Alea jacta est
Verification of probabilistic, real–time and parametric systems

een wetenschappelijke proeve op het gebied
van de Natuurwetenschappen, Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor
aan de Katholieke Universiteit Nijmegen,
volgens besluit van het College van Decanen
in het openbaar te verdedigen op
maandag 22 april 2002
des namiddags om 3:30 precies
door

Mariëlle Ida Antoinette Stoelinga

geboren op 11 augustus 1972 te Eindhoven

Promotor:
    Prof. dr. F. W. Vaandrager

Manuscriptcommissie:
    Prof. dr. C. Baier (Universität Bonn)
    Dr. J.-P. Katoen (Universiteit Twente)
    Prof. dr. M. Z. Kwiatkowska (University of Birmingham)
    Prof. dr. N. A. Lynch (Massachusetts Institute of Technology)
    Prof. dr. R. Segala (Università di Verona)

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

ISBN 90-9015706-9
IPA Dissertation Series 2002–06

Typeset by LaTeX in Times Roman
Printed by Print Partners Ipskamp, Enschede

Preface

Alea jacta est. With the printing of this thesis, almost five years of research end. This was a period of hard work, of astonishing elegance and beauty present in concurrency theory, with a great deal of fun and, sometimes, also doubts as to whether my results were interesting enough for publication. Many people have contributed to my life or to my thesis in various ways. Below, I mention some of them.

Although it is for some reason *not done* in the Dutch graduation tradition, I would first of all like to thank my promotor Frits Vaandrager. He directed my research, provided an enormous amount of valuable ideas, taught me how to write papers and, no matter how busy he was, he carefully read all my writings. Once, he returned his comments on a draft version with many stains of desert sand, as he had been reading it during his vacation in Egypt. His influence can be found on each page of this thesis.

Frits's deep passion for research, his profound knowledge and broad overview of the field, his attitude towards theoretical and more practically oriented research and his capability of never meeting a deadline have made a deep impression on me. I hope, and believe, that I have learned something in each of these points.

Furthermore, this thesis would not have been the same without the contributions of various co–authors, with whom I worked together on several papers. Apart from Frits, these are Christel Baier, Thomas Hune, Judi Romijn and David Simons. I am happy to thank them all for the enjoyable collaborations, from which I learned a lot.

My research also profited a lot from the Promise meetings. Promise, standing for *pro*babilistic *m*ethods *i*n *s*oftware *e*ngineering, is a group of people in the Netherlands working on probabilistic systems and provided an excellent forum for informal presentations and discussions. It enabled me to quickly learn from related work, exchange informal sketches of ideas and to present early work. Our meetings often ended in a pub, where we drank a couple of beers and used napkins or beer mats to sketch our latest ideas on probabilistic bisimulation relations. Hence, many thanks go to Suzana Andova, Erik de Vink, Pedro D'Argenio, Holger Hermanns, Joost–Pieter Katoen, Jerry den Hartog, Jeroen Voeten, Perry de Groot, Kees Middelburg, Jos Baeten and Ed Brinksma, for many fruitful and enjoyable meetings.

I am also very grateful to everyone who commented on earlier versions of this thesis. I would like to thank the members of the reading committee, Christel Baier, Joost–Pieter Katoen, Marta Kwiatkowska, Nancy Lynch and Roberto Segala for their time and for their valuable comments on the manuscript. Furthermore, I would like to thank Jozef Hooman, Holger Hermanns, Erik Poll and Ullrich Hannemann for their useful feedback on various draft chapters of this thesis.

In terms of moral support, I have to acknowledge Bart Jacobs. He is the PhD coordinator of the computer science department. Bart makes the lives of PhD students in Nijmegen a lot

easier and helped me with the final struggle of completing this thesis.

I owe very much to my former Master's thesis supervisor Erik Barendsen. I have been teaching with him for many years, first as a student–assistant, later as a colleague. If ever I gave a good course or talk, then this is because Erik taught me the basics of it.

I would like to thank Scott Smolka and the people in his group for their hospitality during my visit in the beginning of 2000. The experience of doing research in the USA has been extremely valuable for me. It strengthened my decision to go abroad for my post–doc.

Special thanks go to MartijnO. Without his continuous friendship and support, I would probably not even have started on a PhD project at all. Thanks go also to my former office mate Ansgar Fehnker. During four years, we did not only share the office, but also the latest gossips about the Dutch royal family, numerous stroopwafels and other goodies and the experience of starting, making progress on and finishing a PhD project.

I cannot overemphasize the role of my colleagues. The ITT group in Nijmegen provided an excellent working atmosphere, not only for doing excellent research, but also in terms of support, fun and friendship. I would like to thank Marieke Huisman, Judi Romijn, Joachim van den Berg, Erik Poll, Ansgar Fehnker, Angelika Mader, Harco Kuppens, Martijn Oostdijk, Ulrich Hannemann, Hanno Wupper, Jozef Hooman, Hans Meijer, Theo Schouten, Engelbert Hubbers, Cees–Bart Breunesse, Adriaan de Groot, Martijn Warnier, Miaomiao Zhang, Serena Fregonese, Bart Jacobs, Frits Vaandrager, Mirèse Willems, David Griffioen, Marco Devillers, Martijn Warnier and Jesse Hughes for the great time I had working in Nijmegen and the many enjoyable lunches and coffee breaks we had together. 'De harde ITT kern' joined in many marvelous dinners, movies and skating events. With their habit of poking fun at everything, all the time, and their numerous stupid jokes, these people kept me from taking research too seriously; I do not remember how many times I had to run out of a pub, because I had to laugh so much that I could not swallow my drink anymore. Furthermore, they helped me with making the final, long distance arrangements for this thesis and the defense. Thanks go also to the mensa partners Randy, Jeroen, Jeroen, Jasper and Milad (and various people mentioned above) for many efficient, but medium quality meals in the university restaurant. (I think that about 50% of my body was made out of mensa food molecules at the time I submitted the first version of this thesis.)

I would like to thank Luca de Alfaro for persuading me to come to Santa Cruz and Andrea di Blas, Annie Lorrie Anderson, Heather Lee and Eileen Flynn for helping me in settling here.

I want to thank my (almost) parents in law, Ans and Ben van Rossum. While Peter and me were working on our theses, they completely painted and renovated our new house.

My parents, Otto and Netty, and my brother and sister, Christophe and Josine, deserve endless thanks for their unconditional love and support, in spite of me being away from home far too often. With all of them having a background in engineering, they were always interested in what I was doing.

I reserve the greatest thanks for Peter. For the things that really matter.

Santa Cruz, March 2002
Mariëlle Stoelinga

Contents

CHAPTER 1
Introduction

1.1 Motivation

> Anyone who has never made a mistake has never tried anything new.
> Albert Einstein

We are living in a world where software and hardware systems control the temperature in our rooms, our railroad crossings and our nuclear plants, where computer systems are responsible for transferring our salaries, for monitoring kidney dialysis and for supplying our supermarkets with fresh products. The correct functioning of those systems is clearly of vital importance. However, the construction of software and hardware products carrying out these jobs is often an extremely complex task and various examples evidence the presence of serious errors in computer systems [Pet96]. The consequences of a malfunctioning system may vary from user frustration, for instance in the case of an operating system that crashes, to economic damage, as is illustrated by the hardware bug in the Intel Pentium 5 microprocessor, and life threatening situations, as is demonstrated by the software bug in the Therac–25 machine. The latter caused several mortal victims due to overdoses of X–radiation.

Formal Methods

> But there is another reason for the high repute of mathematics: it is mathematics that offers the exact natural sciences a certain measure of security which, without mathematics, they could not attain.
> Albert Einstein

The field of *formal methods* proposes a development methodology that should help to improve the quality of software and hardware and to prevent errors as above. In this field, one argues that the reliability of computer systems improves significantly when formal methods are given a more prominent place in the design and implementation of these systems. Formal methods refer to a wide variety of techniques, tools and languages for system analysis that have rigorous mathematical foundations. These methods can be utilized throughout the entire software life cycle, that is, from the initial investigation of the customer's wishes, the design, implementation, to the testing and maintenance. This thesis pays attention to the **formal specification** and **formal verification** as means to prevent system malfunctioning.

Good software engineering practice requires that, when building a computer system, one starts from a specification, which expresses the desired system behavior. Since misinterpretation is a common source of errors, it is important that the specification is unambiguous and precise. Many people working in formal methods believe that *formal specifications* are often suitable here, because these fulfill the requirement for precision to its highest form.

Formal specifications are specifications written in formal languages, that is, languages whose meaning has been pinned down mathematically. In industry, however, specifications are often written in natural language (usually English) or semi–formal languages (such as UML [BRJ99]).

Apart from avoiding misinterpretations, the use of formal specification languages has another advantage. When the system has been implemented, formal specifications – unlike specifications written in natural language – allow one to prove mathematically that the implementation indeed meets the properties expressed by the specification. This process is called *formal verification* and if a system has been verified formally, one knows that the system behaves correctly under all circumstances – provided, of course, that the specification accurately expresses the intended system behavior.

Since formal verification is rather time consuming, the current industrial practice is to investigate the system correctness via *testing*. As has already been recognized in the 1960s by E. Dijkstra, testing can only show the presence of errors, not their absence. This is because exhaustive testing is usually not feasible, as it requires a very large number of tests, often infinitely many, to be performed. Thus, even if testing did not reveal any errors, this does not mean that the system is correct, because there remain many cases that have not been tested. Therefore, formal verification is favorable when system correctness is crucial.

However, it is important to realize that verification can never give an absolute guarantee for correctness. In fact, verification is always relative to the correctness of the tools applied in the verification, the compilers used to compile the final program, the hardware the implementation runs on and – very importantly – the extent to which the specification really expresses what one has in mind. Therefore, other techniques to evaluate correctness remain necessary.

Reactive Systems

> You see, wire telegraph is a kind of a very, very long cat. You pull his tail in New York and his head is meowing in Los Angeles. Do you understand this? And radio operates exactly the same way: you send signals here, they receive them there. The only difference is that there is no cat.
>
> Albert Einstein

A lot of effort has been spent and is being spent on the verification of *transformational programs*, [Dij69, Apt81, Hui01]. These are computer programs, such as programs to compute the sinus function or to bookkeep one's finances, that are given some input, then compute for some while and finally output their results. This thesis, in contrast, is concerned with verification techniques for **reactive systems**. Reactive systems are systems that continuously interact with the environment in which they are operating and in principle run perpetually. These systems react to stimuli from their environment and generate stimuli which, in return, influence the environment.

A simple example of a reactive system is an automatically controlled railroad crossing. The system controlling the gates has to check continuously whether a train is arriving and, if so, to close the bars. When the train has left the crossing, the bars have to be reopened (unless another train is approaching). The environment of the controller is formed by the train and the bars. Stimuli here are the signals telling that the train is arriving or leaving and telling that the bars should be raised or lowered. Other examples of reactive systems are telephone switching systems, radar systems, chemical batch plants, steam boilers, etc.

Many aspects play a role in the operation of reactive systems. Depending on the type of application, one or more of the following issues can be relevant. One can study the system's *functionality*, which concerns the qualitative behavior of the system; one can consider the system's *performance characteristics*, which concern its quantitative behavior and efficiency; one can focus on *security*, which is the extent to which the system is resistant to undesirable access by intruders; one may take into account the *availability*, which is concerned with the extent to which a system can fulfill the service requests it gets; one can treat the *fault tolerance* of the system, which is the way the system deals with failing components; one can study the *mobility* of the system, which is the possibility of a system to exchange communication links and to run remote applications, etc.

This thesis focuses in particular on **functional behavior** of reactive systems, although several of the techniques described are also relevant in performance analysis and fault tolerant systems.

Secondly, a system can be described at several levels of detail. Depending on the kind of application, one or more of the following features can be incorporated in the system model. *Data* that is manipulated, stored and communicated in the system, *probabilities* which describe the random choices and stochastic behavior in the system, *continuous* behavior that is exhibited by physical components in the system, *parameters* to represent a whole class of systems with the same structure, etc.

This thesis is about **probabilistic**, **timing** and **parametric** aspects in systems modeling and verification. More specifically, as we will explain in the next section, we focus on **discrete** rather than on continuous probabilistic choice, we use **continuous time** rather than discrete time and the only system parameters we consider are **timing parameters**.

Formal Verification Techniques

> We can't solve problems by using the same kind of thinking we used when we created them.
>
> Albert Einstein

There are many verification techniques available to establish the correctness of a system. *Algebraic methods* show that a system meets its specification by expressing them both as terms in process algebra. Then both terms are equated via algebraic laws, very much like the calculations one makes in algebra or analysis. In a proof via *behavioral relations*, one compares the external behavior of two state transition systems, one representing implementation and the other the specification of a system. *Temporal logics* express the specification by a logical formula with temporal operators, which can be used to verify a state transition system. In *Hoare–style proofs* one reasons about the program text by using axioms and logical inference rules to derive statements about the program.

Since verification tends to be a rather complex task, computer support is indispensable for the verification of most real–life applications. *Manual verification* is only feasible for small systems and, moreover, verification carried out by hand is very error prone; in the words of Wolper [Wol98], "Manual verification is at least as likely to be wrong as the program itself." Tool support exists for all of the techniques mentioned above, by providing either fully automated or semi–automatic techniques for program analysis. *Fully automatic* verification methods are preferable, because these are fast and easy, but are only available for a limited class of systems. Especially *model checking*, which is based on temporal logic, has become popular. Also, several variants of (bi–)simulation relations can be checked automatically.

Semi–automatic verification requires user interaction with the computer tool, where the user provides his or her knowledge of the system and the tool checks whether or not the information provided indeed establishes the desired result. Theorem provers such as PVS and Isabelle allow the user to formalize and prove a mathematical theorem and have turned out to be very successful in proving a programs' correctness.

This dissertation considers **manual** and **fully automatic** techniques. More specifically, we use **model checking** and **proofs via (bi–)simulation**. Moreover, we use a combination of model checking and bisimulation: we use model checkers to construct a bisimulation proof.

This Thesis

Summarizing, this thesis presents several techniques based on model checking and (bi–)simulations to reason about the functional correctness of systems with discrete probabilistic choice, real–time and/or timing parameters. More specifically, we develop several theoretical results for probabilistic simulation and bisimulations, we present a model checking technique for parametric systems and we combine both techniques to analyze the IEEE 1394 Root Contention Protocol. This is an industrial leader election protocol in which timing, probabilistic and parametric aspects play a crucial role and which is therefore a suitable case study to investigate the feasibility of the verification techniques presented in this thesis.

The rest of this introduction is organized as follows. Section 1.2 introduces the system models used in this thesis to analyze reactive systems and explains how timing, parametric and probabilistic aspects can be incorporated in these models. In Section 1.3, we explain the concepts of model checking and (bi–)simulation proofs. Finally, Section 1.4 presents an overview of the results in this thesis and the contents of subsequent chapters.

1.2 Models for Timed, Parametric and Probabilistic Systems

> Everything should be made as simple as possible, but not simpler.
>
> The human mind has first to construct forms, independently, before we can find them in things.
>
> Albert Einstein

Verification is an activity to be carried out at the level of language, not on physical objects in reality. In this respect, verification differs from testing, because testing can be performed at physical systems directly. Since reactive systems intrinsically contain physical components, their verification has to be based on system models. A system model describes the system at an appropriate level of detail. As it is the case with a model of any object, one omits the details which are irrelevant for the aspects of interest, thus keeping the verification feasible. Note that a system model can be an intermediate product created during the design phase of the system.

The field of formal methods has put forward an abundance of formalisms to describe reactive systems. We mention just a few of them. *Process algebras* [Hoa85, Mil80] are an algebraic framework in which process equality is specified by algebraic laws. *Petri nets* [Rei85] are based on state transition systems whose dynamics are described by the passage of tokens. *State charts* [Har87], *Message sequence charts* and *UML* [BRJ99] are graphical languages with a rich syntax. UML subsumes the former two and its precise semantics is under construction. Almost all these formalisms have been extended with probabilistic, timing and/or parametric aspects.

The models used in this thesis are all based on **labeled transition systems**, also called **automata**. These have turned out to be an intuitive, yet powerful framework to describe and analyze reactive systems and they have been extended with probabilities, timing and parameters.

This section introduces the automaton model and its extensions with probability, time and parameters in an informal way. A formal treatment will appear in subsequent chapters. As a running example in this section, we treat a railroad crossing. This example is a benchmark problem in verification and has been defined in [HJL93]. Clearly, the models presented here are simplified in many ways, but are still rich enough to illustrate various aspects in modeling and analysis of reactive systems.

Labeled Transition Systems

> Most of the fundamental ideas of science are essentially simple, and may, as a rule, be expressed in a language comprehensible to everyone.
> Albert Einstein

An *automaton*, or *labeled transition system*, describes an application as a set of *states* and *transitions*. A state represents a snapshot of the system and the transitions represent state changes in the application.

Example 1.2.1 Figure 1.1 shows a simple automaton model of a controller in a railroad system. The task of the controller is to open and close the gates. The controller starts in the state $Idle$, which is denoted by the double circle in the picture. When it receives an $approaches?$ signal, indicating that the train is arriving, the controller moves to the state $Lower\_bars$. We also say that the controller *takes a transition* from $Idle$ to $Lower\_bars$ labeled by $approaches?$. In the state $Lower\_bars$, it sends out a $lower!$ signal to tell the gates to close and moves back to the state $Idle$. Similarly, when the controller receives a $leaves?$ signal, it takes the corresponding transition. This leads to the state $Raise\_bars$, in which the controller sends a $raise!$ to the gates and returns to the state $Idle$. Note that the receipt of a signal (usually called an *action*) is denoted by $a?$ and the sending of it by $a!$.

Furthermore, consider the (very much simplified) models of the train and the gate in Figure 1.2, which do not do much more than sending $approaches!$ and $leaves!$ signals and receiving $raise?$ and $lower?$ messages respectively. We only mention these to illustrate the following. When, for instance, the train sends an $approaches!$ signal, then it is received by the controller immediately. This means that the train and the controller take their $approaches-$transitions simultaneously, i.e. both together and at exactly the same time. In technical terms, we say that the train and the controller *synchronize* on the action $approaches$. They also synchronize on the action $leaves$. Similarly, the controller and the gate synchronize on the actions $lower$ and $raise$.

Timed Systems

> The only reason for time is so that everything doesn't happen at once.

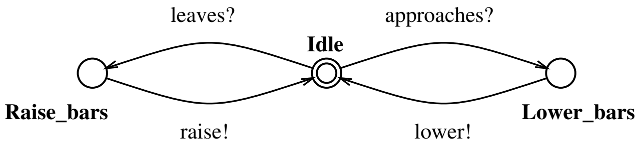


Figure 1.1: Simple automaton model of a railroad controller

Figure 1.2: Train and gate models

Albert Einstein

Timing aspects are crucial in many computer applications. In the railroad example for instance, assume that a train approaches the railroad crossing. Then the bars have to be closed when the train is at the crossing. This means that the closing signal has to be passed to the gate fast enough so that the time needed for the bar to close is smaller than the time needed for the train to reach the crossing. Obviously, small delays can have catastrophic effects.

The necessity of timing aspects has long been recognized and many formal methods have been extended with time. In this dissertation, we use **timed automata** [AD94] and **timed I/O automata** [MMT91], which are both timed versions of labeled transition systems.

With the use of time, the railroad crossing can be modeled more accurately. The following example presents a timed automaton description of the railroad system.

Example 1.2.2 Figure 1.3 depicts a timed automaton model of railroad controller. The states and actions are the same as before, but now *clocks* come into play. Initially, all clocks are zero. Via *state invariants*, clocks can limit the amount of time that a component can stay in a state and via *guards* on transitions, clocks can restrict the times at which a transition can be taken. Finally, clocks can be reset to 0.

Let us explain this by means of the controller in Figure 1.3. When the controller receives a $approaches?$ signal and moves to the state $Lower\_bars$, its clock z is reset. The controller may stay in the state $Lower\_bars$ as long as its invariant $z \leq 1$ holds, that is, for at most one time unit. Furthermore, the system may take the $lower!$ transition, leaving the state, if and only if its guard $z = 1$ is satisfied. This reflects the fact that it takes exactly one time unit between the arrival of the $approaches?$ signal and the sending of the $lower!$ signal.

The intuition behind the gate model (rightmost in Figure 1.4) is similar. When the gate receives a $lower?$ signal, a clock y is reset and the gate moves to the state $Lowering$. In this

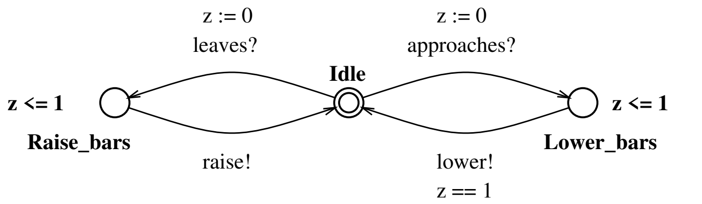


Figure 1.3: Timed automaton model of a railroad controller

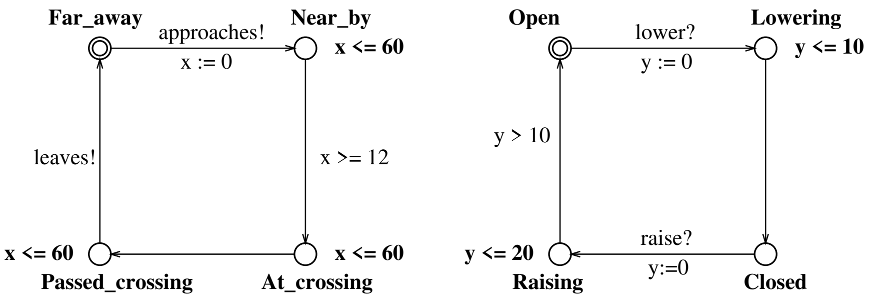


Figure 1.4: Timed automata modeling a train and a gate in a railroad system

state, it can stay for 10 time units at most, reflecting the fact that lowering the gates takes at most 10 time units. The transition going out of the state $Lowering$ is not labeled, which represents an internal step of the gate. Furthermore, in the state $Closed$, the gate waits for a $raise?$ signal to arrive and then it moves to the state $Raising$, where it can stay for strictly more 10 time units but less than 20.

The train is modeled with the same ideas. Note that the clock x is not reset on the transitions leading from $Near\_by$ to $At\_crossing$ and from $At\_crossing$ to $Passed\_crossing$. This means that the time to stay in $Near\_by$, $At\_crossing$ and $Passed\_crossing$ is less than 60 in total, which models the fact that it takes at most 60 time units between the arrival and the departure of the train.

