistic programs. In *Proc. of FOCS'88*, IEEE CS Press, pages 338–345, 1988.

[35] D. Craigen. Reference manual for the language verdi. Technical Report TR-91-5429-09a, ORA Canada, 1991.

[36] C. J. F. Cremers, S. Mauw, and E. P. de Vink. Defining authentication in a trace model. In *Proc. of FAST'03*, IITT-CNR technical report, pages 131–145, 2003.

[37] R. David and S. H. Son. A secure two phase locking protocol. In *Proc. IEEE Symposium on Reliable Distributed Systems*, IEEE CS Press, pages 126–135, 1993.

[38] C. Daws. Symbolic and parametric model checking of discrete-time markov chains. In *Proc. of ICTAC'04*, number 3407 in Springer LNCS, pages 280–294, 2004.

[39] L. de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford Univeristy, 1997.

[40] R. A. DeMillo, N. A. Lynch, and M. Merritt. Cryptographic protocols. In *Proc. of 14th Annual ACM Symposium on Theory of Computing*, ACM Press, pages 383–400, 1982.

[41] Y. Deng, T. Chothia, C. Palamidessi, and J. Pang. Metrics for action-labelled quantitative transition systems. In *Proc. of QAPL'05*, Elsevier ENTCS, 2005. To appear.

[42] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[43] J. Desharnais, R. Jagadeesan, V. Gupta, and P. Panangaden. Metrics for labelled markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.

[44] J. Desharnis, V. Gupta, R. Jagadeesan, and P. Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proc. of 17th Symposium on Logic in Computer Science*, IEEE CS Press, 2002.

[45] D. L. Dill, A. J. Drexler, A. J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *Proc. IEEE Int. Conf. on Computer Design on VLSI in Computer & Processors*, IEEE CS Press, pages 522–525, 1992.

[46] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 25:198–208, 1983.

[47] D. D'Souza and P. S. Thiagarajan. Product interval automata: a subclass of timed automata. In *Proc. of FSTTCS'99*, number 1738 in Springer LNCS, pages 60–71, 1999.

[48] A. Durante, R. Focardi, and R. Gorrieri. A compiler for analysing cryptographic protocols using non-interference. *ACM Transactions on Software Engineering and Methodology*, 9(4):489–530, 2000.

[49] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. of ICALP'80*, Springer LNCS, pages 169–181, 1980.

[50] N. Evans and S. Schneider. Analysing time dependent security properties in csp using pvs. In *Proc. of Symp. on Research in Computer Security*, number 1895 in Springer LNCS, pages 222–237, 2000.

[51] R. Focardi and R. Gorrieri. A classification of security properties. *Journal of Computer Security*, 3(5):5–33, 1995.

[52] R. Focardi, R. Gorrieri, and F. Martinelli. Information flow analysis in a discrete-time process algebra. In *Proc. of 13th CSFW*, IEEE CS Press, pages 170–184, 2000.

[53] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proc. of ICALP'00*, Springer LNCS, pages 354–372, 2000.

[54] J. A. Goguen and J. Meseguer. Security policy and security models. In *Proc. of IEEE Symp. on Security and Privacy*, IEEE CS Press, pages 11–20, 1982.

[55] R. Gorrieri, E. Locatelli, and F. Martinelli. A simple language for real-time cryptographic protocol analysis. In *Proc. of ESOP'03*, number 2618 in Springer LNCS, pages 114–128, 2003.

[56] G. Graham and P. Denning. Protection - principles and practice. In *Proc. of Spring Joint Computer Conference*, AFIPS Press, page cercare, 1972.

[57] J. W. Gray III. Toward a mathematical foundation for information flow security. *Journal of Computer Security*, 1:255–291, 1992.

[58] J. W. Gray III and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., 1992.

[59] P. R. Halmos. *Measure Theory.* Springer-Verlag, 1950.

[60] H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* Elsevier, 1994.

[61] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[62] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[63] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[64] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5:356–380, 1983.

[65] V. Hartonas-Garmhausen, S. Campos, and E. Clarke. Probverus: Probabilistic symbolic model checking. In *Proc. of ARTS'99*, number 1601 in Springer LNCS, pages 96–110, 1999.

[66] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *Proc. of CAV'97*, number 1254 in Springer LNCS, pages 460–463, 1997.

[67] T. A. Henzinger, P. W. Kopke, and H. Wong Toi. The expressive power of clocks. In *Proc. of ICALP'95*, number 944 in Springer LNCS, pages 335–346, 1995.

[68] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[69] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[70] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[71] H. Howard. *Dynamic Programming and Markov Processes.* MIT Press, 1960.

[72] G. E. Hughes and M. J. Creswell. *Introduction to Modal Logic.* Methuen, 1977.

[73] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. *Journal of Log. Algebr. Program.*, 52–53:183–220, 2002.

[74] M. H. Kang, J. Froscher, and I. S. Moskowitz. A framework for mls interoperability. In *Proc. of IEEE High Assurance Systems Engineering Workshop*, pages 198–205. IEEE CS Press, 1996.