A crucial property of the railroad system is that the bars have to be closed whenever the train is at the crossing. In terms of the model, we require that the gate is in the state $Closed$, whenever the train in the state $At\_crossing$. This property is definitely a part of the specification of any realistic railroad controller system. (Obviously, a complete specification takes many more aspects into account, such as the fact that the bars should eventually reopen when the train has passed.) Actually, one can prove automatically that the property mentioned above does hold for the timed railroad system (but not for the untimed system). Note that we have modeled only one train, so we cannot be sure whether the property also holds if more trains are around. The section on verification techniques below will explain how this can be done.

Parametric Systems

> The eternal mystery of the world is its comprehensibility.
>
> Albert Einstein

Parameters are used to describe a class of systems with the same structure but with different constants. Case studies in verification of a variety of applications show that almost all systems are parameterized in some sense. Parameters in the railroad crossing system are for instance the maximal time for the train to arrive at the crossing and the time for the controller to propagate a signal. Parameters in other systems can be the number of processes in the application, the communication delay, the clock speed, the network topology, etc.Usually, these systems work correctly for some values of the parameters and for others they fail. It is the aim of *parameter synthesis* to derive the exact conditions on the parameters that are needed for correct system operation. These conditions are usually called *parameter constraints* and knowing these constraints provides useful information for building correct systems. For the railroad crossing, the parameter constraints tell whether or not the railroad is still safe if a faster train is introduced.

This thesis restricts itself to timing parameters, as in the railroad example. More precisely, we consider **linear parametric timed automata**. These are timed automata where the guards and invariants may contain linear expressions over the parameters.

Example 1.2.3 Figure 1.5 depicts a parameterized version of the railroad crossing. We have replaced all timing constants by parameters. (Note that this need not be the case; for several applications it can be useful to have both parameters and constants.) Thus, the parameter $\mathrm{max\_lt}$ stands for the maximal time needed to lower the bars. The parameter $\mathrm{min\_rt}$ for the minimal time to raise them and $\mathrm{max\_rt}$ is the maximum time needed for doing so. The parameters $\mathrm{min\_dt}$ and $\mathrm{max\_dt}$ stand for the minimal and maximal time that the train remains at the crossing and st is the time it takes for a signal to be propagated by the controller.

Now, we are interested in the question asking for which values of the parameters the railroad is safe, i.e. for which values the gate is in the state Closed, whenever the train is at the crossing. One can show that this property holds for all parameter values such that

$$\mathrm{st} + \mathrm{max\_lt} < \mathrm{min\_dt}.$$

This is not unexpected, because in order for the gates to close on time, the signal has to be passed to the gate *and* the gate has to be closed *before* the train arrives, even if the train is very fast and the signal and gate very slow. Therefore, the maximal time for propagating the signal and closing the gates has to be smaller than the minimal time needed for the train to arrive at the crossing, which is exactly expressed by the formula above.

Probabilistic Systems

> '"God dobbelt niet!" zei hij.'
> 'Zei Einstein dat?'
> 'Het schijnt.'
> 'Geloofde Einstein dan in god?'
> 'Einstein geloofde in ieder geval niet in het toeval.'
>
> Connie Palmen, *De wetten.*

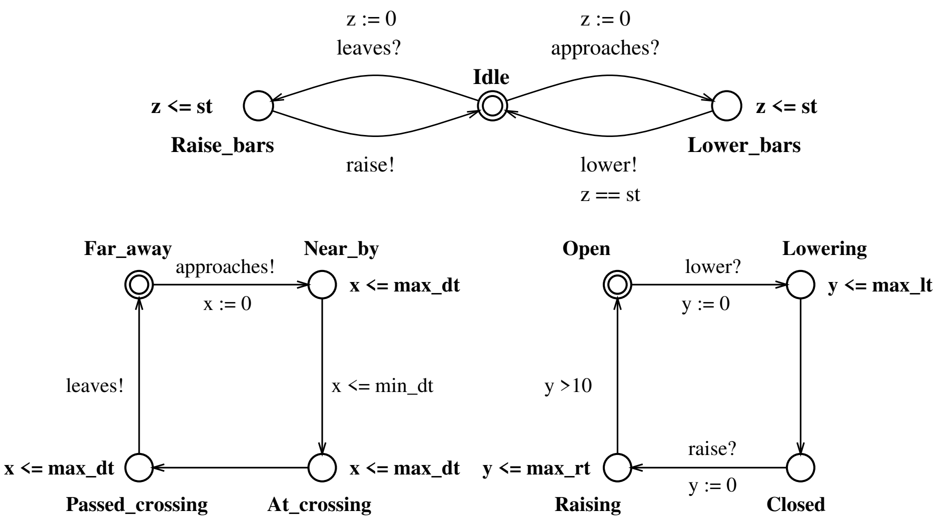


Figure 1.5: Parametric timed automata modeling a railroad system

Probabilities are as old as civilization. Egyptian excavations have brought up several perfectly shaped dice. In the Mahabharata, there is even a demi–god of dice and also the Bible mentions the use of lots to make decisions several times. It might therefore be surprising that it was not until the 16th century that a mathematical probability theory was developed [Hac75]. Pioneering work was done by Leibniz, Huygens and Pascal. Since that time, probability theory turned out to be a powerful means to model and analyze reality and rapidly entered in all fields of science. Today, one finds probabilistic models in physics, sociology, economics, epidemiology, etc.

Also in computer science, there are various applications of randomization[1]. We mention three of these. Probabilities can for be used to model unreliable or unpredictable behavior exhibited by a system. Secondly, stochastic mechanisms play a dominant role in performance evaluation of computer systems. and finally, randomized algorithms make clever use of coin flips or other probability mechanisms to obtain efficient computer programs. Below, we illustrate these applications of probabilities by means of the railroad example.

Note that the first two uses of randomization differ from the latter. When using probabilities to model an unreliable components or the performance of a component, they describe existing phenomena in reality. In randomized algorithms instead, they are deliberately introduced for the purpose of a more efficient solution.

Unreliable behavior

Stochastic mechanisms can be used to model unreliable behavior of a system or its environment. Some components of the system may inevitably fail or be overloaded at some points in time. The challenge is then to design a system with maximal reliability from those components. Usually, failures of system components are governed by a probabilistic mechanism and

[1] In this chapter, we use the words probabilistic, randomized and stochastic as synonyms.

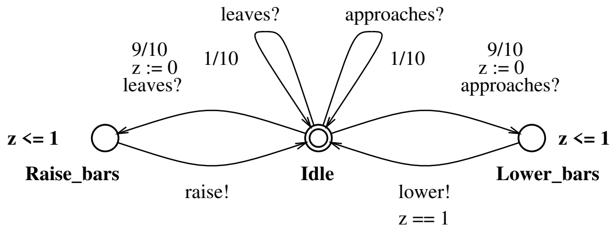


Figure 1.6: Probabilistic model of the controller

probabilistic analysis allows one to derive the probability of a failure of the whole system.

Example 1.2.4 In the railroad example, it is realistic to assume that catching a signal from a moving train is subject to faults. (Note that sending a signal from the controller to the gates, which are part of the same system, is much more reliable.) The system in Figure 1.6 models a controller that does not always receive the $approaches!$ signal sent out by the train correctly: with probability $\frac{1}{10}$ the controller misreceives the signal. This causes the controller not to move to the state $Raise\_bars$, but to remain in the state $Idle$ instead. With probability $\frac{9}{10}$, the signal is received correctly, in which case the system operates as before. As a consequence, the railroad system is not safe anymore and the probability that an error occurs is quite large. Actually, the probability that a signal is lost eventually equals one.

Several solutions exist to deal with this situation – leading to a more complex system design. For instance, the train may repeatedly send $approaches!$ signals. Since the probability that none of the signals arrives correctly is quite small (10^{-n}, if n signals are sent), the system becomes much safer.

Another possible solution would be to have the controller acknowledge the receipt of an $approaches?$ signal. If, after a number of trials, none of the signals has been acknowledged, the train could switch to an emergency scenario, for instance, slowing down.

Performance analysis

Performance analysis aims at evaluating a system in a quantitative sense. Several performance measures can be estimated, for instance the number of telephone calls a telephone system can handle on average, the mean time it takes between two subsequent system failures (e.g. rejected calls due to system overload) etc.Performance analysis is a field in its own right and this thesis focuses more on qualitative rather than quantitative aspects. It is, however, an important topic for present and future research to integrate both aspects in system analysis.

Example 1.2.5 Figure 1.7 depicts a railroad model that could be used in performance evaluation. In each state, the system either waits until a signal is received or it waits for some amount of time t, where t is drawn according to an *exponential probability distribution*. This means that the probability to stay in state s for more than t time units is $e^{-\lambda_s \cdot t}$. Here, λ_s is a parameter of the exponential distribution and, for each state, its value is given by Figure 1.7. The exponential distribution has nice mathematical properties, one of them being that the mean time a process stays in state s equals $\frac{1}{\lambda_s}$. For instance, the parameter of the

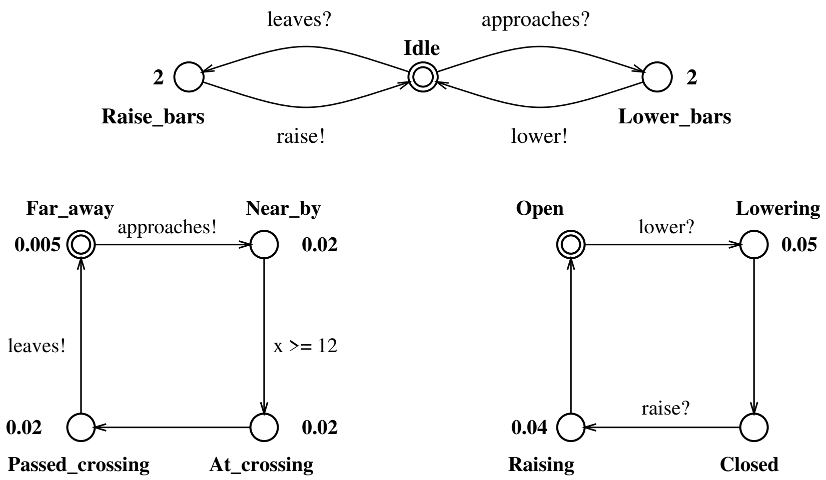


Figure 1.7: Automata modeling railroad system with exponential distributions

state Far_away equals 0.005, which means that, on average, the train stays there for 200 time units.

Note that, in the new model, the train, gate and controller need not meet the timing requirements imposed by the guards and invariants of Figures 1.3 and 1.4 anymore. An important issue, which can be tackled with techniques for performance analysis, is to estimate the probability to violate the safety property mentioned in Example 1.2.2.

Since the behavior of the model in Figure 1.7 is described completely probabilistically, we can compute the exact probabilities and expected times for many events. Apart from the probability to violate the safety property, we can for instance derive the mean time that the gate is in one of the states $Raising$, $Closed$ or $Lowering$, in which no cars can pass the crossing. This performance measure is relevant for the question whether or not traffic jams are caused because of the gate being closed too often.

Randomized algorithms

Thirdly, computer programs (or abstract versions of these, called *algorithms*) may contain random choices. Such programs may roll a dice and make a choice depending on the outcome of the cast. Such programs are called *randomized algorithms*. In a similar way, probabilities can be used in *communication protocols*. The latter are agreements on how several interconnected devices exchange information, for instance, how e–mails are transferred in the Internet. In this thesis, we consider a small part of the IEEE 1394 serial bus protocol. This protocol flips a coin to determine whether it is going to wait a long or a short time before sending a message.

Randomized algorithms are more powerful than non–randomized ones. They perform better in the following three aspects [GBS94]:

1. *Efficiency*, both in time and in space: the probabilistic hashing algorithm described in [GBS94] uses less memory than any known deterministic hashing algorithm. Hashing algorithms are important to store data efficiently.

2. *Symmetry*: it has been shown that there does not exists a solution to the *dining philosophers problem* in which all processes execute the same program, unless randomization is used [LR81]. The problem is relevant in cases of *resource contention,* that is, in order to divide a single resource (printer, processor) among several users (printer jobs, terminals).

3. *Computational power*: Lynch *et al.* [FLP85] have shown that the so–called *Byzantine agreement problem* cannot be solved by any non–randomized computer program. This problem requires several communicating processors to agree on the same value. Some processes may fail, in which case they do not have to agree. It follows from the results in [FLP85] that, no matter which non–randomized program the processors in the network run or how they communicate, agreement on the same value cannot be guaranteed. On the other hand, [GBS94] present a randomized algorithm solving the Byzantine agreement problem.

Probabilistic systems also have their drawbacks: randomized algorithms are harder to analyze than non–randomized ones and one's intuition often fails. This especially holds for probabilistic distributed applications, where a system contains several algorithms running in parallel. Even specialists in this field have proposed solutions that later turned out to be wrong, as has been pointed out for instance in [Seg95b]. This again emphasizes the need for rigorous formal methods in the design and analysis of probabilistic applications.

Some remarks

The probabilistic choice in the applications above can be either discrete or continuous. This thesis restricts itself to **discrete probabilistic choice.** The reason for this is that, among the applications above, we are particularly interested in randomized algorithms and communication protocols. These algorithms and protocols often contain discrete probabilistic choice. Moreover, the benefits of probabilities here are clear and so is the need for formal methods.

Besides probabilistic choice, also nondeterministic choice plays an important role in the verification of reactive systems, no matter whether or not probabilities are around. Hence, we need a model that combines probabilistic choice with nondeterminism. Such models have been developed recently and we consider the probabilistic automaton model from [Seg95b]. The differences and similarities between both types of choices are quite subtle and will be explained elaborately in Chapter 2.

On the other hand, performance analysis mostly studies models that combine discrete and continuous probabilistic choice, but which do not contain nondeterministic choice. Therefore, this thesis hardly addresses performance issues. As said before, it is an important topic for future research to combine techniques from functional analysis and performance analysis.

1.3 Verification Techniques for Reactive Systems

This section briefly explains the ideas behind the two verification techniques studied in this thesis. We explain model checking for timed and parametric systems and (Bi–)simulation relations for timed and probabilistic systems.

Model Checking

When using model checking to verify a system, one describes the system as a (labeled) transition system and expresses its correctness as a logical formula. Then one checks that the transition system satisfies the formula. What made model checking popular is the development of powerful and easy–to–use software tools, called *model checkers*, to perform this check completely automatically. Moreover, if the property does not hold, a model checker comes up with a counter example showing how it is violated. In this way, model checkers support debugging the model.

Model checking tools and techniques exist for a wide variety of specification logics and system models, including timed, probabilistic and parametric transition systems.

Below we show how model checking can be used to analyze the timed and parametric models of the railroad system.

Example 1.3.1 We have already seen that a key correctness property for the railroad crossing is that, if the train is at the crossing, then the gates at the crossing should be closed. This is expressed by the following logical formula Φ.

$$\Phi = \forall\Box(\mathrm{Train}.\mathit{At\_crossing} \implies \mathrm{Gate}.\mathit{Closed}).$$

Here, $\forall\Box$ means that the formula holds in every state of the system and $\mathrm{Train}.\mathit{At\_crossing}$ indicates that the train automaton (see Figure 1.4) is in the state $\mathit{At\_crossing}$. Likewise, $\mathrm{Gate}.\mathit{Closed}$ refers to the state Closed in the automaton Gate (Figure 1.3).

The models presented in Figure 1.4 and 1.3, as well as the property Φ are in the input format of the model checker Uppaal and we used this tool to check the property. Uppaal returned that this property holds for the system and therefore the railroad described by this model is safe. However, if we change the invariant $y \leq 10$ of the location $\mathit{Lowering}$ into $y \leq 11$, then Uppaal will tell us that the property Φ does not hold anymore and it will provide a sequence of steps leading to a situation where the train is in the state $\mathit{At\_crossing}$ and the gate in a state unequal to Closed.

Example 1.3.2 We can also verify parametric systems by model checking. Henceforth, consider the parametric railroad system in Figure 1.5. We are interested in knowing for which values of the parameters the railroad is safe and for which is it not. In other words, we wish to derive the exact conditions on the parameters for which the property Φ holds. If we feed the parametric system and the property Φ to the parametric extension to Uppaal, then the tool will generate the constraint $\mathrm{st} + \mathrm{max\_lt} < \mathrm{min\_dt}$ as the exact condition needed for Φ to be satisfied.

In this thesis, we use the model checker Uppaal to establish the correctness of the IEEE 1394 Root Contention Protocol. Uppaal is a model checker for timed automata with a convenient graphical user interface. Moreover, this thesis presents a model checking technique to synthesize timing parameters for linear parametric timed automata, as we did in the second example above. This technique has been implemented on top of Uppaal. Model checkers for probabilistic systems are relatively new and are not used here.

Simulation and bisimulation

In the previous section, we expressed (a part of) the specification of the railroad system by the formula Φ. One can also describe a specification by an automaton. In order to prove that a

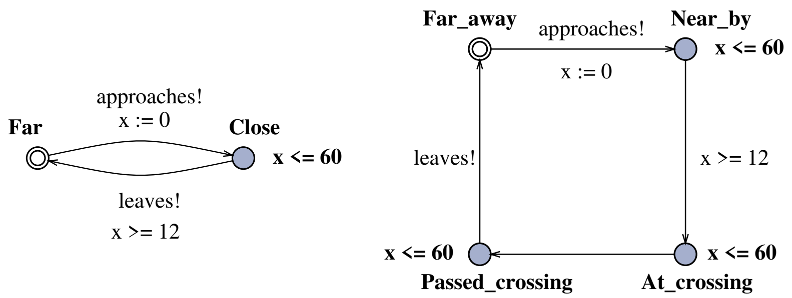


Figure 1.8: Bisimilar train models

system meets its specification, one shows that its external behavior is included in the external behavior of the specification automaton.

Different types of LTSs give rise to different notions of external behavior. Informally, the *external behavior* of an LTS, also called the *visible behavior*, is given by its sequences of external actions. The external behavior of a timed LTS also considers time passage as an external action and, in probabilistic LTSs, the probability on a sequence of actions is taken into account. Finally, the external behavior of parametric automata associates a set of action sequences to each parameter value. Note that internal actions and states do not belong to the external behavior of an LTS.

Proving behavior inclusion, that is showing that the external behavior of one automaton is included in the external behavior of another one, is a rather complex task. Therefore, simulation and bisimulation relations can be useful. These are relations that compare the stepwise behavior of two systems. When two systems are shown to be bisimilar, there is behavior inclusion. The idea behind bisimilar states is that each step one of them can take, can be mimicked by the others, as is explained below.

Example 1.3.3 Consider the leftmost automaton in Figure 1.8. We claim that it has exactly the same stepwise behavior as the train model leftmost in Figure 1.4, which is presented again in Figure 1.8 (rightmost), in order to make the comparison easier. The idea is that the state Far_away is equivalent to the state Far, i.e. they exhibit the same behavior, and so are the states $Near\_by$, $At\_crossing$, $Passed\_crossing$, and $Close$. This can be seen as follows; note that equivalent states are given the same color.

- In the states Far and Far_away, one can remain as long as desired.
- In the states $Near\_by$, $At\_crossing$, $Passed\_crossing$ and $Close$, one can stay any time less or equal than 60 time units.

This shows that the equivalent states have the same timing behavior. Now, we argue that they have the same stepwise behavior too. The idea is that if two states are equivalent and one can take a step from one state, then this step can also be taken by the other state (possibly via one or more internal transitions) and both steps lead again to equivalent states. This is exactly the idea of a weak bisimulation relation.

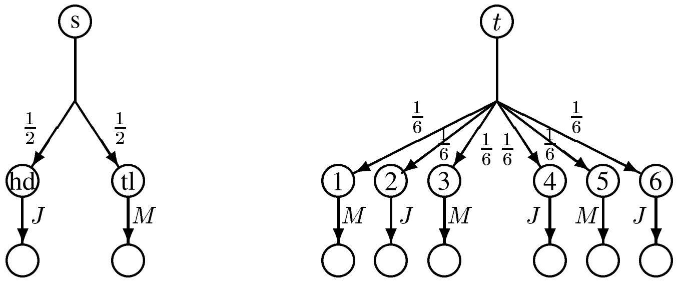


Figure 1.9: One coin flip is bisimilar with one dice roll

- In the state *Far*, one can move with an *approaches*! transition to the state *Close*, which is a grey state, and so can one move in the state *Far_away* with an *approaches*! transition to the state *Near_by*, which is also a grey state. Note that both transitions reset the clock x. Therefore, an *approaches*! transition moves from equivalent states to equivalent states.
- In each of the grey states, one can move with a *leaves*! transition to a white state, provided that $x \geq 12$. In the states *Near_by* and *At_crossing*, one has to take some internal transitions (respectively two and one) before taking the *leaves*! transition. This does not matter, because these are internal transitions, which are not visible.

Therefore, the two train models in Figure 1.8 are, what is called, *weakly bisimilar*.

Since bisimulation relations are compositional, we obtain an equivalent railroad system if we replace the rightmost train model in Figure 1.8 by the leftmost model.

Apart from bisimulations, simulation relations play an important role in this thesis. Simulation relations can be considered "unidirected" variants of bisimulations. In a bisimulation relation, all states have to exhibit the same behavior. In contrast, if a state s is related to t by a *simulation* relation, then all steps of s can also be performed by t, but not necessarily the other way around. Thus, t may exhibit more behavior than s.

Probabilistic simulation and bisimulation

> Do not worry about your difficulties in mathematics. I can assure you mine are still greater.
> Albert Einstein

Simulation and bisimulations are also useful for proving probabilistic systems correct. We illustrate their use by two examples.

Example 1.3.4 Suppose one wishes to flip a fair coin to make a decision, but only a dice is available. For instance, one wants to decide who is paying for the beers, if head comes up then John will and otherwise Mary will). What one can then of course do is the following. Someone rolls the dice and John pays if an even number comes up and otherwise Mary does. This situation is shown in Figure 1.9, where the actions J and M indicate who is paying and where the names of the bottom states, being irrelevant, have been left out.

We claim that both processes are probabilistically bisimilar: the states hd, 2, 4, and 6 are equivalent: they clearly exhibit the same stepwise behavior and so do tl, 1, 3, and 5. But

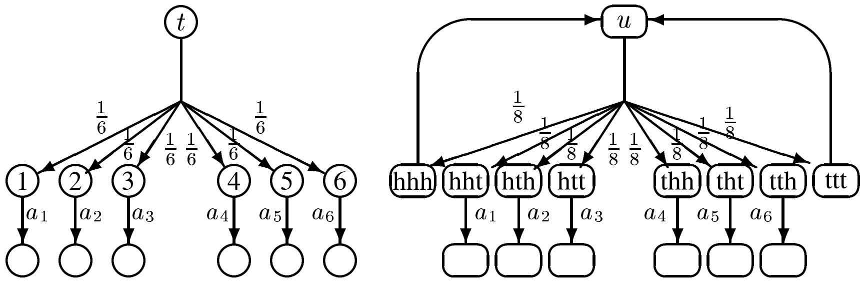


Figure 1.10: Many coin flips can be used for one dice roll

then also the states s and t are also bisimilar, because in both states, the probability to reach one of the equivalent states hd, 2, 4, 6 is equal to $\frac{1}{2}$ and so is the probability to reach one of the states tl, 1, 3, and 5. Thus, the basic idea in probabilistic bisimulation is that one does not consider the probability of a transition to reach an individual state, but the accumulated probability to reach one out of a set of equivalent states.

Example 1.3.5 Now consider the converse problem: one wants to decide on events $a_1, \ldots a_6$ (say who out of a group of 6 people will pay) with a fair dice, but only fair coins are available. Then one can toss 3 coins c_1, c_2, c_3, instead of a dice. If the outcome is either one of hht, hth, htt, thh, tht or tth then the corresponding person will pay. Here h stands for heads, t for tails and the outcomes of the three coins are listed in order. If either 3 heads or 3 tails come up, the coins are rethrown. This coin throwing process is repeated until the outcome is different from hhh or ttt. This situation is depicted in Figure 1.3.5. With elementary probability theory one shows that the probability that the coins will eventually be different equals 1. This means that the probability that one goes on coin flipping for ever is 0. (This probability is exactly 0, not "very close to 0!") Moreover, this implies that, eventually, each of actions a_i is taken with probability $\frac{1}{6}$. Thus, both processes in Figure 1.3.5 have exactly the same stepwise behavior. In other words, they are probabilistically bisimilar. In this type of bisimulation, one is allowed to combine arbitrary many internal transitions.

This thesis considers so–called **step refinements** and **hyperstep refinements**. These are the simplest kind of simulation relations for probabilistic systems that allow one to abstract from internal steps. These are relations that have the form of a function and require every step to be mimicked by a single step, except for internal transitions, which can be mimicked by remaining in the same state. Secondly, we develop several notions of *normed (bi–)simulation*. These are relations that also allow a limited abstraction from internal transitions and we prove that one can check efficiently (polynomial time and space) whether or not two states are normed bisimilar.

1.4 Overview and Results

The search for truth is more precious than its possession.
Albert Einstein

This thesis is divided into 7 chapters. Chapter 2 provides the basic model that we use throughout this thesis and Chapter 3 repeats some basic probability theory. The main results are stated in Chapters 4 – 7. These chapters can be read independently from each other. Since they originate from a collection of papers, the notations and concepts in these chapters are sometimes slightly different from the ones in Chapter 2. In some cases, definitions are repeated.

Main Results

The main results of this thesis can be summarized as follows.

- We present a testing scenario that justifies the definition of external behavior of (closed) PAs proposed in the literature. (Chapter 4)
- We develop a type of probabilistic (bi–)simulation that abstracts from internal computation in some sense and that can be computed efficiently. (Chapter 5)
- We present a model checking technique capable of synthesizing parameter constraints for the correctness of timed systems. (Chapter 6)
- We prove the correctness of the IEEE 1394 Root Contention Protocol using techniques from previous chapters and analyze the probabilistic, timing and parametric behavior of this protocol. (Chapter 7)

Summary

Chapter 2 recalls the probabilistic automaton (PA) model from the literature. This model has been developed for the modeling and verification of systems with discrete probabilities. We use this model throughout the thesis to formulate our verification techniques for probabilistic systems. Moreover, we carry out the verification of the Root Contention Protocol in this framework.

Chapter 2 explains the basic theoretical concepts and ideas behind this model. In particular, we show how PAs can be used to model systems with probabilities and how the basic concepts standard LTSs can be extended to PAs. We elaborate on the rather subtile combination of nondeterministic and probabilistic choice in the PA model. Moreover, we discuss the behavior of a PA, which is defined as the set of its trace distributions. Furthermore, we treat the notion of parallel composition, an extension of the PA model with time and two simulation relations for PAs. Finally, we give an overview of other models for the analysis of probabilistic systems.

As explained in Chapter 2, the implementation relation that has been proposed for (closed) PAs is trace distribution inclusion. This means that we say that one PA is a correct implementation of another if each of its trace distributions is also a trace distribution of the other PA. Chapter 4 justifies, for closed PAs, this implementation relation via a testing scenario. This comprehends a notion of observability for these automata based on relative frequencies. In particular, we define how probabilities can be "observed," for instance, how one can, distinguish between a fair and an unfair coin.

We analyze the outcomes of a number of independent runs of the PA with statistic methods. More precisely, we use the classical theory of hypothesis testing to determine whether a given list of outcomes is likely to be produced by that PA. If so, we consider this list to be an observation of the PA. The main result in this chapter states that the observations of one PA are included in (are equal to) the observations of another PA if and only if its trace distributions are included (equal to) those of the other PA.

Chapter 5 is devoted to simulation and bisimulation relations for PAs. First, we recall several (bi–)simulations known from the literature. Then we introduce a new type of relations, called *delay (bi–)simulation*, for which we consider four variants. These relations abstract from internal transitions only in a limited way, but what we gain is efficient algorithms to compute the (bi–)similar states automatically. In technically terms, delay (bi–)simulation can be decided in polynomial time and space. Furthermore, we give an alternative characterization of the delay (bi–)simulations in terms of norm functions.

In Chapter 6, we present a model checker that is able to synthesize parameters for linear timed parametric automata (LPTAs). These are timed automata where the clock guards and state invariants may contain linear equations over the timing parameters. The tool is an extension of the existing model checker Uppaal and compares favorably with similar tools. Moreover, we present an extensive elaboration of the theory behind our implementation. In particular, we present a correctness proof of the model checking algorithm we implemented.

Furthermore, we identify of a subclass of parametric timed automata (called L/U automata), for which the parametric model checking problem is much easier.

Probabilities, timing and parameters come together in the verification of the IEEE1394 Root Contention protocol. This protocol has become a popular case study in formal methods and we start Chapter 7 with an overview of several approaches to its verification. Then we present a protocol model that includes real–time, probabilistic and parameterized aspects of the protocol. The verification of this model is carried out manually, using the step and hyper step refinements. Since the communication in this model was discovered to be too abstract, we also present an enhanced protocol model. For this model, the real–time and parametric properties have been investigated, because in this respect the enhanced model differs from the first model. The verification has been carried out mechanically, using the model checker Uppaal. First, the parameter analysis has been done "experimentally" by checking a large number of instances and later, the model has been fully explored by the parametric extension of Uppaal.

History of the Chapters

This thesis is based on a collection of papers. Below, we mention their origins.

Chapter 2 This chapter explains the basic theory of probabilistic automata, which has been developed by Segala [Seg95b]. It contains few new technical results, except for the step refinement and hyperstep refinement relations. These appeared in [SV99b].

Chapter 3 This is a short chapter introducing a few concepts and notations from probability theory.

Chapter 4 The testing scenario presented in this chapter have been developed in cooperation with Frits Vaandrager and will appear as [SV02b].

Chapter 5 The work on (bi–)simulations for probabilistic systems reported in this chapter is joint work with Christel Baier. The notion of delay bisimulation version has been reported on in [BS00].

Chapter 6 The parametric extension to the model checker Uppaal is joint work with Thomas Hune, Judi Romijn and Frits Vaandrager. A shorter version has appeared in [HRSV01] and the full version will be published as [HRSV02].

Chapter 7 The case study described in this chapter has been reported on in several papers. The overview of the approaches to the verification of RCP is presented in [Sto01]. A discrete time verification, which was a starting point to further verification, appeared in [Sto99a]. The first verification described in this chapter was carried out together with Frits Vaandrager and was published in [SV99b]. The second verification described in this chapter was done in cooperation with David Simons and appeared as [SS01]. The third verification appeared as a part of [HRSV01].

CHAPTER 2
Probabilistic Automata

Al–hoewel in spelen daer alleen het geval plaats heeft, de uytkomsten onseecker zijn, so heeft nochtans de kansse die yemand om te winnen of te verliesen, haere seeckere bepaling [...] maer hoe veel minder kans hij heeft om te winnen of te verliesen, dat is in selver seecker, en werde door reeckeningh uyt–gevonden."

Christiaan Huygens, *Van Rekeningh in Spelen van Geluck* [Huy50]

Abstract This chapter provides an informal introduction to the probabilistic automaton (PA) model. This framework has been developed by Segala [Seg95b] and serves as a basis for subsequent chapters. We describe how distributed systems with discrete probabilities can be modeled and analyzed by means of PAs. We also point out how the basic concepts for the analysis of nonprobabilistic systems can be extended to probabilistic systems. In particular, we treat the parallel composition operator on PAs, the semantics of a PA as a set of trace distributions, an extension of the PA model with time and two simulation relations for PAs. Finally, we give an overview of various other state based models that are used for the analysis of probabilistic systems.

2.1 Introduction

Probabilistic Automata (PAs) constitute a mathematical framework for the specification and analysis of probabilistic systems. They have been developed by Segala [Seg95b, SL95, Seg95a] for the purpose of modeling and analyzing asynchronous, concurrent systems with discrete probabilistic choice in a formal and precise way. Examples of such systems are randomized, distributed algorithms, such as the randomized dining philosophers [LR81]; probabilistic communication protocols, such as the IEEE1394 Root Contention protocol [IEE96] and the Binary Exponential Back Off protocol (see [Tan81]); and fault tolerant systems, such as unreliable communication channels.

PAs are based on state transition systems and make a clear distinction between probabilistic and nondeterministic choice. We will extensively treat the differences and similarities between both types of choices. In fact, PAs can be considered as a combination of discrete time Markov chains and (nondeterministic) state machines. Moreover, PAs subsume the Markov decision processes. The PA framework does not provide any syntax. However, several syntaxes could be defined (and in fact have been defined) on top of it to facilitate the modeling of a system as a PA.

A PA has a well–defined semantics as a set of trace distributions. The parallel composition operator $||$ allows one to construct a PA from several component PAs running in parallel and thus keeping system models modular. Properties of probabilistic systems that can be established formally using PAs include correctness and performance issues, such as: Is the probability that an error occurs small enough (e.g. $< 10^{-9}$)? Is the average time between two failures large enough?

The aim of this chapter is to explain the basic ideas behind PAs and their behavior in an informal way. In our explanation, we stress the differences and similarities with nonprobabilistic automata.

Organization of the chapter

This chapter is organized as follows. Section 2.2 introduces the probabilistic automaton model, Section 2.3 treats the behavior (or semantics) of the model and Section 2.4 is concerned with simulation relations for PAs. In Section 2.6, we give several other models dealing with probabilistic choice. Finally, Section 2.7 presents a summary of the chapter.

2.2 The Probabilistic Automaton Model

Basically, a probabilistic automaton is just an ordinary automaton (also called *labeled transition system* or *state machine*) with the only difference that the target of a transition is a probabilistic choice over several next states. Before going into the details of this, we briefly explain the notion of a nonprobabilistic automaton, abbreviated NA.

2.2.1 Nonprobabilistic Automata

As we have already pointed out in Chapter 1, an NA is a structure consisting of *states* and *transitions*; the latter are also called *steps*. The states of an NA represent the states in the system that is modeled. One or more states are designated as *start states*, representing the initial configuration of the system. The transitions in the NA represent changes of the system states and are labeled by actions. Thus, the transition $s \xrightarrow{a} s'$ represents that, being in state s, the system can move via an a–action to another state s'. The action labels are partitioned into *internal* and *external* actions. The former represent internal computation steps of the automaton[1] and are not visible to the automaton's environment. We often use the symbol τ to denote an internal action. The external actions are visible for the automaton's environment and are used for interaction with it. This will be explained in more detail in Section 2.2.4.

Example 2.2.1 Consider the NA in Figure 2.1. It models a 1–place buffer used as a communication channel for the transmission of bits between two processes. The states ε, 0 and 1 represent respectively the channel being empty, the channel containing the bit 0 and the channel containing a 1. Initially, the channel is empty (denoted by the double circle in the picture). The transitions $\varepsilon \xrightarrow{snd(i)} i$ represent the sender process (not depicted here) sending the bit i to the channel. Similarly, the transitions $i \xrightarrow{rec(i)} \varepsilon$ represent the delivery of the bit at the receiver process. Notice that the transition labeled $snd(i)$ represents the sending of a bit by the sender, which is the receipt of a bit by the communication channel. Similarly, the receipt of a bit at the receiving process corresponds to the sending of a bit by the channel, which is modeled by the $rec(i)$–transitions.

It is natural that the actions $snd(0)$, $rec(0)$, $snd(1)$ and $rec(1)$ in this NA are external, since these are used for communication with the environment.

[1]Note that at this point, the notation differs from Chapter 1, where internal transitions were unlabeled. In the current model, each transition has a label.

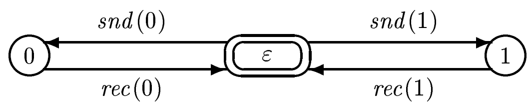


Figure 2.1: A channel automaton

Thus, the notion of an NA can be formalized as follows.

Definition 2.2.2 An NA $\mathcal{A}$ consists of four components:

1. A set $S_{\mathcal{A}}$ of *states*,
2. a nonempty set $S^0_{\mathcal{A}} \subseteq S_{\mathcal{A}}$ of *start states*,
3. An *action signature* $sig_{\mathcal{A}} = (V_{\mathcal{A}}, I_{\mathcal{A}})$, consisting of *external (visible)* and *internal actions* respectively. We require $V_{\mathcal{A}}$ and $I_{\mathcal{A}}$ to be disjoint and define the set of *actions* as $\mathrm{Act}_{\mathcal{A}} = V_{\mathcal{A}} \cup I_{\mathcal{A}}$.
4. a *transition relation* $\Delta_{\mathcal{A}} \subseteq S_{\mathcal{A}} \times \mathrm{Act}_{\mathcal{A}} \times S_{\mathcal{A}}$.

We write $s \xrightarrow{a}_{\mathcal{A}} s'$, or $s \xrightarrow{a} s'$ if $\mathcal{A}$ is clear from the context, for $(s, a, s') \in \Delta_{\mathcal{A}}$. Moreover, we say that the action a is *enabled* in s, if s has an outgoing transition labeled by a.

Several minor variations of this definition exist. Other definitions require, for instance, a unique start state rather than a set of start states or allow only a single internal action, rather than a set of these. In the I/O automaton model [LT89], external actions are divided into *input* and *output* actions. Input actions, not being under the control of the NA, are required to be enabled in any state. The basic concepts are similar for all the various definitions.

Nondeterminism

Nondeterministic choices can be specified in an automaton by having several transitions leaving from the same state. Nondeterminism is used when we wish to incorporate several potential system behaviors in a model. Hoare [Hoa85] phrases it as follows:

> There is nothing mysterious about nondeterminism, it arises from the deliberated decision to ignore the factors which influence the selection.

Nondeterministic choices are often divided into *external* and *internal* nondeterministic choices. External nondeterministic choices are choices that can be influenced by the environment. Since interaction with the environment is performed via external actions, external nondeterminism is incorporated by having several transitions with different labels leaving from the same state.

Internal nondeterministic choices are choices that are made by the automaton itself, independent of the environment. These are modeled by internal actions or by having several transitions with the same labels leaving from the same state. In the literature, the word nondeterminism sometimes refers to what we call internal nondeterminism.

As pointed out by [Seg95b, Alf97], nondeterminism is essential for the modeling of the following phenomena.

Scheduling freedom When a system consists of several components running in parallel, we often do not want to make any assumptions on the relative speeds of the components, because we want the application to work no matter what these relative speeds are. Therefore, nondeterminism is essential to define the parallel composition operator (see Definition 2.2.14), where we model the choice of which automaton in the system takes the next step as an (internal or external) nondeterministic choice.

Implementation freedom Automata are often used to represent a specification. Good software engineering practice requires the specification to describes *what* the system should do, not *how* it should be implemented. Therefore, a specification usually leaves room for several alternative implementations. Since it does not matter for the correct functioning of the system which of the alternatives is implemented, such choices are also represented by (internal or external) nondeterminism.

External environment An automaton interacts with its environment via its external actions. When modeling a system, we do not wish to stipulate how the environment will behave. Therefore the possible interactions with the environment are modeled by (external) nondeterministic choices.

Incomplete information Sometimes it is not possible to obtain exact information about the system to be modeled. For instance, one might not know the exact duration of or — in probabilistic systems — the exact probability of an event, but only a lower and upper bound. In that case, one can incorporate all possible values by a nondeterministic choice. This is appropriate since we consider a system to be correct if it behaves as desired no matter how the nondeterministic choices are resolved.

Example 2.2.3 In the state ε, the channel NA in Figure 2.1 contains an external nondeterministic choice between the actions $snd(0)$ and $snd(1)$. This models a choice to be resolved by the environment (in this case a sender process) which decides which bit to sent.

Nondeterministic choices modeling implementation freedom, scheduling freedom, and incomplete information are given in Examples 2.2.9 and 2.2.15.

2.2.2 Probabilistic Automata

As said before, the only difference between a nonprobabilistic and a probabilistic automaton is that the target of a transition in the latter is no longer a single state, but is a probabilistic choice over several next states. For instance, a transition in a PA may reach one state with probability $\frac{1}{2}$ and another one with probability $\frac{1}{2}$ too. In this way, we can represent a coin flip and a dice roll, see Figure 2.2.

Thus, a transition in a PA relates a state and an action to a *probability distribution* over the set of states. A probability distribution over a set X is a function μ that assigns a probability in $[0, 1]$ to each element of X, such that the sum of the probabilities of all elements is 1. The reader is referred to Chapter 3 for a formal definition. Let $\mathrm{Distr}(X)$ denote the set of

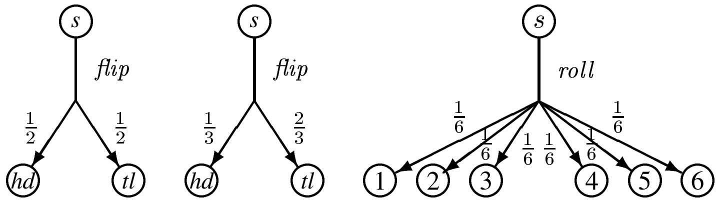


Figure 2.2: Transitions representing a fair coin flip, an unfair coin flip and a fair dice roll.

all probability distributions over X. The *support* of μ is the set $\{x \in X \mid \mu(x) > 0\}$ of elements that are assigned a positive probability. If $X = \{x_1, x_2, \ldots\}$, then we often write the probability distribution μ as $\{x_1 \mapsto \mu(x_1), x_2 \mapsto \mu(x_2) \ldots\}$ and we leave out the elements that have probability 0.

Example 2.2.4 Let the set of states be given by s, hd, tl, 1, 2, 3, 4, 5 and 6. The transitions in Figure 2.2 are respectively given by

$$s \xrightarrow{\mathit{flip}} \{hd \mapsto \tfrac{1}{2}, tl \mapsto \tfrac{1}{2}\},$$
$$s \xrightarrow{\mathit{flip}} \{hd \mapsto \tfrac{1}{3}, tl \mapsto \tfrac{2}{3}\} \text{ and}$$
$$s \xrightarrow{roll} \{1 \mapsto \tfrac{1}{6}, 2 \mapsto \tfrac{1}{6}, 3 \mapsto \tfrac{1}{6}, 4 \mapsto \tfrac{1}{6}, 5 \mapsto \tfrac{1}{6}, 6 \mapsto \tfrac{1}{6}\}.$$

Note that we have left out many elements with probability 0, for instance the state s is reached with probability 0 by each of the transitions above. Moreover, each of the three pictures in Figure 2.2 represents a single transition, where several arrows are needed to represent the probabilistic information.

The definition of a PA is now given as follows.

Definition 2.2.5 A *PA* $\mathcal{A}$ consists of four components:

1. A set $S_{\mathcal{A}}$ of *states*,
2. a nonempty set $S^0_{\mathcal{A}} \subseteq S_{\mathcal{A}}$ of *start states*,
3. An *action signature* $sig_{\mathcal{A}} = (V_{\mathcal{A}}, I_{\mathcal{A}})$, consisting of *external* and *internal actions* respectively. We require $V_{\mathcal{A}}$ and $I_{\mathcal{A}}$ to be disjoint and define the set of *actions* as $\mathrm{Act}_{\mathcal{A}} = V_{\mathcal{A}} \cup I_{\mathcal{A}}$.
4. a *transition relation* $\Delta_{\mathcal{A}} \subseteq S_{\mathcal{A}} \times \mathrm{Act}_{\mathcal{A}} \times \mathrm{Distr}(S_{\mathcal{A}})$.

Again, we write $s \xrightarrow{a}_{\mathcal{A}} \mu$ for $(s, a, \mu) \in \Delta_{\mathcal{A}}$. Furthermore, we simply write $s \xrightarrow{a}_{\mathcal{A}} s'$ for $s \xrightarrow{a}_{\mathcal{A}} \{s' \mapsto 1\}$.

Obviously, the definition of PAs gives rise to the same variations as the definition of NAs. Table 2.16 gives an overview of the variations used in subsequent chapters of this thesis. The basic concepts are the same for all variants.

Example 2.2.6 The PA in Figure 2.3 represents the same 1–place communication channel as the NA in Figure 2.1, except that now the channel is lossy: a bit sent to the channel is lost with a probability of $\frac{1}{100}$. By convention, the transitions $i \xrightarrow{rec(i)} \varepsilon$ reach the state ε with probability 1.

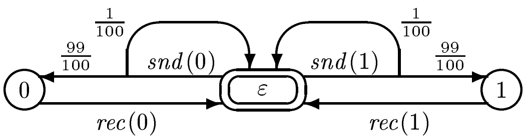


Figure 2.3: A lossy channel PA

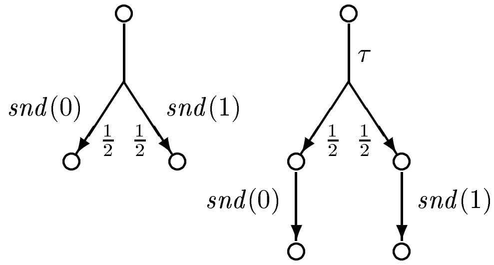


Figure 2.4: Modeling multi–labeled transitions in a PA

Remark 2.2.7 Note that each transition in a PA is labeled with a single action. The system depicted on the left in Figure 2.4 models a process sending the bits 0 and 1 each with probability $\frac{1}{2}$. However, this system is not a PA, because two actions appear in the same transition. Rather, a PA model of such a process is shown on the right of Figure 2.4.