[75] M. H. Kang, J. Froscher, and I. S. Moskowitz. An architecture for multilevel security interoperability. In *Proc. of IEEE Computer Security Application Conference*, pages 194–204. IEEE CS Press, 1997.

[76] M. H. Kang and I. S. Moskowitz. A pump for rapid, reliable, secure communication. In *Proc. of ACM Conference on Computer and Communication Security*, pages 119–129. ACM Press, 1993.

[77] M. H. Kang, I. S. Moskowitz, and D. Lee. A network pump. *IEEE Trans. Software Eng.*, 22(5):329–338, 1996.

[78] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. D. Van Nostrand Company, Inc., 1966.

[79] R. A. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.

[80] S. R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. of 14th Annual ACM Symp. on Theory of Computing*, pages 267–281. ACM Press, 1982.

[81] D. Craigen S. Kromodimoeljo, I. Meisels, W. Pase, and M. Saaltink. Eves: An overview. In *Proc. of VDM'91*, number 551 in Springer LNCS, pages 389–405, 1991.

[82] S. Kromodimoeljo, B. Pase, M. Saaltink, D. Craigen, and I. Meisels. The eves system. In *Proc. of Functional Programming, Concurrency, Simulation and Automated Reasoning*, number 693 in Springer LNCS, pages 349–373, 1993.

[83] M. Kwiatkowska. Model checking for probability and time: From theory to practice. In *Proc. of LICS'03*, pages 351–360. IEEE CS Press, 2003.

[84] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. In *Proc. of Tools'02*, number 2324 in Springer LNCS, pages 200–204, 2002.

[85] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theor. Comput. Sci.*, 282:101–150, 2002.

[86] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the ieee 1394 firewire root contention protocol. *Formal Asp. Comput.*, 14(3):295–318, 2003.

[87] M. Kwiatkowska, G. Norman, and J. Sproston. Symbolic model checking of probabilistic timed automata using backwards reachability. Technical Report CSR-03-10, University of Birmingham, 2003.

[88] B. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[89] B. Lampson. Protection. *ACM Operating Systems Review*, 8(1), 1974.

[90] R. Lanotte and D. Beauquier. A decidable probability logic for timed probabilistic systems. Technical Report cs.LO/0411100, CoRR, 2004.

[91] R. Lanotte, A. Maggiolo-Schettini, and A. Peron. Timed cooperating automata. *Fundamenta Informaticae*, 43:153–173, 2000.

[92] R. Lanotte, A. Maggiolo-Schettini, S. Tini, and A. Troina E. Tronci. Automatic analysis of the nrl pump. *Elsevier ENTCS*, 99:245–266, 2004.

[93] R. Lanotte, A. Maggiolo-Schettini, S. Tini, and A. Troina E. Tronci. Automatic covert channel analysis of a multilevel secure component. In *Proc. of ICICS'05*, Springer LNCS, 2004.

[94] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Weak bisimulation for probabilistic timed automata and applications to security. In *Proc. of SEFM'03*, pages 34–43. IEEE CS Press, 2003.

[95] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Decidability results for parametric probabilistic transition systems with an application to security. In *Proc. of SEFM'04*, pages 114–121, 2004.

[96] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Information flow analysis for probabilistic timed automata. In *Proc. of FAST'04*, volume 173 of *Springer IFIP*, pages 13–27, 2004.

[97] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Automatic analysis of a non-repudiation protocol. *Electr. Notes Theor. Comput. Sci.*, 112:113–129, 2005.

[98] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. A classification of time and/or probability dependent security properties. In *Proc. of QAPL'05*, Elsevier ENTCS, 2005. to appear.

[99] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Timed automata with data structures for distributed system design and analysis. In *Proc. of SEFM'05*, IEEE CS Press, pages 44–53, 2005.

[100] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Weak bisimulation for probabilistic timed automata. Draft at http://www.di.unipi.it/ lanotte/pub.html, 2005.

[101] F. Laroussinie, K. G. Larsen, and C. Weise. From timed automata to logic - and back. In *Proc. of MFCS'95*, number 969 in Springer LNCS, pages 27–41, 1995.

[102] G. Lowe. A hierarchy of authentication specifications. In *Proc. of 10th CSFW*, pages 31–44. IEEE CS Press, 1997.

[103] R. Mayr, P. Abdulla, and N. Ben Henda. Verifying infinite markov chains with a finite attractor or the global coarseness property. In *Proc. of the 20th IEEE Symp. on Logic in Computer Science*, pages 127–136. IEEE Computer Society Press, 2005.

[104] D. McCullough. Noninterference and the composability of security properties. In *Proc. of Symp. on Research in Security and Privacy*, pages 177–186. IEEE CS Press, 1988.

[105] J. McLean. *Security Models*, pages 1–19. Encyclopedia of Software Engineering. Wiley Press, 1994.

[106] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966.

[107] C. Meadows. What makes a cryptographic protocol secure? the evolution of requirements specification in formal cryptographic protocol analysis. In *Proc. of ESOP'03*, number 2618 in Springer LNCS, pages 10–21, 2003.

[108] I. Melatti. *Explicit Algorithms for Probabilistic Model Checking*. PhD thesis, Univeristà degli Studi Dell'Aquila, 2005.