There is, however, a multilabeled version of the PA model, see Section 2.2.5. That model is technically more complex and, in practice, the PA model is expressive enough.

Remark 2.2.8 The nonprobabilistic automata can be embedded in the probabilistic ones by viewing each transition $s \xrightarrow{a} s'$ in an NA as the transition $s \xrightarrow{a} \{s' \mapsto 1\}$ in a PA. Conversely, each PA $\mathcal{A}$ can be "deprobabilized," yielding an NA $\mathcal{A}^-$, by forgetting the specific probabilistic information and by only considering whether a state can be reached with a positive probability. That is, $s \xrightarrow{a} s'$ is a transition in $\mathcal{A}^-$ if and only if there is a transition $s \xrightarrow{a} \mu$ in $\mathcal{A}$ with $\mu(s') > 0$.

Probabilistic versus nondeterministic choice

One can specify nondeterministic choices in a PA in exactly the same way as in a NA, viz. by having internal transitions or by having several transitions leaving from the same state. Also the distinction between external and internal nondeterminism immediately carries over to PAs. Hence, the probabilistic choices are specified *within* the transitions of a PA and the nondeterministic choices *between* the transitions (leaving from the same state) of a PA.

Section 2.2.1 has pointed out the need for nondeterministic choices in automata, namely to model scheduling freedom, implementation freedom, the external environment and incomplete information. These arguments are still valid in the presence of probabilistic choice. In

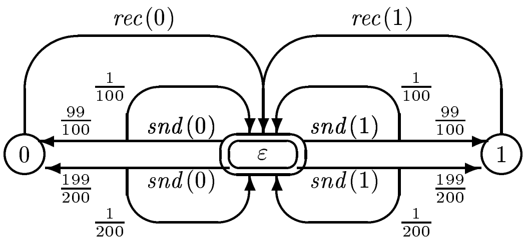


Figure 2.5: A lossy channel PA with partially unknown probabilities

particular, *nondeterminism cannot be replaced by probability* in these cases. As mentioned, nondeterminism is used if we deliberately decide not to specify how a certain choice is made, so in particular we do not want to specify a probability mechanism that governs the choice. Rather, we use a probabilistic choice if the event to be modeled has really all the characteristics of a probabilistic choice. For instance, the outcome of a coin flip, random choices in programming languages, and the arrivals of consumers in a shop.

Thus, probability and nondeterminism are two orthogonal and essential ingredients in the PA model.

An important difference between probabilistic and nondeterministic choice is that the former are governed by a probability mechanism, whereas the latter are completely free. Therefore, probabilistic choices fulfill the laws from probability theory, and in particular the law of large numbers. Informally, this law states that if the same random choice is made very often, the average number of times that a certain event occurs is approximately (or, more precisely, it converges to) its expected value. For instance, if we flip a fair coin one hundred times, it is very likely that about half of the outcomes is heads and the other half is tails. If, on the other hand, we make a nondeterministic choice between two events, then we cannot quantify the likelihood of the outcomes. In particular, we cannot say that each of the sequences is equally likely, because this refers to a probabilistic choice!

The following example illustrates the combination of nondeterministic choice and probabilistic choice.

Example 2.2.9 Like in Figure 2.3, the PA in Figure 2.5 also represents a faulty communication channel. The difference with the PA in Figure 2.3 is that, here, we do not know the probability that a bit is lost exactly. Depending on which transition is taken, it can be $\frac{1}{200}$ or $\frac{1}{100}$. However, we will see in Section 2.3 that this probability can in fact have any value between $\frac{1}{200}$ and $\frac{1}{100}$. Thus, the nondeterministic choice between the two $send(i)$ transitions is used here to represent incomplete information about exact probabilities.

This PA can also be considered as the specification of a system. Then the nondeterministic choice represents implementation freedom: A PA correctly implements the one in Figure 2.5 if the probability to lose a message is at most $\frac{1}{100}$ and at least $\frac{1}{200}$. (The latter might be a bit awkward in practice.)

For future use, define distributions μ_x^i and the transitions θ_x^i as follows. For $x = 100, 200$ and $i = 0, 1$ let

$$\theta_x^i = \varepsilon \xrightarrow{snd(i)} \mu_x^i, \qquad \mu_x^i = \{\varepsilon \mapsto \tfrac{1}{x}, i \mapsto \tfrac{x-1}{x}\}.$$

Thus, the superscripts denote the bit and the subscripts the probability involved.

Finally, we remark that the philosophical debate on the nature of nondeterminism choice and probability has not ended. In this thesis, we do not take a standpoint in that discussion and neither do we contribute to it. Rather, we take a practical point of view. The only assumption we rely on in this thesis that both types of choices are appropriate for describing and predicting certain phenomena. For more information on the philosophical issues arising in the area of probability theory, the reader is referred to [Coh89].

2.2.3 Timing

Timing can be incorporated in the PA model in a similar way as in the NA model (c.f. the "old fashioned recipe for time" [AL92]). A *probabilistic timed automaton (PTA)* is a PA with time passage actions. These are actions $d \in \mathbb{R}^{>0}$ that indicate the passage of d time units. While time elapses, no other actions take place and, in the PTA approach, time advances deterministically. So, in particular, no (internally) nondeterministic or probabilistic choices can be specified within time passage actions, see requirements 1 and 2 in the definition below. The third requirement below, Wang's Axiom [Yi90], requires that, while time advances, the state of the PTA is well–defined at each point in time and that, conversely, two subsequent time passage actions can be combined into a single one.

The PTA model presented here is somewhat simpler, but similar to the one in [Seg95b], which is based on a variation of Wang's axiom.

Definition 2.2.10 A PTA $\mathcal{A}$ is a PA enriched with a partition of $\mathrm{Act} \setminus \{\tau\}$ into a set of *discrete actions* Act_D and the set $\mathbb{R}^{>0}$ of positive real numbers or *time–passage actions*. We require[2] that, for all $s, s', s'' \in S_{\mathcal{A}}$ and $d, d' \in \mathbb{R}^{>0}$ with $d' < d$,

1. each transition labeled with a time–passage action leads to a distribution that chooses one element with probability 1,
2. (Time determinism) if $s \xrightarrow{d}_{\mathcal{A}} s'$ and $s \xrightarrow{d}_{\mathcal{A}} s''$ then $s' = s''$.
3. (Wang's Axiom) $s \xrightarrow{d}_{\mathcal{A}} s'$ iff $\exists s'' : s \xrightarrow{d'}_{\mathcal{A}} s''$ and $s'' \xrightarrow{d-d'}_{\mathcal{A}} s'$.

As PTAs are a special kind of PAs, we can use the notions defined for PAs also for PTAs.

By letting time pass deterministically, PTAs treat probabilistic choice, nondeterministic choice and time passage as orthogonal concepts, which leads to a technically clean model. Example 2.2.11 below shows that discrete probabilistic choices over time can be encoded in PTAs via internal actions. Nondeterministic choices over time can be encoded similarly: just replace the probabilistic choice in the example by a nondeterministic one. Thus, although we started from a deterministic view on time, nondeterminism and probabilistic choices over time sneak in via a back door. The advantage of the PTA approach is that we separate concerns by specifying one thing at the time: time passage or probabilistic/nondeterministic choice.

Example 2.2.11 We can use a PTA to model a system that decides with an internal probabilistic choice whether to wait a short period (duration one time unit) or a long period (two time units) before performing an a action. This PTA is partially given in Figure 2.6, where the second element of each state records the amount of time that has elapsed. Note that there are uncountably many transitions missing in the picture, for instance the transitions

[2] For simplicity the conditions here are slightly more restrictive than those in [LV96b].

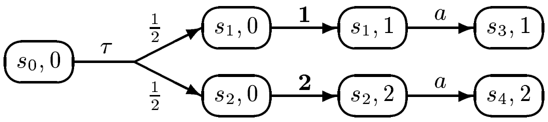


Figure 2.6: A part of a PTA

$(s_1, 0) \xrightarrow{\mathbf{0.2}} (s_1, 0.2)$ and $(s_1, 0.2) \xrightarrow{\mathbf{0.5}} (s_1, 0.7)$, see Wang's Axiom in the preceding definition. The full transition relation is given by

$$
\begin{array}{ll}
(s_0, 0) \xrightarrow{\tau} \{(s_1, 0) \mapsto \frac{1}{2}, (s_2, 0) \mapsto \frac{1}{2}\}, & \\
(s_1, d) \xrightarrow{d'} (s_1, d + d'), & \text{if } d + d' \le 1, \\
(s_2, d) \xrightarrow{d'} (s_2, d + d'), & \text{if } d + d' \le 2, \\
(s_1, 1) \xrightarrow{a} (s_3, 1), & \\
(s_2, 2) \xrightarrow{a} (s_4, 1). &
\end{array}
$$

Continuous distributions over time can of course not be encoded in a PTA, since such distributions cannot be modeled in the PA model anyhow. There are various other models combining time and probability, see Section 2.6, including models dealing with continuous distributions over time.

Moreover, there is a second model that extends PAs with nondeterministic timing. The automata introduced in [KNSS01] — also called PTAs — augment the classical timed automata [AD94] with discrete probabilistic choice. They allow timing constraints to be specified via real–valued clocks, as in Chapter 1. An example PTA model of this kind has been given in Example 1.2.4 on page 18.

2.2.4 Parallel Composition

The parallel composition operator $\|$ allows one to construct a PA from several component PAs. This makes system descriptions more understandable and enables component–wise design. The component PAs run in parallel and interact via their external actions. As before, the situation is similar to the nonprobabilistic case.

Consider a composite PA that is built from two component PAs. Then the state space of the composite PA consists of pairs (s, t), reflecting that the first component is in state s and the second in state t. If one of the components can take a step, then so can the composite PA, where *synchronization* on shared actions has to be taken into account. This means that whenever one component performs a transition involving a visible action a, the other one should do so simultaneously, provided that a is in its action set.

When synchronization occurs, both automata resolve their probabilistic choices independently, because the probability mechanisms used in different components are not supposed to influence each other. Thus, if the transitions $s_1 \xrightarrow{a} \mu_1$ and $s_2 \xrightarrow{a} \mu_2$ synchronize, then the state (s_1', s_2') is reached with probability $\mu_1(s_1') \cdot \mu_2(s_2')$.

No synchronization is required for transitions labeled by an internal action $a \in I$ nor for visible actions which are not shared (i.e. present in only one of the automata). In this case, one component takes a transition, while the other remains in its current state with probability one. For instance, if the first component takes the transition $s_1 \xrightarrow{\tau} \mu_1$ and the other one

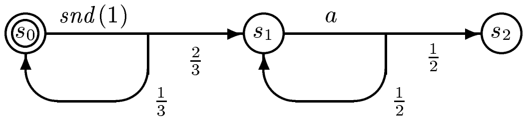


Figure 2.7: A sender PA

remains in the state s_2, then the probability to reach the state (s'_1, s_2) by taking this transition equals $\mu_1(s'_1)$ and the probability to reach a state (s'_1, s'_2) with $s'_2 \neq s_2$ is zero.

To define the parallel composition, we first need two auxiliary definitions. The first one defines the probabilistic choice arising from two independent probabilistic choices. The second one ensures that we only take the parallel composition of PAs whose action signatures do not clash.

Notation 2.2.12 Let μ be a probability distribution on X and ν one on Y. Define the distribution $\mu \times \nu$ on $X \times Y$ by

$$(\mu \times \nu)(x, y) = \mu(x) \cdot \nu(y).$$

Definition 2.2.13 We say that two PAs $\mathcal{A}$ and $\mathcal{B}$ are *compatible* if $I_{\mathcal{A}} \cap \mathrm{Act}_{\mathcal{B}} = \mathrm{Act}_{\mathcal{A}} \cap I_{\mathcal{B}} = \emptyset$.

Definition 2.2.14 For two compatible PAs $\mathcal{A}$ and $\mathcal{B}$, the *parallel composition* is the probabilistic automaton $\mathcal{A} \parallel \mathcal{B}$ defined by:

1. $S_{\mathcal{A}\parallel\mathcal{B}} = S_{\mathcal{A}} \times S_{\mathcal{B}}$.
2. $S^0_{\mathcal{A}\parallel\mathcal{B}} = S^0_{\mathcal{A}} \times S^0_{\mathcal{B}}$.
3. $sig_{\mathcal{A}\parallel\mathcal{B}} = (V_{\mathcal{A}} \cup V_{\mathcal{B}}, I_{\mathcal{A}} \cup I_{\mathcal{B}})$.
4. $\Delta_{\mathcal{A}\parallel\mathcal{B}}$ is the set of transitions $(s_1, s_2) \xrightarrow{a} \mu_1 \times \mu_2$ such that at least one of the following requirements is met.
   - $a \in V_{\mathcal{A}} \cap V_{\mathcal{B}}$, $s_1 \xrightarrow{a} \mu_1 \in \Delta_{\mathcal{A}}$ and $s_2 \xrightarrow{a} \mu_2 \in \Delta_{\mathcal{B}}$.
   - $a \in \mathrm{Act}_{\mathcal{A}} \setminus \mathrm{Act}_{\mathcal{B}}$ or $a \in I_{\mathcal{A}}$, and $s_1 \xrightarrow{a} \mu_1 \in \Delta_{\mathcal{A}}$ and $\mu_2 = \{s_2 \mapsto 1\}$.
   - $a \in \mathrm{Act}_{\mathcal{B}} \setminus \mathrm{Act}_{\mathcal{A}}$ or $a \in I_{\mathcal{B}}$, and $s_2 \xrightarrow{a} \mu_2 \in \Delta_{\mathcal{B}}$ and $\mu_1 = \{s_1 \mapsto 1\}$.

Note that nondeterminism is essential in this definition.

Example 2.2.15 The system in Figure 2.7 represents a sender process. Its action set is $\{snd(0), snd(1), a\}$. Figure 2.8 shows the parallel composition process of this process and the channel process in Figure 2.3.

2.2.5 Other Automaton Models combining Nondeterminism and Discrete Probabilities

Below, we discuss two other automaton–based frameworks that combine discrete probabilistic and nondeterministic choice. We treat the GPA model and the alternating model, which are both equivalent to the PA model in some sense.

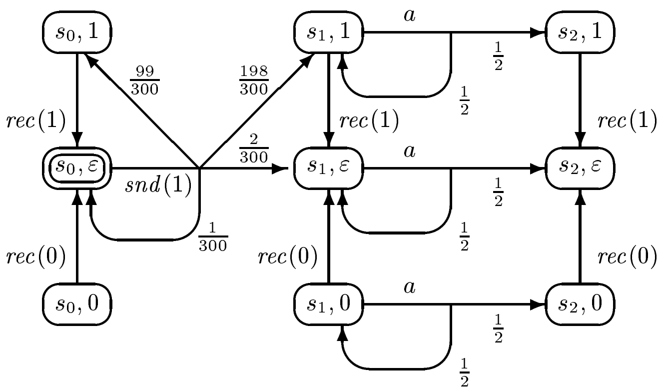


Figure 2.8: Parallel composition

General probabilistic automata

Segala [Seg95b] introduces a more general notion of probabilistic automata, which we call *general probabilistic automata* (GPAs)[3]. A GPA is the same as a PA, except that the transitions have the type $S \times \mathrm{Distr}(\mathrm{Act} \times S \cup \{\bot\})$. Thus, each transition chooses both the action and the next state probabilistically. Moreover, it can choose to deadlock ($\bot$) with some probability. In the latter case, no target state is reached. Figure 2.4 on page 34 shows a GPA that is not a PA.

Problems arise when defining the parallel composition operator for arbitrary GPAs. The trouble comes from synchronizing transitions that have some shared actions and some actions which are not shared (see [Seg95b], Section 4.3.3).

The problem can be solved by imposing the I/O distinction on GPAs (c.f. the remark below Definition 2.2.2). This distinction comes with the requirement that input actions are enabled in every state and occur only on transitions labeled by a single action. This approach is followed in [WSS97] and in [HP00]. Surprisingly, the latter reverses the role of input and output actions.

In our experience, many practical systems can be modeled conveniently with PAs (see Remark 2.2.7; deadlocks can be modeled by moving to a special deadlock state). Moreover, several notions have been worked out for the PA model only. Therefore, this thesis deals with PAs rather than with GPAs.

Alternating model

The alternating model, introduced by Hansson and Jonsson [Han94, HJ94], distinguishes between *probabilistic* and *nondeterministic states*. Nondeterministic states have zero or more outgoing transitions. These are labeled by actions and lead to a unique probabilistic state. Probabilistic states have one or more outgoing transitions. These are labeled by probabilities and specify a probability distribution over the nondeterministic states.

The alternating model and the PA model are isomorphic upto so-called strong bisimu-

[3]Segala uses the word PA for what we call GPA and says simple PA to what we call PA.

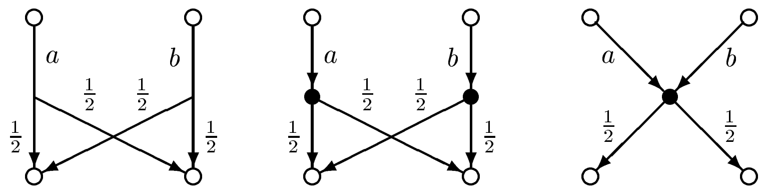


Figure 2.9: A PA model and two alternating models equivalent to it

lation [LS91]. (This is a kind of graph isomorphism on unfolded automata that takes into account the action labels, see Chapter 5 for details.) This means that all notions defined on PAs that respect strong bisimulation can be translated into the alternating model and vice versa.

In order to translate an alternating model into a PA, one removes all probabilistic states and contracts each ingoing transition of a probabilistic state with the probability distribution going out of that state, thus passing by the probabilistic state. Conversely, in order to translate a PA into an alternating model, one introduces an intermediate probabilistic state for each transition. The reason why this only yields an isomorphism upto bisimulation, rather than just an isomorphism, is illustrated in Figure 2.9.

2.3 The Behavior of Probabilistic Automata

This section defines the semantics of a PA as the set of its trace distributions. Each trace distribution is a probability space assigning a probability to certain sets of traces. The idea is that a trace distribution arises from resolving the nondeterministic choice first and by then abstracting from nonvisible elements, i.e. by removing the states and internal actions and leaving only the external actions. A difference with the nonprobabilistic case is that the theory of PAs allows nondeterministic choices to be resolved by probabilistic choices. As we will see, these are described by an adversary. Moreover, resolving the nondeterministic choices in a PA no longer yields a linear execution, as in an NA but can be considered as a tree–like structure.

We recall the definition of traces for NAs first.

2.3.1 Paths and Traces

The semantics of an NA is given by the set of its traces. Each trace represents one of the potential visible behaviors of the system and is obtained by the following steps. First, the nondeterministic choices in the NA are resolved. This yields an execution, i.e. a sequence of states and actions. Then the states and internal actions are removed from the execution. This yields a sequence of actions, called a *trace*.

Definition 2.3.1 1. An *path* (or *execution*) of an NA $\mathcal{A}$ is a possibly infinite sequence $\pi = s_0 a_1 s_1 a_2 \dots$ where s_0 is an initial state of $\mathcal{A}$, s_i is a state of $\mathcal{A}$, a_i is an (internal or external) action of $\mathcal{A}$ and $s_i \xrightarrow{a_{i+1}}_{\mathcal{A}} s_{i+1}$ is a transition. Moreover, we require that if π is finite, then it ends in a state.

2. A *trace* is a finite or infinite sequence of external actions that is obtained from a path by omitting the states and internal actions. We denote the set of traces of $\mathcal{A}$ by $trace(\mathcal{A})$.

Example 2.3.2 The three sequences ε and $\langle \varepsilon\ snd(1)\ 1\ rec(1)\ \varepsilon\ snd(0)\ 0\rangle$ and $\langle \varepsilon\ snd(0)\ 0\ rec(0)\ \varepsilon\ snd(0)\ 0\ rec(0)\ldots\rangle$ are paths of the NA in Figure 2.1. Their traces are, respectively, the empty sequence ε, the sequence $\langle snd(1)\ rec(1)\ snd(0)\rangle$ and the sequence $\langle snd(0)\ rec(0)\ snd(0)\ rec(0)\ snd(0)\ldots\rangle$. (The brackets $\langle\rangle$ here have been inserted for the sake of readability, but have no meaning.)

We can also identify paths and traces in a PA.

Definition 2.3.3 1. A *probabilistic path* (abbreviated as *path*) of a PA $\mathcal{A}$ is an alternating, finite or infinite sequence

$$\pi \ = \ s_0 a_1 \mu_1 s_1 a_2 \mu_2 s_2 a_3 \mu_3 \ldots$$

where $s_0 \in S^0_{\mathcal{A}}$ $s_i \in S_{\mathcal{A}}$, $a_i \in \mathrm{Act}_{\mathcal{A}}$, $s_i \xrightarrow{a_{i+1}}_{\mathcal{A}} \mu_{i+1}$ and $\mu_{i+1}(s_{i+1}) > 0$. Moreover, if π is finite, then it has to end in a state. Let $last(\pi)$ denote the last state of a finite path, $|\pi| \in \mathbb{N}\cup\{\infty\}$ the number of actions occurring in π, $Path^*(\mathcal{A})$ the set of finite paths of $\mathcal{A}$ and $Path^*(s,\mathcal{A})$ the set of finite paths in $\mathcal{A}$ starting in state s.

2. A *trace* is a finite or infinite sequence of external actions that is obtained from a path by omitting the states, internal actions and distributions. Let $trace$ denote the function that assigns to each execution its trace.

Both the probabilistic and the nondeterministic choices have been resolved in a probabilistic path, since it reveals that the i^{th} transition taken is $s_{i-1} \xrightarrow{a_i} \mu_i$ and that the outcome of the i^{th} probabilistic choice μ_i is s_i. Moreover, note that each probabilistic path of $\mathcal{A}$ gives rise to a path of $\mathcal{A}^-$, by removing all the probability distributions.

Example 2.3.4 The sequence $\langle \varepsilon\ snd(1)\ \mu^1_{100}\ \varepsilon\ snd(1)\ \mu^1_{100}\ 1\rangle$ is a finite probabilistic path of the PA in Figure 2.5. Its trace is $\langle snd(1)\ snd(1)\rangle$.

In practical verifications, a lot of effort is usually spent on finding out which states in a system can be reached and which cannot. Thus, the following definition prepares for the case study in Chapter 7.

Definition 2.3.5 A state s in a NA or a PA $\mathcal{A}$ is *reachable* if there is a finite path ending in s. We denote the set of reachable states by $reach_{\mathcal{A}}$.