[109] J. K. Millen, S. C. Clark, and S. B. Freedman. The interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, 13(2):274–288, 1987.

[110] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer–Verlag, 1980.

[111] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[112] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murφ. In *Proc. IEEE Symposium on Security and Privacy*, pages 141–151. IEEE CS Press, 1997.

[113] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of ssl 3.0. In *Proc. of USENIX Security Symposium*, 1998.

[114] M. H. Kang A. P. Moore and I. S. Moskowitz. Design and assurance strategy for the nrl pump. *IEEE Computer*, 31(4):56–64, 1998.

[115] I. S. Moskowitz and A. R. Miller. The channel capacity of a certain noisy timing channel. *IEEE Trans. Information Theory*, 38(4):1339–1343, 1992.

[116] Murphi. Web site: http://sprout.stanford.edu/dill/murphi.html.

[117] M. Napoli, M. Parente, and A. Peron. Specification and verification of protocols with time constraints. *Electronic Notes in Theoretical Computer Science*, 99:205–227, 1992.

[118] Y. Roggeman O. Markowitch. Probabilistic non-repudiation without trusted third party. In *Proc. of 2nd Conf. on Security in Communication Network*, 1999.

[119] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.

[120] L. C. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.

[121] G. Della Penna, B. Intriglia, I. Melatti, E. Tronci, and M.V. Zilli. Finite horizon analysis of markov chains with murphy verifier. In *Proc. of CHARME'03*, number 2860 in Springer LNCS, pages 394–409, 2003.

[122] G. Della Penna, B. Intriglia, I. Melatti, E. Tronci, and M.V. Zilli. Finite horizon analysis of stochastic systems with the murphi verifier. In *Proc. of ICTCS'03*, number 2841 in Springer LNCS, pages 58–71, 2003.

[123] A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. *Journal of Computer Security*, 12:37–82, 2004.

[124] A. Pnueli. The temporal logic of programs. In *Proc. of IEEE Symposium on Foundation of Computer Science*, pages 46–57. IEEE CS Press, 1977.

[125] PRISM Model Checker. Web site: http://www.cs.bham.ac.uk/dxp/prism.

[126] J. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Proc. of the 5th Colloquium on International Symposium on Programming*, Springer LNCS, pages 337–351, 1982.

[127] M. O. Rabin. Probabilistic automata. *Information and Computation*, 6:230–245, 1963.

[128] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[129] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, 1998.

[130] W. Reisig. *Petri nets: an introduction.* Springer-Verlag, 1985.

[131] M. K. Reiter and A. D. Rubin. Crowds: anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[132] Reliable Software Technologies, 21515 Ridgetop Circle, Suite 250, Sterling, Virginia 20166. *WhiteBox DeepCover: User Reference Manual*, 1996.

[133] J. Renegard. On the computational complexity and geometry of the first-order theory of the reals, part i: Introduction. preliminaries. the geometry of semi-algebraic sets. the decision problem for the existential theory of the reals. *Journal of Symb. Comput*, 13:255–300, 1992.

[134] J. Rumbaugh, I. Jacobson, and G. Booch, editors. *The Unified Modeling Language reference manual*. Addison-Wesley, 1999.

[135] P. Ryan and S. Schneider. Process algebra and non-interference. In *Proc. of 12th CSFW*, pages 214–227, 1999.

[136] P. Ryan and S. Schneider. *Modeling and analyzing security protocols: the CSP approach*. Addison-Wesley, 2001.

[137] S. Schneider. Security properties and csp. In *Proc. of IEEE Symposium on Security and Privacy*, pages 174–187. IEEE CS Press, 1996.

[138] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, Laboratory for Computer Science, 1995.

[139] S. H. Son, R. Mukkamala, and R. David. Integrating security and real-time requirements using covert channel capacity. *IEEE Trans. Knowl. Data Eng.*, 12(6):865–879, 2000.

[140] I. Stewart. *Galois Theory*. Chapman and Hall, 1989.

[141] M. Stoelinga. *Alea jacta est: verification of probabilistic, real-time and parametric systems*. PhD thesis, University of Nijmegen, the Netherlands, 2002.

[142] D. Sutherland. A model of information. In *Proc. of 9th National Computer Security Conference*. National Bureau of Standards/National Computer Security Center, 1986.

[143] T. Tanimoto, A. Nakata, H. Hashimoto, and T. Higashino. Double depth first search based parametric analysis for parametric time-interval automata. *IEICE Transaction on Fundamentals*, 11:3007–3021, 1998.

[144] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Second Edition, 1951.

[145] A. Troina, A. Aldini, and R. Gorrieri. Approximating imperfect cryptography in a formal model. *Elsevier ENTCS*, 99:183–203, 2004.

[146] A. Troina, A. Aldini, and R. Gorrieri. Towards a formal treatment of secrecy against computational adversaries. In *Proc. of GC'04*, volume 3267 of *Springer LNCS*, pages 77–92, 2005.

[147] M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. of FOCS'85*, pages 327–338. IEEE CS Press, 1985.

[148] D. Williams. *Probability with Martingales*. Cambridge University press, 1991.

[149] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.