2.3.2 Trace Distributions

The semantics of a PA is given by the set of distributions over its traces, called *trace distributions*. Each trace distribution of a PA represents one of the potential visible behaviors of the system, just as a trace in an NA. As said before, a trace distribution is obtained by resolving the nondeterministic choices in the PA (replacing them by probabilistic choices) first and by then removing the states and the internal actions next.

More precisely, the way in which the nondeterministic choices in a PA are resolved is given by a so–called *randomized, partial adversary.* An adversary can be considered as the equivalent of an execution in an NA. To study the probabilistic behavior generated by an adversary, a *probability space* is associated to each adversary. This assigns a probability to certain sets of paths in the PA. The trace distribution is then obtained by removing all states and internal actions from the associated probability space.

The forthcoming sections discuss each of the steps above in more detail.

Resolving nondeterministic choices in PAs

In order to understand what the behavior of a PA is like and how the nondeterministic choices in a PA are resolved, consider the channel in Figure 2.3 on page 34 again and recall that $snd(i)$ models the sending of a bit by a sender process, which corresponds to the receipt by the channel.

What can happen during the execution of this channel? Being in its start state ε, the channel may either receive a 0, a 1 or it might receive no bit at all. One of the fundamental aspects in the theory of PAs is that each of these possibilities may occur with a certain probability. Say that the probability on a 0 to arrive is q_0, on a 1 to arrive is q_1 and on no bit to arrive at all is $1 - q_0 - q_1$. Then the channel takes the transition $snd(0)$ with probability q_0. Similarly, it takes the transition $snd(1)$ with probability q_1 and remains in the state ε (forever) with $1 - q_0 - q_1$. In the latter case we say that the execution of the channel is interrupted.

Each choice for the values q_0 and q_1 in $[0, 1]$ yields a potential (and different) behavior of the channel. In this example, the probabilities naturally arise from a probabilistic environment (a sender process) that determines probabilistically whether to send a bit and which one. In general, we describe the resolution of the nondeterministic choices by an adversary.

Upon taking the transition that has been chosen probabilistically, the system determines its next state according to the target distribution of the transition chosen. In the example, the channel moves to the state 0 with probability $q_0 \cdot \frac{99}{100}$, to 1 with probability $q_1 \cdot \frac{99}{100}$ and stays in ε with the remaining probability mass $1 - q_0 \cdot \frac{99}{100} - q_1 \cdot \frac{99}{100}$. Here, we see that the probability to lose a bit is only determined exactly if we know how the nondeterminism in the system is resolved (i.e. if we know q_0 and q_1). Before resolving the nondeterministic choices, do not know the probability to lose a bit, we can only say that it is at most $\frac{1}{100}$.

After taking the transition, the procedure starts over in the new state: the channel makes a probabilistic choice between the outgoing transitions in the new state and an interruption. That is, in the states i, the channel has the choice between $rec(i)$ and an interruption; in the state ε there is a choice between $snd(0)$, $snd(1)$ and interruption. Obviously, these choices are not there if we are in ε as the result of an interruption. Moreover, when resolving the nondeterministic choice in ε, we do not have to take the same probabilities q_0 and q_1 as before: for instance, the environment may now send the bits with different probabilities. Moreover, the probabilities may be different depending on the bit that the channel previously received. Therefore the resolution of the nondeterminism can be history–dependent: it may not only depend on the current system state, but also on the path leading to that state.

Formally, the resolution of the nondeterministic choices in a PA is described by an *adversary* (also called *scheduler* or *policy*). In each state of the system, the adversary determines the next transition to be taken. The explanation above shows that we need adversaries that are

- *partial* i.e. may interrupt the execution at any time,
- *randomized* i.e. determine their choices randomly and
- *history–dependent* i.e. may base their choices not only on the current state, but also on

the path leading to that state.

This means that, given a finite path π ending in a state s, the adversary A schedules the transition $s \xrightarrow{a} \mu$ with probability $A(\pi)(a, \mu)$. The value $A(\pi)(\perp)$ is the probability on an interruption. We let our adversaries start from a fixed start state s_0[4].

Definition 2.3.6 Let s_0 be a start state of a PA $\mathcal{A}$. A *randomized, partial, history–dependent adversary* (or shortly an *adversary*) of $\mathcal{A}$ starting from s_0 is a function

$$A : Path^*(s_0, \mathcal{A}) \to \mathrm{Distr}(\mathrm{Act}_{\mathcal{A}} \times \mathrm{Distr}(S_{\mathcal{A}}) \cup \{\perp\})$$

such that if $A(\pi)(a, \mu) > 0$ then $last(\pi) \xrightarrow{a}_{\mathcal{A}} \mu$.

Example 2.3.7 Reconsider the lossy communication channel from Example 2.2.9. Let A_1 be the adversary that schedules the transition θ^1_{100} (i.e. sends a 1) whenever the system is in the state ε. Furthermore, A_1 schedules the $rec(i)$–action whenever the system is in i. Then A_1 is defined by

$$\begin{aligned}
&A_1(\pi)(snd(1), \mu^1_{100}) = 1, && \text{if } last(\pi) = \varepsilon\\
&A_1(\pi)(rec(i), \{\varepsilon \mapsto 1\}) = 1, && \text{if } last(\pi) = i
\end{aligned}$$

and 0 in all other cases. Obviously, in the second clause, only the case $i = 1$ is relevant, because the bit 0 is never sent. Nevertheless, we require the adversary also to be defined on paths containing an $snd(0)$ action, since this is technically simpler. Later, we will see that such paths are assigned probability 0.

The adversary A_2 schedules the transitions θ^0_{100}, θ^0_{200}, θ^1_{100} and θ^1_{200} each with probability $\frac{1}{4}$ whenever the system is in state ε . The $rec(i)$ action is taken with probability one if the system is in the state i. Then A_2 is given by

$$\begin{aligned}
&A_2(\pi)(snd(i), \mu^i_j) = \tfrac{1}{4}, && \text{if } last(\pi) = \varepsilon\\
&A_2(\pi)(rec(i), \{\varepsilon \mapsto 1\}) = 1, && \text{if } last(\pi) = i
\end{aligned}$$

for $i = 0, 1$, $j = 100, 200$ and 0 in all other cases.

The adversary A_3 corresponds to scheduling the transition θ^1_{100} in state ε with probability $\frac{1}{3}$, the transition θ^1_{200} with probability $\frac{1}{3}$, the transitions θ^0_{100} and θ^0_{200} with probability 0 and to interrupt the execution with probability $\frac{1}{3}$. Also, in state i, the probability of interruption is $\frac{1}{3}$. This adversary is defined by

$$\begin{aligned}
&A_3(\pi)(snd(1), \mu^1_{100}) = \tfrac{1}{3}, && \text{if } last(\pi) = \varepsilon\\
&A_3(\pi)(snd(1), \mu^1_{200}) = \tfrac{1}{3}, && \text{if } last(\pi) = \varepsilon\\
&A_3(\pi)(rec(i), \{\varepsilon \mapsto 1\}) = \tfrac{2}{3}, && \text{if } last(\pi) = i\\
&A_3(\pi)(\perp) = \tfrac{1}{3},
\end{aligned}$$

and 0 otherwise.

[4]Obviously, the forthcoming theory also applies to adversaries starting in non–start states, but we wish to consider the behavior generated from start states only.

Remark 2.3.8 An adversary A starting in a state s_0 can be described by a tree whose root is the state s_0, whose nodes are the finite paths in A and whose leaves are the sequences $\pi\perp$, where π is a path satisfying $A(\pi)(\perp) > 0$. The children of a node π are the finite paths $\pi a\mu t$, where $A(\pi)(a,\mu) > 0$ and $\mu(t) > 0$, and the sequence $\pi\perp$ if$A(\pi)(\perp) > 0$. The edge from π to $\pi a\mu t$ is labeled with the probability $A(\pi)(a,\mu)\cdot\mu(t)$ and the edge from π to $\pi\perp$ with the probability $A(\pi)(\perp)$. In fact, this tree is a cycle–free discrete time Markov chain.

By considering partial, history–dependent, randomized adversaries, the theory of PAs makes three fundamental choices for the behavior of PAs. We have motivated these choices by the channel NA already and below we give a more generic motivation of these decisions.

Partiality is already present in the nonprobabilistic case, where the execution of an NA may end in any state, even if there are transitions available in that state. Partiality is needed for compositionality results, both in the probabilistic and the nondeterministic case.

History dependence is also exactly the same as in the non–probabilistic case: each time the execution of an NA visits a certain state, it may take a different outgoing transition to leave that state.

Randomization has no counterpart in NAs. There are several arguments why we need randomized adversaries rather than deterministic ones.

- *Including all possible resolutions.* First of all, it is very natural to allow a nondeterministic choice to be resolved probabilistically. Nondeterminism is used if we do not wish to specify the factors that influence the choice. In particular, such factors can be probabilistic; there is no reason to assume that these are deterministic. Thus, randomized adversaries are needed to include all possible ways (including probabilistic ones) to resolve the nondeterministic choices.

- *Modeling probabilistic environments.* As we saw in the channel example, nondeterminism can model choices to be resolved by the environment. Since the environment may be probabilistic, randomized adversaries are needed to model the behavior of the PA in this environment.

- *Randomized algorithms: implementing a nondeterministic choice by a probabilistic choice*. The specification of a system often leaves room for several alternative implementations. Randomized algorithms usually implement their specifications by a probabilistic choice over those alternative implementations. By allowing randomized adversaries, the behavior of the randomized algorithm is included in the behavior of the specification, but this is not true when ranging over deterministic adversaries only. In Section 2.4 we will see that implementation relations for PAs are based on inclusion of external behavior. If we base the notion of behavior based on deterministic adversaries, then it is not possible to implement nondeterministic with randomized algorithms, unlike a notion of randomized adversaries [Alf97].

However, it has been proven [Alf99] that, if one is only interested in the minimal and maximal probability of a certain event, then it suffices to consider only deterministic adversaries.

The probability space associated to an adversary

Once the nondeterminism has been resolved by an adversary, we can study the probabilistic behavior of the system under this adversary. This is done via the associated probability space. The behavior generated by an adversary A is obtained by scheduling the transitions described by A and executing them until – possibly – A tells us to stop. The paths obtained in this way are the *maximal paths* in A, i.e. the infinite paths and the finite paths that have a positive probability on termination. Thus, the maximal paths represent the complete rather than partial behavior of A. Each maximal path can be assigned a probability. As we will see, the associated probability space assigns a probability to certain sets of maximal paths.

Throughout this section, let A be a randomized partial adversary for a PA $\mathcal{A}$.

Definition 2.3.9 A *path in* A is a finite or infinite path

$$\pi = s_0 a_1 \mu_1 s_1 a_2 \mu_2 \ldots$$

such that $A(s_0 a_1 \mu_1 s_1 \ldots a_i \mu_i)(a_{i+1}, \mu_{i+1}) > 0$ for all $0 \leq i < |\pi|$. The *maximal paths in* A are the infinite paths in A and the finite paths π in A where $A(\pi)(\perp) > 0$. Define $Path^{max}(A)$ as the set of maximal paths in A.

Note the difference between paths in a PA and those in an adversary. The former are a superset of the latter: compare Definitions 2.3.3 and 2.3.9.

For every finite path π, we can compute the probability $\mathbf{Q}^A(\pi)$ that a path generated by A starts with π. This probability is obtained by multiplying the probabilities that A actually schedules the transitions given by π with the probabilities that taking a transition actually yields the state specified by π. Note that the probability that the path generated by A is *exactly* π equals $\mathbf{Q}^A(\pi) \cdot A(\pi)(\perp)$.

Definition 2.3.10 Let $\mathcal{A}$ be a PA and let $s_0 \in S_{\mathcal{A}}$ be a state. Then we define the function $\mathbf{Q}^A : Path^*(s_0, \mathcal{A}) \to [0,1]$ inductively by

$$\mathbf{Q}^A(s_0) = 1 \text{ and } \mathbf{Q}^A(\pi a \mu t) = \mathbf{Q}^A(\pi) \cdot A(\pi)(a, \mu) \cdot \mu(t).$$

Example 2.3.11 Reconsider the adversaries A_1, A_2 and A_3 from the Example 2.3.7 The path $\pi = \langle \varepsilon, snd(1), \mu^1_{100}, \varepsilon, snd(1), \mu^1_{100}, 1\rangle$ is a path in A_1, A_2 and in A_3. It is assigned the following probabilities by the adversaries:

$$\mathbf{Q}^{A_1}(\pi) = 1 \cdot \tfrac{1}{100} \cdot 1 \cdot \tfrac{99}{100}, \quad \mathbf{Q}^{A_2}(\pi) = \tfrac{1}{4} \cdot \tfrac{1}{100} \cdot \tfrac{1}{4} \cdot \tfrac{99}{100} \quad \mathbf{Q}^{A_3}(\pi) = \tfrac{1}{3} \cdot \tfrac{1}{100} \cdot \tfrac{1}{3} \cdot \tfrac{99}{100}.$$

This path is maximal in A_3, but not in A_1 and A_2. Furthermore, the sequence $\langle \varepsilon$, $snd(0)$, μ^0_{100}, $\varepsilon\rangle$ is a path of the system in Figure 2.5. It is also a path of the adversary A_2, but not of the adversaries A_1 and A_3.

To study the probabilistic behavior generated by an adversary A, we associate a probability space to it. A probability space is a mathematical structure that assigns a probability to certain sets of (in this case) maximal paths such that the axioms of probability are respected (viz., the probability on the set of all events is 1; the probability on the complement of a set is one minus the probability on the set; and the probability of a countable, pairwise disjoint union of sets is the sum of the probabilities on the sets).

Note that we cannot describe the probabilistic behavior of an adversary by a probability *distribution*, assigning a probability to each element (in this case a maximal path), because this does not provide enough information. For instance, consider the adversaries A_1 and A_2 from Example 2.3.7. Their maximal paths are all infinite and both adversaries assign probability 0 to each infinite path. However, they differ on many sets of maximal paths, e.g. the probability that the first action in a path is $snd(1)$ equals 1 for A_1 and $\frac{1}{2}$ for A_2.

Definition 2.3.12 A *probability space* is a triple $(\Omega, \mathcal{F}, \mathbf{P})$, where

1. Ω is a set, called the *sample space*,
2. $\mathcal{F} \subseteq 2^{\Omega}$ is *σ-field*, i.e. a collection of subsets of Ω which is closed under countable[5] union and complement and which contains Ω,
3. $\mathbf{P} : \mathcal{F} \to [0, 1]$ is a *probability measure* on $\mathcal{F}$, which means that $\mathbf{P}[\Omega] = 1$ and for any countable collection $\{X_i\}_i$ of pairwise disjoint subsets in $\mathcal{F}$ we have $\mathbf{P}[\cup_i X_i] = \sum_i \mathbf{P}[X_i]$.

Note that it now follows that $\mathbf{P}[\emptyset] = 0$ and $\mathbf{P}[\Omega - X] = 1 - \mathbf{P}[X]$. It is also obvious that $\mathcal{F}$ is closed under intersection.

The idea behind the definition of a probability space is as follows. One can prove that it is not possible to assign a probability to each set and to respect the axioms of probability at the same time. Therefore, we collect those sets to which we can assign a probability into a σ–field $\mathcal{F}$. A σ–field is a collection of sets that contains the set Ω of all events and that is closed under complementation and countable union. The rationale behind this is that we can follow the axioms of probability. Thus, we can assign probability one to Ω and therefore $\Omega \in \mathcal{F}$. Moreover, if we assign probability $\mathbf{P}[X]$ to a set $X \in \mathcal{F}$, then we can also assign a probability to its complement, viz. $1 - \mathbf{P}[X]$. Therefore, the $\mathcal{F}$ is closed under complementation. Similarly, if we have a collection of pairwise disjoint sets $\{X_i\}_i$ which are all assigned a probability then $\mathbf{P}[\cup_i X_i] = \sum_i \mathbf{P}[X_i]$. Hence, the union can also be given a probability and therefore $\mathcal{F}$ is closed under countable unions.

The probability space associated to an adversary is generated from the sets C_π. Here C_π is the *cone above* π, the set containing all maximal paths that start with the finite path π. Since we know the probabilities on the set C_π – namely $\mathbf{Q}^A(\pi)$ – and we need to have a σ–field, we simply consider the smallest σ–field that contains these sets. A fundamental theorem from measure theory now states that, under the conditions met here, we can give a probability measure on all sets in $\mathcal{F}_A$ by specifying it on the sets C_π only, see for instance [Hal50] and [Seg95b].

Definition 2.3.13 The *probability space* associated to a partial adversary A starting in s_0 is the probability space given by

1. $\Omega_A = Path^{max}(A)$,
2. $\mathcal{F}_A$ is the smallest σ–field that contains the set $\{C_\pi \mid \pi \in Path^*(A)\}$, where $C_\pi = \{\pi' \in \Omega_A \mid \pi \sqsubseteq \pi'\}$ and $\sqsubseteq$ denotes the prefix relation on paths,
3. $\mathbf{P}_A$ is the unique measure on $\mathcal{F}_A$ such that $\mathbf{P}_A[C_\pi] = \mathbf{Q}^A(\pi)$ for all $\pi \in Path^*(s, \mathcal{A})$.

The fact that $(\Omega_A, \mathcal{F}_A, \mathbf{P}_A)$ is a probability space follows from standard measure theory arguments, see for instance [Hal50] or [Coh80]. Note that Ω_A and $\mathcal{F}_A$ do not depend on A but only on $\mathcal{A}$, and that $\mathbf{P}_A$ is fully determined by the function $\mathbf{Q}^A$.

[5] In our terminology, countable objects include finite ones.

Note that the cone C_π is contained in $\mathcal{F}_A$ for every finite path in A, but that the cone itself contains finite and infinite *maximal* paths. The reason for requiring π to be a path in the adversary A rather than a path in $\mathcal{A}$ is that in this way $\mathcal{F}_A$ is generated by countably many cones, even if the set of states or actions of $\mathcal{A}$ is uncountable (as is the case for PTAs).

Furthermore, it is not difficult to see that if the set Ω_A is countable, then $\mathcal{F}_A$ is simply the power set of A. However, if Ω_A is uncountable (and this is the case for which probability spaces have been designed), then the set $\mathcal{F}_A$ is quite complicated — probably more complicated than it seems at first sight. Obviously, this collection can be generated inductively by starting from the cones and by taking complementation and countable unions. This requires ordinal induction, rather than ordinary induction. Moreover, the construction of a set not being in $\mathcal{F}_A$ crucially depends on the axiom of choice. The branch of mathematics that is concerned with probability spaces and, more general, measure spaces is called *measure theory*.

The following example presents a few sets that are contained in $\mathcal{F}_A$.

Example 2.3.14 The collection $\mathcal{F}_A$ contains many sets of traces that occur in practice, or that come easily to one's mind. For instance, the set of paths containing at most three elements a is given by

$$\bigcup_{\rho \in X} C_\rho,$$

where $X = \{\alpha \in \mathit{Path}^*(\mathcal{A}) \mid \alpha \text{ contains at most three } a\text{'s}\}$. Since X is countable, the set above is an element of $\mathcal{F}_A$. The set containing the single infinite path π equals

$$\bigcap_{\rho \sqsubseteq \pi, \rho \neq \pi} C_\rho.$$

Example 2.3.15 Consider the adversary A_2 from Example 2.3.7. Then Ω_{A_2} is just the sets of all infinite paths. The set C_π contains the infinite paths extending the path π and $\mathcal{F}_{A_2}$ is the smallest σ–algebra containing those cones. Some values of the function $\mathbf{P}^{A_2}$ are

$$\mathbf{P}^{A_2}[C_{\langle \varepsilon\ snd(0)\ \mu^1_{100}\ 1\rangle}] = \mathbf{Q}^{A_2}(\langle \varepsilon snd(0) \mu^1_{100} 1\rangle) = \tfrac{1}{4} \cdot \tfrac{99}{100}.$$

and

$$\begin{aligned}
&\mathbf{P}^{A_2}[\text{a max. path generated by } A \text{ contains at most three actions } snd(0)] \leq \\
&\mathbf{P}^{A_2}[\text{a max. path generated by } A \text{ contains finitely many actions } snd(0)] = \\
&\mathbf{P}^{A_2}\left[\bigcup_i \text{a max. path generated by } A \text{ contains no } snd(0) \text{ after position } i\right] = \\
&\lim_{i\to\infty} \mathbf{P}^{A_2}[\text{a max. path generated by } A \text{ contains no } snd(0) \text{ after position } i] = \\
&\lim_{i\to\infty} 0 = 0.
\end{aligned}$$

The third step in this computation follows easily from the definition of probability space.

The trace distribution of an adversary

Now, the trace distribution H generated by an adversary A is obtained by removing all states and internal actions from the probability space associated to A. The probability on a set of traces X is now the probability $\mathbf{P}_H[X]$ that A generates a maximal path with a trace in X.

Definition 2.3.16 The *trace distribution* H of an adversary A, denoted by $\mathit{trdistr}(A)$, is the probability space given by

1. $\Omega_H = \mathrm{Act}_{\mathcal{A}}{}^* \cup \mathrm{Act}_{\mathcal{A}}{}^\infty$,
2. $\mathcal{F}_H$ is the smallest σ-field that contains the sets $\{C_\beta \mid \beta \in \mathrm{Act}_{\mathcal{A}}{}^*\}$, where $C_\beta = \{\beta' \in \Omega_H \mid \beta \sqsubseteq \beta'\}$,
3. $\mathbf{P}_H$ is given by $\mathbf{P}_H[X] = \mathbf{P}_A[\{\pi \in \Omega_A \mid \mathit{trace}(\pi) \in X\}]$ for all $X \in \mathcal{F}_H$.

Standard measure theory arguments [Hal50] together with the fact that the function trace is measurable ensure that $(\Omega_H, \mathcal{F}_H, \mathbf{P}_H)$ is well-defined. We denote the set of trace distributions of $\mathcal{A}$ by $\mathit{trdistr}(\mathcal{A})$.

Example 2.3.17 Consider the trace distribution H of adversary A_2 from Example 2.3.7 again. The sets Ω_H and $\mathcal{F}_H$ need no further explanation. The probability on the set $\{\pi \mid \mathit{trace}(\pi) = \mathit{snd}(1)\}$, i.e. the maximal paths whose trace starts with $\mathit{snd}(1)$, is given as follows.

$$\begin{aligned}
\mathbf{P}_H[C_{\mathit{snd}(1)}] &= \mathbf{P}_{A_2}[C_{\langle \varepsilon\, \mathit{snd}(1) \mu^1_{100} 1\rangle}] \\
&\quad + \mathbf{P}_{A_2}[C_{\langle \varepsilon\, \mathit{snd}(1) \mu^1_{100} \varepsilon\rangle}] \\
&\quad + \mathbf{P}_{A_2}[C_{\langle \varepsilon\, \mathit{snd}(1) \mu^1_{200} 1\rangle}] \\
&\quad + \mathbf{P}_{A_2}[C_{\langle \varepsilon\, \mathit{snd}(1) \mu^1_{200} \varepsilon\rangle}] \\
&= \tfrac{1}{4} \cdot \tfrac{1}{100} + \tfrac{1}{4} \cdot \tfrac{99}{100} + \tfrac{1}{4} \cdot \tfrac{1}{200} + \tfrac{1}{4} \cdot \tfrac{199}{200} = \tfrac{1}{2}
\end{aligned}$$

2.3.3 Alternative Interpretations for PAs

The theory of PAs takes the standpoint that the behavior of a PA is obtained by resolving the nondeterministic choices first and by then resolving the probabilistic choices. This view is widely adopted, but not the only one. We briefly discuss two different interpretations of PAs.

The work [NCI99] resolves the probabilistic choices first and then the nondeterministic ones. This means that, for each outgoing transition of a state, one of the probabilistic alternatives in the target of the transition is chosen. Since the resolution of the probabilistic choices can be history–dependent, this yields a tree. Then a probability space can be associated to this process along the same lines as we did: The σ–field is generated by a collection of cones, which now contain trees extending a certain finite tree and the probability measure extends the probability on the cones. (The latter are obtained by multiplying the probabilities on the alternatives chosen in this tree.)

This approach leads to a completely different notion of external behavior of a PA, which assigns the same behavior to certain PAs with different trace distributions and which also assigns a different behavior to certain PAs with the same trace distribution. Therefore, this approach is considered controversial.

Secondly, the interpretation of Andova [And99b, And99a] is that the probabilistic resolution is internal to a process and therefore one cannot say when exactly the nondeterministic choices are resolved until one sees an action of the system. Andova works in the alternating model and, although her work is concerned more with bisimulation relations than with trace distributions, one can say that her approach differs on the probabilistic states, but boils down to the same for the nondeterministic states.

2.4 Implementation Relations for PAs

A common approach in verification of concurrent systems is to describe both the implementation of a system and its specification by automata. An *implementation relation* then expresses when one automaton is a correct implementation of another. There are many different implementation relations around for various types of automata, including timed, hybrid and probabilistic ones. For NAs, the *trace inclusion* relation, denoted by $\sqsubseteq_{\mathrm{TR}}$, is often used. This relation defines $\mathcal{A}$ to be a correct implementation of $\mathcal{B}$ if and only if $trace(\mathcal{A}) \subseteq trace(\mathcal{B})$. Trace inclusion is one of the simplest implementation relations and many others are based on it. Trace inclusion preserves *safety properties*, i.e. if $\mathcal{A} \sqsubseteq_{\mathrm{TR}} \mathcal{B}$ then $\mathcal{A}$ meets all safety properties imposed by the specification $\mathcal{B}$. Safety properties typically state that “some bad thing never happens,” for instance, that an error never occurs or that the temperature in a microwave oven remains within the safety bounds.

There are two properties, viz. transitivity and substitutivity, which are crucial for the applicability of an implementation relation. These are important, because they allow a complex verification or implementation task to be chopped up in smaller ones. Recall that a relation $\sqsubseteq$ is *transitive* if $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$ and $\mathcal{A}_2 \sqsubseteq \mathcal{A}_3$ imply $\mathcal{A}_1 \sqsubseteq \mathcal{A}_3$. A relation $\sqsubseteq$ is *substitutive* if $\mathcal{A}_1 \sqsubseteq \mathcal{B}_1$ and $\mathcal{A}_2 \sqsubseteq \mathcal{B}_2$ imply $\mathcal{A}_1 \parallel \mathcal{A}_2 \sqsubseteq \mathcal{B}_1 \parallel \mathcal{B}_2$, where $\parallel$ is the parallel composition operator introduced in Definition 2.2.14. Transitivity is essential because it allows for hierarchical design and verification. This means that from a specification $\mathcal{B}$ one derives an implementation $\mathcal{A}$ via several intermediate specifications $\mathcal{I}_1, \mathcal{I}_2, \ldots \mathcal{I}_n$, which are successively more operational. If one ensures that $\mathcal{A} \sqsubseteq \mathcal{I}_n \sqsubseteq \mathcal{I}_{n-1} \sqsubseteq \ldots \mathcal{I}_1 \sqsubseteq \mathcal{B}$, then transitivity ensures that $\mathcal{A} \sqsubseteq \mathcal{B}$. Substitutivity is crucial because it supports compositional design and verification. This means that the correctness of a composite system follows from the correctness of its components. Hence, we can implement a(n intermediate) specification $\mathcal{B}_1 \parallel \mathcal{B}_2$ by implementing $\mathcal{B}_1$ as $\mathcal{A}_1$ and $\mathcal{B}_2$ as $\mathcal{A}_2$. If the implementation was done correctly (that is, $\mathcal{A}_1 \sqsubseteq \mathcal{B}_1$ and $\mathcal{A}_2 \sqsubseteq \mathcal{B}_2$), then a substitutive relation guarantees $\mathcal{A}_1 \parallel \mathcal{A}_2 \sqsubseteq \mathcal{B}_1 \parallel \mathcal{B}_2$, that is, the whole system is correct.

Since trace distributions are the natural counterparts of traces, one might propose trace distribution inclusion $\sqsubseteq_{\mathrm{TD}}$ as an implementation relation for PAs. The trace distribution equivalence $\equiv_{\mathrm{TD}}$ expresses that two systems have the same external behavior.

Definition 2.4.1 Define the relation $\sqsubseteq_{\mathrm{TD}}$ on PAs by $\mathcal{A} \sqsubseteq_{\mathrm{TD}} \mathcal{B}$ if and only if $trdistr(\mathcal{A}) \subseteq trdistr(\mathcal{B})$. Furthermore, define $\equiv_{\mathrm{TD}}$ by $\mathcal{A} \equiv_{\mathrm{TD}} \mathcal{B}$ if and only if $trdistr(\mathcal{A}) = trdistr(\mathcal{B})$.

It is easy to see that $\sqsubseteq_{\mathrm{TR}}$ and $\sqsubseteq_{\mathrm{TD}}$ are both transitive. However, $\sqsubseteq_{\mathrm{TD}}$ is not substitutive as is shown in the following example. This example is an adaptation from an example by Segala [Seg95b] and is the result of a discussion with Segala. We find this example here more convincing, because the adversaries used here are very natural and, unlike the ones in [Seg95b] do not have unexpected properties, (c.f. Example 2.4.4).

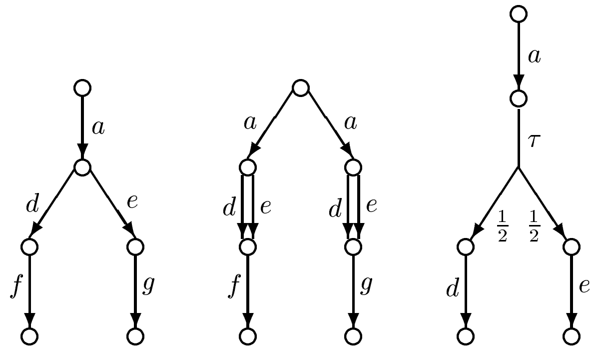


Figure 2.10: The automata $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{B}$ showing that trace distribution inclusion is not compositional

Example 2.4.2 Consider the PAs $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{B}$ in Figure 2.10. It is not difficult to see that $\mathcal{A}_1 \sqsubseteq_{\mathrm{TD}} \mathcal{A}_2$. However, $\mathcal{A}_1 \parallel \mathcal{B} \not\sqsubseteq_{\mathrm{TD}} \mathcal{A}_2 \parallel \mathcal{B}$. The automata $\mathcal{A}_1 \parallel \mathcal{B}$ and $\mathcal{A}_2 \parallel \mathcal{B}$ are represented in Figure 2.11.

Now, consider the adversary A_1 of $\mathcal{A}_1 \parallel \mathcal{B}$ that, in each state having an outgoing transition, schedules this unique outgoing transition with probability one. (The tree representing this adversary has the same structure as the PA $\mathcal{A}_1 \parallel \mathcal{B}$.) It is not difficult to see that there is no adversary of $\mathcal{A}_2 \parallel \mathcal{B}$ with the same trace distribution $H_1 = \mathit{trdistr}(A_1)$: assume that there is one, say A_2. Let $H_2 = \mathit{trdistr}(A_2)$. As $\mathbf{P}_{H_1}[\{adf\}] = \frac{1}{2}$, A_2 should schedule the leftmost transition of $\mathcal{A}_2 \parallel \mathcal{B}$ with probability one. But then $\mathbf{P}_{H_2}[\{aeg\}] = 0$, whereas $\mathbf{P}_{H_1}[\{aeg\}] = \frac{1}{2}$.

Three aspects in the example above are worth noticing. Firstly, when considered as NAs, then we have $\mathcal{A}_1 \sqsubseteq_{\mathrm{TR}} \mathcal{A}_2$. Thus, probabilistic environments (modeled as a PA) can distinguish more than nonprobabilistic environments, even if the automata considered do not contain probabilistic choices.

Secondly, the distinguishing PA $\mathcal{B}$ is very natural, there is nothing tricky going on: The PA $\mathcal{B}$ takes an a–transition, then it decides via an internal probabilistic transition whether to take a d or an e transition.

Nonsubstitutivity seems to be a fundamental property of trace distribution inclusion. Small adaptations to the theory, such as enforcing an I/O distinction on PAs, do not seem to work. In fact, the previous example can easily be transformed into an example for I/O automata.

Since substitutivity is essential in verification of real–life applications, we need a notion of implementation for PAs that is preserved under parallel composition. Segala proposes to simply consider the coarsest (i.e. the largest) precongruence contained in $\sqsubseteq_{\mathrm{TD}}$, denoted by $\sqsubseteq_{\mathrm{PTD}}$. Chapter 5 presents several simulation relations for PAs, which can be used to prove that $\mathcal{A} \sqsubseteq_{\mathrm{PTD}} \mathcal{B}$ for given PAs $\mathcal{A}$ and $\mathcal{B}$. An important result is the alternative characterization of $\sqsubseteq_{\mathrm{PTD}}$ by the principal context $\mathcal{C}$.

Theorem 2.4.3 ([Seg95b]) *Let $\mathcal{C}$ be the PA shown in Figure 2.12, where we suppose that the actions pleft and pright are fresh. Then $\mathcal{A} \sqsubseteq_{\mathrm{PTD}} \mathcal{B}$ iff $\mathcal{A} \parallel \mathcal{C} \sqsubseteq_{\mathrm{TD}} \mathcal{B} \parallel \mathcal{C}$.*

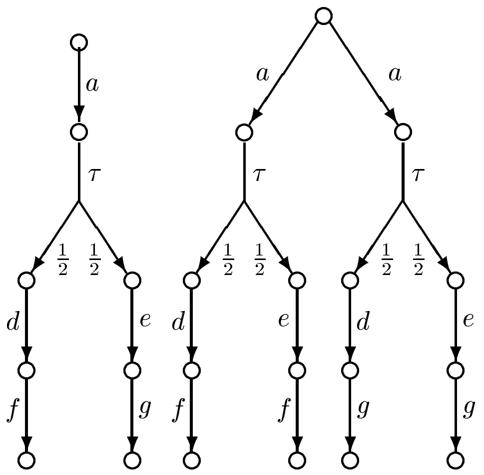


Figure 2.11: The PAs $\mathcal{A}_1 \parallel \mathcal{B}$ and $\mathcal{A}_2 \parallel \mathcal{B}$, showing that trace distribution inclusion is not compositional

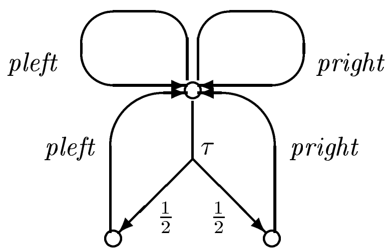


Figure 2.12: Principal context

The relation $\sqsubseteq_{\mathrm{PTD}}$ preserves probabilistic safety properties. These are properties expressing that for a given probability p, "some bad thing happens with a probability smaller than p." For instance, the probability that an error occurs is smaller that 10^{-10} or the probability that the temperature of the microwave is 1% too high is smaller than $\frac{1}{10}$. Moreover, it is transitive and substitutive and therefore suitable as an implementation relation.

One can prove that $\mathcal{A} \sqsubseteq_{\mathrm{TR}} \mathcal{B}$ via simulation relations. Section 2.5 presents two simple kinds of simulations for PAs and Chapter 5 is entirely devoted to simulation and bisimulation relations.

2.4.1 Miscellaneous Remarks on Trace Distributions

This section discusses several technical points with respect to the definition of trace distribution and the relations $\sqsubseteq_{\mathrm{TD}}$ and $\sqsubseteq_{\mathrm{PTD}}$. The reader may skip this section and still understand the rest of this thesis.

The definition of trace distribution

In his definition of trace distribution, Segala takes the completion of the probability space $(\Omega_H, \mathcal{F}_H, \mathbf{P}_H)$ that we used in our definition. The completion is a measure theoretic operation that yields a measure space with several nice mathematical properties. Both conceptually and technically, there is no problem at all in dealing with the definition we gave.

As observed by Segala, the motivation for not taking $trace(\Omega_A)$ as a sample space Ω_H, but dealing with $V_A{}^* \cup V_A{}^\infty$ is that the latter avoids distinguishing between two trace distributions that are the same except that one has more elements whose total probability mass is 0.

However, one might wonder why the σ–field $\mathcal{F}_H$ of a trace distribution is defined as the σ–field generated by the cones and not as the collection of pre–images of sets in $\mathcal{F}_A$. The latter is the set $\{X \in \Omega \mid trace^{-1}(X) \in \mathcal{F}_A\}$. In general, this set contains more elements than $\mathcal{F}_A$ and having more sets whose probability is defined might be useful. However, the set $\mathcal{F}_H$ might have nicer mathematical properties than the set of pre–images.

Compositionality and trace distribution inclusion

Example 2.4.2 demonstrated that $\sqsubseteq_{\mathrm{TD}}$ is not compositional. Therefore, Segala proposed $\sqsubseteq_{\mathrm{PTD}}$ as an implementation relation for PAs. An alternative would be to keep the implementation relation as it is, but to change the parallel composition operator. We believe that the latter is not the right thing to do. Although the parallel composition operator for PAs might have some unexpected properties, we believe it is the implementation relation which causes the troubles with substitutivity. One of these unexpected properties is shown in the following example. It presents an adversary of PAs consisting of two components, where one component looks into the state of a the other. That is, the former component bases its decision on the other one's state, whereas states are supposed internal to a process.

Example 2.4.4 Consider the systems $\mathcal{A}_1$ and $\mathcal{B}$ in Figure 2.10 and the adversary in Figure 2.13. This adversary has quite much power: depending on the state of $\mathcal{B}$, it decides to schedule an f or a g transition in $\mathcal{A}_1$. One might wonder whether this is reasonable, because $\mathcal{A}_1$ and $\mathcal{B}$ are independent systems, whose state space is not supposed to be visible for the environment. On the other hand, one might argue that an adversary can really be the worst

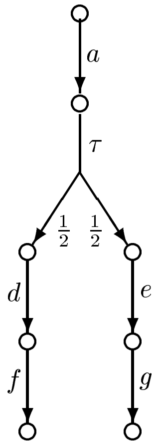


Figure 2.13: An adversary of $\mathcal{A}_1 \parallel \mathcal{B}$ with much power

devil ever, which has knowledge about the entire world and even about dependencies that are unknown to us.

Moreover, also in the nondeterministic case, we can define a scheduler which schedules an a or a b transition in one component, depending on the state of another component. The difference, however, is that in the nondeterministic case, one only sees one trace at the same time (i.e. each behavior corresponds to a single trace) whereas in the probabilistic case one sees many traces at the same time "in superposition," (i.e. each behavior is a trace distribution consisting of several traces with different probabilities).

We believe that, in order to obtain a substitutivity, the implementation relation should be changed, rather than the notion of parallel composition. The reason for this is that the adversary in Example 2.4.2 does not use the strange kind of information as the scheduler in the example above: in order to determine the next step to be taken in a component, it only uses observable information from the other component. So, even when reasonable adversaries are used, trace distribution inclusion is not a precongruence.

The recent work [AHJ01] provides a compositional notion of behavior for probabilistic systems in a variable based setting, which is essentially different ours.

Trace distributions versus traces

Automata can be interpreted as probabilistic automata by simply considering the target state of a transition as a Dirac distribution, see Remark 2.2.8. Then the natural question arises whether two trace equivalent NAs Surprisingly, this is not the case. For infinitely branching NAs, that is, for NAs that can have infinitely many transitions leaving the same state, this result fails.

For finitely branching NAs, the result follows easily by combining the Induction Approximation principle (Theorem 4.3.1) and Theorem 5.3.24.

We conjecture that the result also holds for countably branching PAs. What remains then are the uncountably branching PAs and it is not too much surprising that the combination of uncountably branching and discrete probabilities does not work: discrete probabilities are tailored for countable structures. For uncountable ones, richer structures, notably probability spaces, have been developed. However, it is important to note that uncountably branching PAs are needed to encode real–time into PAs.

Theorem 2.4.5 *Let $\mathcal{A}$ and $\mathcal{B}$ be automata, which we consider as PAs. If $\mathcal{B}$ is finitely branching, then*

$$\mathcal{A} \sqsubseteq_{\mathrm{TD}} \mathcal{B} \iff \mathcal{A} \sqsubseteq_{\mathrm{TR}} \mathcal{B}.$$

PROOF: (sketch) In this proof, we use the terminology and results from Chapters 5 and 4.

$\Longrightarrow$ Immediate.

$\Longleftarrow$ Assume $\mathcal{A} \sqsubseteq_{\mathrm{TR}} \mathcal{B}$. Since $\mathcal{B}$ is finitely branching, it follows from the Induction Approximation Principle that, in order to prove $\mathcal{A} \sqsubseteq_{\mathrm{TD}} \mathcal{B}$, it suffices to prove that each finite trace distribution of $\mathcal{A}$ is also one of $\mathcal{B}$. Thus, let H be a finite trace distribution of $\mathcal{A}$ and let E be the adversary generating H. By Theorem 5.3.24, we can write E as a convex combination of deterministic adversaries. Let $E = \sum_i p_i \cdot D_i$, where D_i is a deterministic adversary and $p_i \in [0, 1]$. Deterministic adversaries generate trace distributions that are isomorphic to finite traces. Therefore, we can find finite adversaries D'_i of $\mathcal{B}$ with $trdistr(D_i) = trdistr(D'_i)$. It now follows easily that $D' = \sum_i p_i \cdot D'_i$ is an adversary of $\mathcal{B}$ such that $trdistr(D') = H$.

□

2.5 Step Refinements and Hyperstep Refinements

This section is concerned with proof methods for $\sqsubseteq_{\mathrm{PTD}}$ based on simulation techniques. In the nonprobabilistic case, these techniques have been successfully applied in the verification of a number of systems [DGRV00]. The idea is that simulation relations reduce global reasoning about adversaries and trace distributions (in NAs of executions and traces) to local reasoning in terms of states and transitions. Since the latter is far more easy, the verification effort is reduced significantly.

This section presents two simple simulations for PAs, namely step refinements and hyperstep refinement. They first appeared in [SV99b] and simplify the more complex probabilistic simulation relations introduced in [SL95, Seg95b]. Those ones, and other simulations are studied in Chapter 5.

Probabilistic step refinements

The simplest form of simulations between probabilistic automata that we consider are the probabilistic step refinements. These are mappings from the states of one automaton to the states of another automaton that preserve the initial states and the probabilistic transitions. For the latter, both the external actions and the probabilistic information have to be preserved. We first need an auxiliary definition.

Definition 2.5.1 Let X and Y be sets, $\mu \in \mathrm{Distr}(X)$ and $f : X \to Y$. The *image* of μ under f, notation $f_*(\mu)$, is the probability distribution $\nu \in \mathrm{Distr}(Y)$ satisfying $\nu(y) = \sum_{x \in f^{-1}(y)} \mu(x)$.

Informally, the image distribution $f_*(\mu)$ yields the probability distribution on Y that arises by first picking an element from X according to the distribution μ and by then applying f. If several elements of X are mapped to the same element in Y, then one adds their probabilities.

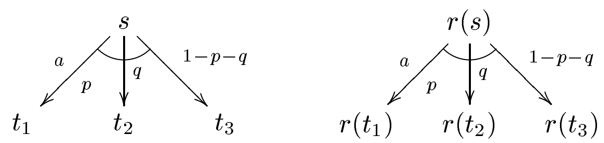
Definition 2.5.2 Let $\mathcal{A}$ and $\mathcal{B}$ be two PAs with $V_\mathcal{A} = V_\mathcal{B}$. A *probabilistic step refinement* from $\mathcal{A}$ to $\mathcal{B}$ is a function $r : S_\mathcal{A} \to S_\mathcal{B}$ such that:

1. for all $s \in S^0_\mathcal{A}, r(s) \in S^0_\mathcal{B}$;
2. for all steps $s \xrightarrow{a}_\mathcal{A} \mu$ with $s \in \mathit{reach}_\mathcal{A}$, at least one of the following conditions holds:
   (a) $r(s) \xrightarrow{a}_\mathcal{B} r_*(\mu)$, or
   (b) $a \in I_\mathcal{A} \wedge r(s) \xrightarrow{b}_\mathcal{B} r_*(\mu)$, for some $b \in I_\mathcal{B}$, or
   (c) $a \in I_\mathcal{A} \wedge r_*(\mu) = \{r(s) \mapsto 1\}$.

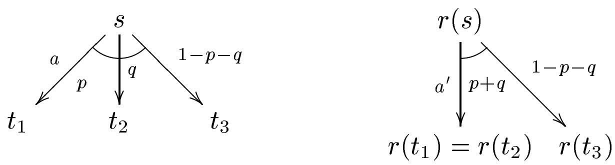
We write $\mathcal{A} \sqsubseteq_{\mathrm{PSR}} \mathcal{B}$ if there is a probabilistic step refinement from $\mathcal{A}$ to $\mathcal{B}$. Note that condition 2(c) is equivalent to $a \in I_\mathcal{A} \wedge \forall s'[\mu(s') > 0 \implies r(s') = r(s)]$.

Example 2.5.3 The following diagrams illustrate three typical situations that may occur if r is a probabilistic step refinement from $\mathcal{A}$ to $\mathcal{B}$. The transitions on the left are steps of the probabilistic automaton $\mathcal{A}$, those on the right of $\mathcal{B}$.

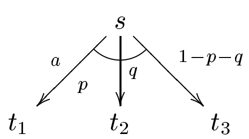
Condition 2a and $r(t_1) \neq r(t_2) \neq r(t_3) \neq r(t_1)$:



Condition 2b, $a \in I_\mathcal{A}$, $a' \in I_\mathcal{B}$ and $r(t_1) = r(t_2) \neq r(t_3)$:



Condition 2c and $a \in I_\mathcal{A}$:



$r(s) = r(t_1) = r(t_2) = r(t_3)$

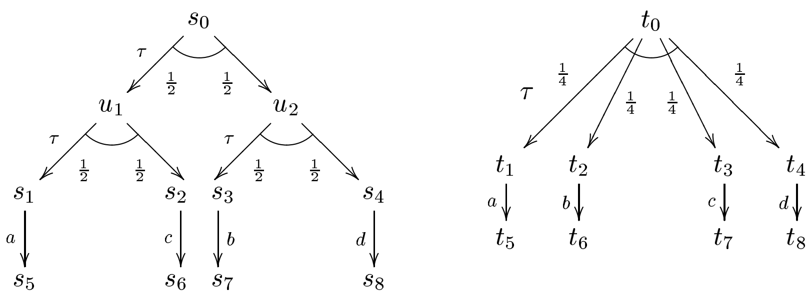


Figure 2.14: The PAs $\mathcal{A}_1$ and $\mathcal{A}_2$.

Probabilistic hyperstep refinements

Probabilistic hyperstep refinements generalize the probabilistic step refinements introduced above. Both step refinements and hyperstep refinements are mappings that preserve initial states and probabilistic transitions in some sense. Whereas probabilistic step refinement map states to states, probabilistic hyperstep refinements map states to distributions over states. This requires a more involved notion of preserving transitions, which is formalized by lifting both the hyperstep refinement and the transition relation. Unlike step refinements, are hyperstep refinements able to contract two subsequent probabilistic choices into a single one, as is shown by the following example (c.f. [Seg95b], Example 8.5.1).

Example 2.5.4 Consider the following PAs $\mathcal{A}_1$ and $\mathcal{A}_2$ in Figure 2.14. Both automata have the same trace distributions: $\mathcal{A}_2$ just contracts the first two τ–transitions of $\mathcal{A}_1$. However, there does not exist a probabilistic step refinement from the first to the second. In order to see this, examine the state u_1. We cannot relate this state to t_0 because then we loose the information that we have chosen for the future a and c actions already. However, u_1 cannot be related to t_1 or t_2 either, because unlike t_1 or t_2 does the state u_1 provide a probabilistic choice between the future a and b actions. The solution provided by the concept of probabilistic hyperstep refinement is to relate u_1 to t_1 and to t_2 with probability $\frac{1}{2}$ each, i.e. to the distribution $\{t_1 \mapsto \frac{1}{2}, t_2 \mapsto \frac{1}{2}\}$.

Definition 2.5.5 Let X, Y be sets and $R \subseteq X \times \mathrm{Distr}(Y)$. The *lifting* of R is the relation $R_{**} \subseteq \mathrm{Distr}(X) \times \mathrm{Distr}(Y)$ given by: $(\mu, \nu) \in R_{**}$ if and only if there is a choice function $r : \mathit{support}(\mu) \to \mathrm{Distr}(Y)$ for R, i.e., a function such that $(x, r(x)) \in R$ for all $x \in \mathit{support}(\mu)$, satisfying

$$\nu(y) = \sum_{x \in \mathrm{supp}(\mu)} \mu(x) \cdot r(x)(y).$$

The idea is that we obtain ν by choosing the probability distribution $r(x)$ with probability $\mu(x)$.

Note that R_{**} is a function if R is so.

Example 2.5.6 Given a probabilistic automaton $\mathcal{A}$ and an action $a \in \mathrm{Act}_{\mathcal{A}}$, we can lift the relation $\xrightarrow{a}$ over $S_{\mathcal{A}} \times \mathrm{Distr}(S_{\mathcal{A}})$ to the relation $\xrightarrow{a}_{**}$ over $\mathrm{Distr}(S_{\mathcal{A}}) \times \mathrm{Distr}(S_{\mathcal{A}})$. For instance, if $s_1 \xrightarrow{a} \mu_1$, $s_2 \xrightarrow{a} \mu_2$ and $s_1 \neq s_2$, then

$$\{s_1 \mapsto \tfrac{1}{3}, s_2 \mapsto \tfrac{2}{3}\} \xrightarrow{a}_{**} \tfrac{1}{3} \cdot \mu_1 + \tfrac{2}{3} \cdot \mu_2.$$

The idea is as follows. Assume that $s_1 \xrightarrow{a} \mu_1$ and $s_2 \xrightarrow{a} \mu_2$ are transitions in $\mathcal{A}$ and that the probability to be in s_1 is $\frac{1}{3}$ and the probability to be in s_2 is $\frac{2}{3}$. Then, after performing the a–transitions mentioned, we choose the next state according to μ_1 with probability $\frac{1}{3}$ and according to μ_2 with probability $\frac{2}{3}$. This yields the distribution $\frac{1}{3} \cdot \mu_1 + \frac{2}{3} \cdot \mu_2$. If there is another a–transition leaving from s_2, say $s_2 \xrightarrow{a} \nu$, then we may take this step instead of $s_2 \xrightarrow{a} \mu_2$. Hence

$$\{s_1 \mapsto \tfrac{1}{3}, s_2 \mapsto \tfrac{2}{3}\} \xrightarrow{a}_{**} \tfrac{1}{3} \cdot \mu_1 + \tfrac{2}{3} \cdot \nu.$$

Note that we do *not* have

$$\{s_1 \mapsto \tfrac{1}{3}, s_2 \mapsto \tfrac{2}{3}\} \xrightarrow{a}_{**} \tfrac{1}{3} \cdot \mu_1 + \tfrac{1}{3} \cdot \mu_2 + \tfrac{1}{3} \cdot \nu.$$

Obviously, we have $s \mapsto 1 \xrightarrow{a} \mu$ whenever $s \mapsto 1 \xrightarrow{a} \mu$.

Example 2.5.7 For the PA $\mathcal{A}_1$ in Figure 2.14, we have that

$$\begin{aligned}\{u_1 \mapsto \tfrac{1}{2}, u_2 \mapsto \tfrac{1}{2}\} \xrightarrow{\tau}_{**} \tfrac{1}{2} \cdot \{s_1 \mapsto \tfrac{1}{2}, s_2 \mapsto \tfrac{1}{2}\} + \tfrac{1}{2} \cdot \{s_3 \mapsto \tfrac{1}{2}, s_4 \mapsto \tfrac{1}{2}\}\\ = \{s_1 \mapsto \tfrac{1}{4}, s_2 \mapsto \tfrac{1}{4}, s_3 \mapsto \tfrac{1}{4}, s_4 \mapsto \tfrac{1}{4}\}\end{aligned}$$

Definition 2.5.8 Let $\mathcal{A}$ and $\mathcal{B}$ be probabilistic automaton with $V_{\mathcal{A}} = V_{\mathcal{B}}$. A *probabilistic hyperstep refinement* from $\mathcal{A}$ to $\mathcal{B}$ is a function $h : S_{\mathcal{A}} \to \mathrm{Distr}(S_{\mathcal{B}})$ such that:

1. for all $s \in S^0_{\mathcal{A}}$, $h(s) = \{s' \mapsto 1\}$ for some $s' \in S^0_{\mathcal{B}}$;
2. for all steps $s \xrightarrow{a}_{\mathcal{A}} \mu$ with $s \in \mathit{reach}_{\mathcal{A}}$, at least one of the following conditions holds:
   (a) $h(s) \xrightarrow{a}_{\mathcal{B}**} h_{**}(\mu)$, or
   (b) $a \in I_{\mathcal{A}} \wedge h(s) \xrightarrow{b}_{\mathcal{B}**} h_{**}(\mu)$, for some $b \in I_{\mathcal{B}}$, or
   (c) $a \in I_{\mathcal{A}} \wedge h(s) = h_{**}(\mu)$.

Write $\mathcal{A} \sqsubseteq_{\mathrm{PHSR}} \mathcal{B}$ if there is a probabilistic hyperstep refinement from $\mathcal{A}$ to $\mathcal{B}$.

Example 2.5.9 Consider the function h given by

$$\begin{aligned}&s_i \mapsto t_i, && \text{for all } i\\ &u_i \mapsto \{t_1 \mapsto \tfrac{1}{2}, t_2 \mapsto \tfrac{1}{2}\} && \text{for } i = 1, 2.\end{aligned}$$

Then we have for the lifting h_{**} that

$$\begin{aligned}\{s_i \mapsto 1\} &\mapsto \{t_i \mapsto 1\}, && \text{for all } i\\ \{u_1 \mapsto 1\} &\mapsto \{t_1 \mapsto \tfrac{1}{2}, t_2 \mapsto \tfrac{1}{2}\},\\ \{u_2 \mapsto 1\} &\mapsto \{t_3 \mapsto \tfrac{1}{2}, t_4 \mapsto \tfrac{1}{2}\},\\ \{s_i \mapsto \tfrac{1}{2}, s_{i+1} \mapsto \tfrac{1}{2}\} &\mapsto \{t_i \mapsto \tfrac{1}{2}, t_{i+1} \mapsto \tfrac{1}{2}\} && \text{for } i = 1, 3\\ \{u_1 \mapsto \tfrac{1}{2}, u_2 \mapsto \tfrac{1}{2}\} &\mapsto \tfrac{1}{2} \cdot \{t_1 \mapsto \tfrac{1}{2}, t_2 \mapsto \tfrac{1}{2}\} + \tfrac{1}{2} \cdot \{t_3 \mapsto \tfrac{1}{2}, t_4 \mapsto \tfrac{1}{2}\}\\ &= \{t_1 \mapsto \tfrac{1}{4}, t_2 \mapsto \tfrac{1}{4}, t_3 \mapsto \tfrac{1}{4}, t_4 \mapsto \tfrac{1}{4}\}.\end{aligned}$$

Notice that h_{**} is only partially specified above. Now, it is easy to see that h is a hyperstep refinement from the $\mathcal{A}_1$ to $\mathcal{A}_2$ in Figure 2.14. For this, consider the transitions in $\mathcal{A}_2$. We have $s_0 \xrightarrow{a} \{u_1 \mapsto \frac{1}{2}, u_2 \mapsto \frac{1}{2}\}$ in $\mathcal{A}_1$, and indeed,

$$h_{**}(s_0) \xrightarrow{\tau}_{**} h_{**}(\{u_1 \mapsto \tfrac{1}{2}, u_2 \mapsto \tfrac{1}{2}\}),$$

which is just $\{t_0 \mapsto 1\} \xrightarrow{\tau} \{t_1 \mapsto \frac{1}{4}, t_2 \mapsto \frac{1}{4}, t_3 \mapsto \frac{1}{4}, t_4 \mapsto \frac{1}{4}\}$, which is a lifted transition in $\mathcal{A}_2$. Moreover, we have $u_1 \xrightarrow{\tau} \{t_1 \mapsto \frac{1}{2}, t_2 \mapsto \frac{1}{2}\}$ in $\mathcal{A}_1$ and we have $h_{**}(u_1) = h_{**}(\{s_1 \mapsto \frac{1}{2}, s_2 \mapsto \frac{1}{2}\})$ indeed. The other transitions are now easy.

The following theorem states that probabilistic (hyper-)step refinements are a sound proof method for establishing trace distribution precongruence.

Theorem 2.5.10 *Let $\mathcal{A}$ and $\mathcal{B}$ be probabilistic automata with $V_{\mathcal{A}} = V_{\mathcal{B}}$.*

1. *If $\mathcal{A} \sqsubseteq_{\mathrm{PSR}} \mathcal{B}$ then $\mathcal{A} \sqsubseteq_{\mathrm{PHSR}} \mathcal{B}$.*

2. *If $\mathcal{A} \sqsubseteq_{\mathrm{PHSR}} \mathcal{B}$ then $\mathcal{A} \sqsubseteq_{\mathrm{PTD}} \mathcal{B}$.*

PROOF: For (1), suppose that $\mathcal{A} \sqsubseteq_{\mathrm{PSR}} \mathcal{B}$. Then there exists a probabilistic step refinement r from $\mathcal{A}$ to $\mathcal{B}$. Let $h : S_{\mathcal{A}} \to \mathrm{Distr}(S_{\mathcal{B}})$ be given by $h(s) = \{r(s) \mapsto 1\}$. It is routine to check that h is a probabilistic hyperstep refinement from $\mathcal{A}$ to $\mathcal{B}$. Use that

$$\begin{array}{rcl} h_{**}(\mu) & = & r_*(\mu), \\ s \xrightarrow{a}_{\mathcal{B}} \nu & \iff & \{s \mapsto 1\} \xrightarrow{a}_{\mathcal{B}^{**}} \nu. \end{array}$$

Hence $\mathcal{A} \sqsubseteq_{\mathrm{PHSR}} \mathcal{B}$.

For (2), suppose that $\mathcal{A} \sqsubseteq_{\mathrm{PHSR}} \mathcal{B}$. Then there exists a probabilistic hyperstep refinement h from $\mathcal{A}$ to $\mathcal{B}$. We claim that h is a probabilistic forward simulation in the sense of [Seg95b, SL95]. Now $\mathcal{A} \sqsubseteq_{\mathrm{PTD}} \mathcal{B}$ follows from the soundness result for probabilistic forward simulations, see Section 8.7 in [Seg95b].

For a simple, direct proof of (2) we refer to [SV02a]. $\square$

2.6 Other Models for Probabilistic Systems

Several models have been proposed in the literature to model systems with probabilistic choice. These models can be classified according to the types nondeterministic and probabilistic choices they incorporate: They may either not allow nondeterministic choices at all, they may allow external nondeterminism only, or any kind of nondeterminism. With respect to probabilistic choice, we distinguish between models that deal with discrete probabilistic choice, with exponential probability distributions choice or with arbitrary probabilistic choice. [6] Moreover, various languages have been developed to express and analyze the models conveniently. We mention a few of these.

The reader is referred to Table 2.15 for a schematic overview of the models discussed below.

[6]Note that not every probability distribution is either discrete or continuous, for instance the outcome of first rolling a dice and if 6 comes up throwing a dice is 'hybrid.'

2.6.1 Probabilistic Models without Nondeterminism

First, we discuss several purely probabilistic models. These are models that deal with probabilistic choice but not with nondeterministic choice. They have a long mathematical history. Discrete time Markov chains (DTMCs) have been invented by A.A. Markov (1856 – 1922) and further developed A. N. Kolmogorov (1903 – 1987), who was also a pioneer in Continuous time Markov chains (CTMCs). Semi–Markov chains are much newer. All these models have been applied in a wide variety of applications, including genetics, physics, manufacturing systems and queuing networks, and so on. We refer the reader to [Put94] for a comprehensive study of these models.

This section focuses on models with a countable state space. The ideas for systems with uncountable state spaces are the same, but their formulation requires greater care. We present the models as they traditionally are, without action labels. Under the influence of the automaton model, several labeled variants exists nowadays.

Discrete time Markov chains

A DTMC is basically a PA without nondeterministic choices and without action labels. In each state of the system, there is a unique probabilistic transition specifying the probability to reach a certain target state. Rather than a set of start states, a DTMC specifies a initial probability distribution.

Definition 2.6.1 A *discrete time Markov chain* (DTMC) $\mathcal{A}$ consists of three components:

1. A countable set $S_\mathcal{A}$ of *states*,
2. an *initial probability distribution* $\pi \in \mathrm{Distr}(S_\mathcal{A})$,
3. a *transition matrix* $\Delta_\mathcal{A} : S_\mathcal{A} \to \mathrm{Distr}(S_\mathcal{A})$.

The semantics of a DTMC is given by a sequence of random variables $X_1, X_2, X_3, \ldots$ on the state space $S_\mathcal{A}$, where X_i gives the probability that the DTMC ends up in state s after exactly i transitions. We deviate slightly from the usual terminology in the sense that traditionally the approach is converse: one starts with a sequence of random variables having the so-called Markov Property and shows that this sequence can be represented by a transition matrix and an initial distribution. Basically, the Markov Property for DTMCs states that X_{i+1} depends on X_i, but not X_j for $j < i$. Moreover, this dependence is required to be time–invariant. In other words, $\mathbf{P}[X_{i+1} = s \mid X_i = s']$ does not depend on i. Thus, the sequence $X_1, X_2 \ldots$ is *memoryless*, meaning that the future behavior of the system only depends on the present state and not on the past, i.e. the path leading to this state.

Continuous time Markov chains

A *continuous time Markov chain (CTMC)* specifies a unique probability to move from one state to another, as does a DTMC. In addition, a CTMC assigns a *rate* λ_s to each state s.

This rate determines the amount of time that can be spent in the state s. The latter is called the *delay*, *residence time* or *sojourn time* in s and denoted by D_s. In CTMCs, D_s is (negatively) exponentially distributed with parameter λ_s. This means that the probability to stay in s for at most t time units is given by $1-e^{-\lambda_s \cdot t}$. As a formula, $\mathbf{P}[D_s < t] = 1-e^{\lambda_s \cdot t}$.

An exponential distribution with parameter λ has mean value $\frac{1}{\lambda}$. So, the larger the rate of a state, the faster the process leaves this state. Note that the probability to stay in s for *exactly* t time units is 0.

One of the key features of the exponential distributions, which make CTMCs relatively easy to analyze, is the memoryless property. This means that the probability to stay in a state s for at most another t time units does not depend on the amount of time that has already been spent in s, say t'. Mathematically, this is expressed by $\mathbf{P}[D_s < t + t' \mid D_s > t'] = \mathbf{P}[D_s < t] = 1 - e^{-\lambda_s \cdot t}$. Thus, in a CTMC, the entire future behavior, that is the forthcoming states and the future timing behavior, depend only on the present state and not on past states or on the amount of time already spent in this state.

Definition 2.6.2 A *continuous time Markov chain* (CTMC) $\mathcal{A}$ consists of four components:

1. A countable set $S_\mathcal{A}$ of *states*,
2. an *initial probability distribution* $\pi \in \mathrm{Distr}(S_\mathcal{A})$,
3. a *transition matrix* $\Delta_\mathcal{A} : S_\mathcal{A} \to \mathrm{Distr}(S_\mathcal{A})$.
4. a *rate function* $\lambda_\mathcal{A} : S_\mathcal{A} \to \mathbb{R}^{>0}$.

The semantics of a CTMC is given by a family of random variables $\{X_t\}_{t \in \mathbb{R}^{>0}}$, where X_t yields the distributions of being in state s at time t. Also here, we deviate from the tradition to start with the random variables.

Sometimes, the transition matrix and the rate function are integrated into a single function $\overline{\lambda} : S_\mathcal{A} \times S_\mathcal{A} \to \mathbb{R}^{>0}$ by setting $\overline{\lambda}(s, s') = \Delta(s, s') \cdot \lambda(s)$ for $s \neq s'$. Note that the original rates and probabilities can be derived by $\lambda(s) = \sum_{t' \in S_\mathcal{A}} \overline{\lambda}(s, t')$ and $\Delta(s, t) = \frac{\overline{\lambda}(s,t)}{\sum_{t' \in S_\mathcal{A}} \overline{\lambda}(s,t')}$. Then $\overline{\lambda}(s, s')$ expresses that the time to move from s to s' is exponentially distributed with parameter $\lambda(s, s')$.

Stochastic Petri Nets (SPNs) constitute another model based on exponential distributions and can be translated into CTMCs.

Semi–Markov chains

Semi–Markov chains are similar to CTMCs, except that the sojourn time associated to a state can have an arbitrary distribution. As a consequence, the *timing behavior* in a SMC is not memoryless anymore. The remaining time to be spent in a state may depend on the time already spent in there, but not depend on (the time spent in) previous states. In other words, the *state behavior* of a SMC is still memoryless.

Definition 2.6.3 A *Semi–Markov chain* (SMC) $\mathcal{A}$ consists of four components:

1. A set $S_\mathcal{A}$ of *states*,
2. an *initial probability distribution* $\pi \in \mathrm{Distr}(S_\mathcal{A})$,
3. a *transition matrix* $\Delta_\mathcal{A} : S_\mathcal{A} \to \mathrm{Distr}(S_\mathcal{A})$.
4. a function $F_\mathcal{A} : S_\mathcal{A} \to (\mathbb{R} \to [0, 1])$ that assigns an arbitrary *cumulative distribution function* to each state.

As before, the transition matrix and sojourn time distributions can be combined into a single function $\overline{F}_{\mathcal{A}} : S_{\mathcal{A}} \times S_{\mathcal{A}} \to (\mathbb{R} \to [0,1])$ and the semantics of a SMC can be given as a sequence of random variables $\{X_t\}_{t\in\mathbb{R}}$.

A model that can be used to specify SMCs in a more syntactic way are *Generalized Stochastic Petri Nets (GSPNs)* [MCB84]. One can also use *Generalized Semi–Markov Processes* to specify systems with arbitrary distributions. This model is more oriented to system simulation [Gly89, Cas93].

Properties of purely probabilistic models

Unlike for models with probabilistic and nondeterministic choice, where only bounds on the probabilities can be given, allows the purely probabilistic nature of DTMCs, CTMCs and SMCs one to calculate the exact probabilities on many events. A long research tradition in these models has put forward algebraic, analytical and numerical techniques for doing so.

A central issue in these models is *steady state analysis*. Under certain weak assumptions, the probability $\mathbf{P}[X_t = s]$ of being in state s at time t (or after t steps in a DTMC) converges to a limit when t goes to infinity. This means that on the long run, the behavior of the system hardly changes any more.

Another issue is *transient analysis*, which involves for instance the expected number of steps before a certain state is reached.

As remarked before, nondeterminism is essential to define a notion of asynchronous parallel composition and therefore, this operator is not present in these three models without nondeterministic choice. One can however define a synchronous parallel composition operation in which all components operate in lock step [GJS90, GSST95] or a probabilistic merge operator, in which the component that takes the next step is chosen probabilistically [BBS95].

2.6.2 Probabilistic Models with External Nondeterminism

In models that combine probabilistic choice with external nondeterminism, all outgoing edges of a state have different labels. Thus, given the current state and the next action, the next state is determined by a unique probability distribution. Internal nondeterministic choices, expressed by transitions labeled by internal actions or equally labeled transitions leaving from one state, are not allowed in these models. The advantage of such models is that, unlike in purely probabilistic models, a parallel composition operator can be defined. This allows a large system to be specified by several smaller components. On the other hand, when put in an environment that is purely probabilistic (and each transition synchronizes with the environment), the whole system becomes purely probabilistic. Therefore, it can be mapped to one of the models without nondeterminism and the mentioned analysis techniques can be used.

Markov decision processes

A Markov decision process is basically a PA with external nondeterministic choices only. This model is sometimes called the *reactive model*.

Definition 2.6.4 A *Markov Decision Process* (MDP) $\mathcal{A}$ consists of four components:

1. A set $S_{\mathcal{A}}$ of *states*,

2. A set $\mathrm{Act}_{\mathcal{A}}$ of *actions*,
3. an *initial probability distribution* $\pi \in \mathrm{Distr}(S_{\mathcal{A}})$,
4. a *transition matrix* $\Delta_{\mathcal{A}} : S_{\mathcal{A}} \times \mathrm{Act}_{\mathcal{A}} \to \mathrm{Distr}(S_{\mathcal{A}})$.

Note the difference between a transition matrix above, which is a function, and a transition relation in the definition of PAs.

Probabilistic I/O automata

In models with exponential distributions, there is still a problem to define a parallel composition operator. On the one hand, exponential distributions, being memoryless, are very convenient for modeling transitions that are taken independently. This is so because the composite automaton leaves a state (s, t) whenever one of the component processes takes a transition. Therefore the delay in this state is the minimum of two exponential distributions with parameters λ_s and λ_t. This is again an exponential distribution with parameter $\lambda_s + \lambda_t$.

Problems arise, however, when two transitions have to synchronize. It is reasonable to require that synchronization can take place as soon as the sojourn time of both processes involved has expired. As a consequence, the delay of the synchronized transition is given by the maximum of the exponential distributions. Since the maximum of two exponential distributions is not an exponential distribution, the delay cannot be expressed by a rate.

Several solutions for this problem have been proposed. We find the solution adopted by [WSS97] one of the cleanest. It distinguishes between input and output actions. Input actions are required to be always enabled and, importantly, may be taken before the sojourn time in a state has expired. The choice between output and internal actions is purely probabilistic and these actions can only be taken after the sojourn time has passed. Therefore, a closed system (i.e. a system without input actions) can always be mapped to a (labeled) CTMC.

Definition 2.6.5 A *Probabilistic I/O Automaton* (PIOA) $\mathcal{A}$ consists of five components:

1. A set $S_{\mathcal{A}}$ of *states*,
2. An initial state $S^0_{\mathcal{A}} \in S_{\mathcal{A}}$,
3. A set $\mathrm{Act}_{\mathcal{A}}$ of *actions*, which are partitioned into a set of *internal actions* $I_{\mathcal{A}}$, a set of *input actions* $in_{\mathcal{A}}$ and a set of *output actions* $out_{\mathcal{A}}$. The set $out_{\mathcal{A}} \cup I_{\mathcal{A}}$ contains the *locally controlled actions*.
4. a *transition probability function* $\Delta_{\mathcal{A}} : S_{\mathcal{A}} \times \mathrm{Act}_{\mathcal{A}} \times S_{\mathcal{A}} \to [0, 1]$. satisfying
   (a) For all $s \in S_{\mathcal{A}}$ and $a \in in_{\mathcal{A}}$, $\sum_{s' \in S_{\mathcal{A}}} \Delta(s, a, s') = 1$.
   (b) For all $s \in S_{\mathcal{A}}$, $\sum_{a \in I_{\mathcal{A}} \cup out_{\mathcal{A}}} \sum_{s' \in S_{\mathcal{A}}} \Delta(s, a, s') \in \{0, 1\}$.
5. a *rate transition relation* $\lambda_{\mathcal{A}} : S_{\mathcal{A}} \times S_{\mathcal{A}} \to \mathbb{R}^{>0}$. The rate of a state can be 0 if there are no outgoing transitions labeled by locally controlled actions.

The reason why the PIOA model can have internal actions is that the system has to leave the current state if the sojourn time has expired. Therefore, there is no choice between remaining in the same state and taking an internal transition is present and thus, internal transitions do not introduce nondeterministic choice.

The semantics of a PIOA is given as a transformation of so–called observables, where an observable measures a particular quantity of input.

Semi–Markov decision processes

Puterman [Put94] discusses Semi–Markov decision processes. These are basically Semi–Markov chains with external nondeterministic choice. The definition below is a simplified version of Puterman's. A special class of SMDPs are those with exponential sojourn times. In our classification, these models fall in the same class as PIOAs.

Definition 2.6.6 A *Semi–Markov decision process* (SMDP) $\mathcal{A}$ consists of four components:

1. A set $S_{\mathcal{A}}$ of *states*,
2. an *initial probability distribution* $\pi \in \mathrm{Distr}(S_{\mathcal{A}})$,
3. a *transition matrix* $\Delta_{\mathcal{A}} : S_{\mathcal{A}} \times \mathrm{Act}_{\mathcal{A}} \to \mathrm{Distr}(S_{\mathcal{A}})$.
4. a function $F_{\mathcal{A}} : S_{\mathcal{A}} \to (\mathbb{R} \to [0,1])$ that assigns an arbitrary *cumulative distribution function* to each state.

Given an adversary A, the behavior of an SMDP is purely probabilistic. Therefore, the semantics of a SMDP under the adversary A can be given by a sequence of random variables $\{X_t^A\}_{t\in\mathbb{R}}$ that describe the probability distribution at time t. The semantics of an SMDP can then be given by the set of sequences of random variables arising from an arbitrary adversary of the system.

2.6.3 Probabilistic Models with Full Nondeterminism

Probabilistic automata

We have already seen that probabilistic automata and variants thereof such as the alternating model combine nondeterministic and discrete probabilistic choice. There are several process algebras such as ACP [And99b, And99a], the probabilistic π calculus and the probabilistic process algebra defined in [Han94] that allow us to specify such models at a more abstract level.

Interactive Markov chains

Interactive Markov Chains [Her99] distinguish between *interactive transitions*, which are as in nondeterministic automata and allow one to specify external and internal nondeterministic choices, and *Markovian transitions*, which specify the rate with which a transition is taken, as in CTMCs. Moreover, internal transitions are taken as soon as they become enabled. This separation allows for a clean notion of parallel composition which implicitly realizes synchronization as a maximum of exponential distributions.

Definition 2.6.7 A *Interactive Markov chain* (IMC) $\mathcal{A}$ consists of five components:

1. A set $S_{\mathcal{A}}$ of *states*,
2. An initial state $S_{\mathcal{A}}^0 \in S_{\mathcal{A}}$,
3. A set $\mathrm{Act}_{\mathcal{A}}$ of *actions*,
4. a *transition relation* $\Delta_{\mathcal{A}} \subseteq S_{\mathcal{A}} \times \mathrm{Act}_{\mathcal{A}} \times S_{\mathcal{A}}$.

5. a *rate transition relation* $\lambda_{\mathcal{A}} : S_{\mathcal{A}} \times S_{\mathcal{A}} \to \mathbb{R}$.

Moreover, [Her99] introduces a process algebra for IMC as a high level specification language for IMCs. Several notions of bisimulation for IMCs exists, together with their process algebraic axiomatizations, and these can be decided efficiently. A behavior semantics (such as the set of trace distributions for PAs or the sequence of random variables of DTMCs) is not given in [Her99].

Process algebras that combine exponential distributions with nondeterminism are PEPA [Hil96], EMPA [BDG97] and TIPP [GHR93].

SPADES

SPADES, also written ♤, [D'A99], is a process algebra that allows the definitions of *stochastic automata* (SAs). The sojourn time in a state of an SA is specified via clocks, which can have arbitrary probability distributions. That is, each transitions is decorated with a (possibly empty) set of clocks, expressing that this transition can only be taken if these have expired. Immediately afterwards, all clocks in this set are assigned new values according to their probability distributions.

This also allows for a clean notion of parallel composition, because when two transitions are required to synchronize, one just takes the union of their clock reset sets.

Definition 2.6.8 A *Stochastic Automaton* consists of the following components

1. A set $S_{\mathcal{A}}$ of *states*,
2. A set $\mathrm{Act}_{\mathcal{A}}$ of *actions*,
3. A set $\mathcal{C}$ of *random clocks*,
4. a *transition relation* $\Delta_{\mathcal{A}} \subseteq S_{\mathcal{A}} \times \mathrm{Act}_{\mathcal{A}} \times \mathcal{P}(\mathcal{C}) \times S_{\mathcal{A}}$.
5. a *clock resetting function* $\kappa_{\mathcal{A}} : S_{\mathcal{A}} \to \mathcal{P}(\mathcal{C})$,

where $\mathcal{P}(\mathcal{C})$ denotes the powerset of $\mathcal{C}$.

The semantics of stochastic automata is given in terms of stochastic transition systems. These are transition systems in which the target of a transition can be an arbitrary probability space. A behavior semantics for SAs is not given in [D'A99].

2.6.4 Stochastic and Nondeterministic Timing

All the models mentioned above that deal with continuous distributions (i.e. those in the bottom two rows in Figure 2.15) use these to specify probabilistic choices over time (so–called stochastic timing). The probabilistic choice over the state space in these models remains discrete.

Continuous distributions over the states can be useful in various cases, for instance to describe the trajectory of a particle. For stochastic processes (i.e. the sequences of random variables $\{X_t\}_t$), continuous distributions over the state are well studied, but automaton models describing these are less common.

As mentioned before, the SA model generalizes PAs by specifying a general probability space as the target of a transition. In a similar way, the work [DGJP99] defines a MDP with

	none	external	full nondeterminism
discrete	DTMC	MDP	PAs, GPA, AM pr ACP, pr π–calculus
exponential	CTMC SPN	PIOA	IMC algebra for IMC
general	SMP GSNP		SAs $\spadesuit$

Figure 2.15: Classification of probabilistic system models according to the types of nondeterministic and probabilistic choices

continuous state space. By restricting the set of possible target probability spaces, this work enables a natural notion of bisimulation for these processes. Ongoing work in that direction [DP01] concerns the development of a CTMC model with a continuous space.

Another useful direction to extend the models discussed above is nondeterministic timing. This type of timing allows a(n external) nondeterministic choice between several (possibly uncountably many) potential delays, which can be used to specify bounds such as “this delay lies between 1 and 2 seconds.” The PA model has been extended in this way, leading to (two different types of) PTAs, one in [Seg95b] and another in [KNSS01].

Similar extensions could be made to the MDP model. For the PIOA, IMC and $\spadesuit$ model, the situation is more complicated because these models already deal with stochastic sojourn times and equipping them with nondeterministic time would require new analysis methods to compute for instance expected times. Recently, the modeling language MoDest has been proposed [DHKK01], which combines discrete probabilistic and nondeterministic choices over states with both stochastic and nondeterministic timing.

2.6.5 This Thesis

In this thesis, we are interested in the verification and analysis of randomized, distributed algorithms and probabilistic communication protocols. Then the PA model is a natural choice. The applications often contain discrete probabilistic choices and — as it is the case for non-probabilistic distributed algorithms and communication protocols — external and internal nondeterminism are needed to model implementation freedom, scheduling freedom, incomplete information and the environment.

We use the PA model with minor variations and also the notion of behavior differs in minor details throughout the chapters of this thesis. As a consequence, the basic definitions treated here are briefly recalled at several places in this thesis.

The tables below present an overview of variant models and definitions of behavior we used.

	start states	I/O distinction	internal actions	timing
This chapter	set	no	set	no/yes
Chapter 4	single	no	single	no
Chapter 5	none/single	no	single	no
Chapter 7	set single	yes yes	set single	no/yes yes

Figure 2.16: Overview of the variants of the PA model used in this thesis

chapter	notion of behavior based on
This chapter	adversary over all paths starting in start state
Chapter 4	adversary, starting in the start state
Chapter 5	adversary, defined over all paths in a PA

Figure 2.17: Overview of the variants of the notions of behavior used in this thesis

2.7 Summary

The probabilistic automaton model introduced in this chapter combines discrete probabilistic choice and nondeterministic choice in an orthogonal way. This allows us to define an asynchronous parallel composition operator and makes the model suitable for reasoning about randomized distributed algorithms, probabilistic communication protocols and systems with failing components. PAs subsume nonprobabilistic transition systems, Markov decision processes and Markov chains. Nondeterministic timing can be naturally incorporated in this model, but stochastic time, allowing for continuous probabilistic choice over time, cannot.

The behavior of a PA relies on randomized, partial adversaries. These resolve the nondeterministic choices in the model and replace them by probabilistic ones. By ranging over all possible adversaries, one obtains the set of associated probability spaces of a PA. These, in their turn, yield the set of trace distributions, describing the external behavior of a PA.

The implementation relation proposed for PAs is the trace distribution precongruence. This is the largest precongruence relation contained in the trace distribution inclusion relation. The latter is not a precongruence. The trace distribution precongruence can be characterized alternatively by a principal context. Surprisingly, this context can also distinguish between nonprobabilistic automata that are trace equivalent.

We ended this section with an overview of probabilistic models and classified them according to their treatment of probabilistic choice, nondeterministic choice and the nature of time (nondeterministic or probabilistic).

CHAPTER 3
Preliminaries from Probability Theory

La théorie des probabilités n'est autre que le sens commun fait calcul
Pierre Simon Laplace, 1819

This chapter recalls a few basic notions from probability theory and introduces some notation that is used throughout this thesis.

The following definition of sum allows one to work with sums over arbitrary sets. For countable sums, the notion coincides with the usual one.

3.1 Probability Distributions

Definition 3.1.1 Let $\mathcal{I}$ be an index set and let $x_i \in [0, \infty]$ for all $i \in \mathcal{I}$. Define $\sum_{i \in \mathcal{I}} x_i$ by

1. $\sum_{i \in \emptyset} x_i = 0$,
2. $\sum_{i \in \mathcal{I}} x_i = x_{i_1} + x_{i_2} + x_{i_3} + \cdots + x_{i_n}$, if $\mathcal{I} = \{i_1, i_2, i_3, \ldots, i_n\}$ is a finite set with $n > 0$ elements,
3. $\sum_{i \in \mathcal{I}} x_i = \mathrm{supr}\{\sum_{i \in \mathcal{J}} x_i \mid \mathcal{J} \subseteq \mathcal{I} \text{ is finite}\}$, if $\mathcal{I}$ is infinite.

Here $\mathrm{supr}\ X$ denotes the supremum of a set X. Notice that $\sum_{i \in \mathbb{N}} x_i = \sum_{i=0}^{\infty} x_i$ because, due to the fact that $x_i \geq 0$, the summation order is irrelevant. It is also easy to see that, if $\sum_{i=0}^{\infty} x_i$ is finite, then there are countably[1] many nonzero elements x_i.

Definition 3.1.2 A *discrete probability distribution*,[2] or simply a *probability distribution*, over a set X is a function $\mu : X \to [0, 1]$ such that $\sum_{x \in X} \mu(x) = 1$. We write $\mathrm{supp}(\mu) = \{x \in X \mid \mu(x) > 0\}$. It follows immediately from the definitions that this is a countable set. We denote the set of all probability distributions over X by $\mathrm{Distr}(X)$.

We denote a probability distribution μ on a countable domain by enumerating it as a set of pairs. So, if $\mathrm{Dom}(\mu) = \{x_1, x_2 \ldots\}$ then we denote μ by $\{x_1 \mapsto \mu(x_1), x_2 \mapsto \mu(x_2) \ldots\}$. If the domain of μ is known from the context, then we often leave out elements of probability

[1] In our terminology, the countable sets subsume the finite sets.

[2] Here, we deviate from the terminology used in classical probability theory. In that setting, the outcomes of probabilistic experiments are described by random variables. The probability distribution of a random variable Z is the function $z \mapsto \mathbf{P}[Z \leq z]$. For discrete random variables, the function $z \mapsto \mathbf{P}[Z = z]$ is a probability distribution in our sense and is called the *probability density function* or *probability mass function* of Z.

zero. For instance, the probability distribution assigning probability one to an element $x \in X$ is denoted by $\{x \mapsto 1\}$, irrespective of X. This distribution is called the *Dirac distribution* over x. The *uniform distribution* over a finite set with $n > 0$ elements, say $\{x_1, \ldots, x_n\}$, is given by $\{x_1 \mapsto \frac{1}{n}, \ldots, x_n \mapsto \frac{1}{n}\}$.

Definition 3.1.3 Let X and Y be sets, $\mu \in \mathrm{Distr}(X)$ and $\nu \in \mathrm{Distr}(Y)$. The *product* of μ and ν, notation $\mu \times \nu$, is the probability distribution $\kappa : X \times Y \to [0, 1]$ such that $\kappa(x, y) = \mu(x) \cdot \nu(y)$.

CHAPTER 4
A Testing Scenario for Probabilistic Automata

Een kind op een strand pakt een handvol zand en gooit het weg. Hoe groot was de kans dat die zandkorrels ooit nog eens bij elkaar zouden komen? Nul. Maar hoe groot was dan duizend jaar geleden die kans geweest? Ook nul. Toch waren ze bij elkaar gekomen. Zo was het met alles. Een voetstap die werd gezet, een druppel die uit de kraan viel, een vogel die op een tak ging zitten – nooit meer zou het precies zo gebeuren, nooit had het kunnen gebeuren.

Tim Krabbé, *De grot*

Abstract Recently, a large number of equivalences for probabilistic automata has been proposed in the literature. However, to the best of our knowledge, none of these equivalences has been characterized in terms of intuitive testing scenarios, except for the probabilistic bisimulation of Larsen & Skou. In our view, this is an undesirable situation: any behavioral equivalence should either be characterized via some plausible testing scenario, or be strictly finer than such an equivalence and be justified via computational arguments. In this chapter, we propose and study a simple and intuitive testing scenario for probabilistic automata. We prove that the equivalence induced by this scenario coincides with the trace distribution equivalence proposed by Segala. A technical result that we need to establish on the way is an *Approximation Induction Principle (AIP)* for probabilistic processes.

4.1 Introduction

A fundamental idea in concurrency theory is that two systems are deemed equivalent if they cannot be distinguished by observation. Depending on the power of an observer, different notions of behavioral equivalence arise. For labeled transition systems, this idea has been thoroughly explored and a large number of behavioral equivalences has been characterized operationally, algebraically, denotationally, logically, and via intuitive "testing scenarios" (also called "button pushing experiments"). We refer to Van Glabbeek [Gla01] for an excellent overview of results in this area of *comparative concurrency semantics*.

Testing scenarios provide an intuitive understanding of a behavioral equivalence via a machine model. A process is modeled as a black box that contains as its interface to the outside world (1) a display on which the name of the action is shown that is currently carried out by the process, and (2) some buttons via which an observer may attempt to influence the execution of the process. A process autonomously chooses an execution path that is consistent with its position in the labeled transition system that is contained in the black box. Trace semantics, for instance, is explained in [Gla01] with the *trace machine*, depicted

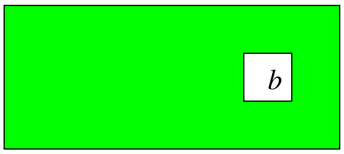


Figure 4.1: The trace machine.

in Figure 4.1. As one can see, this machine has no buttons at all. A slightly less trivial example is the *failure trace machine*, depicted in Figure 4.2, which, apart from the display,

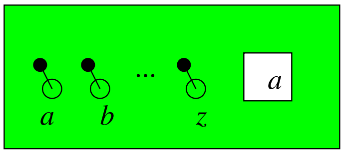


Figure 4.2: The failure trace machine.

contains as its interface to the outside world a switch for each observable action. By means of these switches, an observer may determine which actions are *free* and which are *blocked*. This situation may be changed at any time during a run of a process. The display becomes empty if (and only if) a process cannot proceed due to the circumstance that all actions it is prepared to continue with are blocked. If, in such a situation, the observer changes her mind and allows one of the actions the process is ready to perform, an action will become visible again in the display. Figure 4.3 gives an example of two labeled transition systems

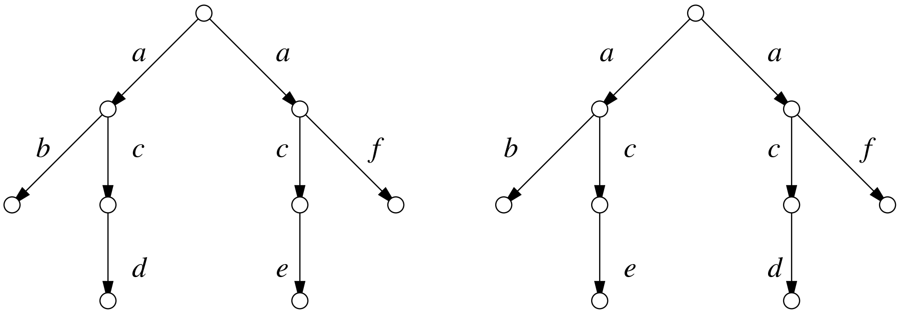


Figure 4.3: Trace equivalent but not failure trace equivalent.

that can be distinguished by the failure trace machine but not by the trace machine. Since both transition systems have the same traces (ε, a, ab, af, ac, acd and ace), no difference can be observed with the trace machine. However, via the failure trace machine an observer can see a difference by first blocking actions c and f, and only unblocking action c when the display becomes empty. In this scenario an observer of the left system may see an e, which is impossible for an observer of the right system. We refer to [Gla01] for an overview of testing scenarios for labeled transition systems.

In this chapter, we propose and study a very simple and intuitive testing scenario for probabilistic automata: we just add an *on/off* button to the trace machine. The resulting *trace distribution machine* is depicted in Figure 4.4. By turning the machine off and then on again

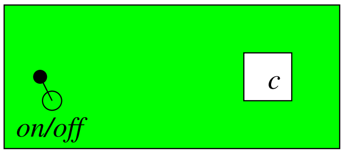


Figure 4.4: The trace distribution machine.

it returns to its initial state and starts again from scratch. In the non-probabilistic case the presence of an *on/off* button does not make a difference[1], but in the probabilistic case it does: we can observe probabilistic behavior by repeating experiments and applying methods from statistics. Consider the two probabilistic automata in Figure 4.5. Here the arcs indicate

Figure 4.5: Probabilistic automata representing a fair and an unfair coin.

probabilistic choice (as opposed to the nondeterministic choice in Figure 4.3), and probabilities are indicated next to the edges. These automata represent a fair and an unfair coin, respectively. We assume that the trace distribution machine has an "oracle" at its disposal which resolves the probabilistic choices according to the probability distributions specified in the automaton. As a result, an observer can distinguish the two systems of Figure 4.5 by repeatedly running the machine until the display becomes empty and then restart it using the *on/off* button. For the left process the number of occurrences of trace ab will approximately equal the number of occurrences of trace ac, whereas for the right process the ratio of the occurrence of the two traces will converge to $1 : 2$. Elementary methods from statistics allow one to come up with precise definitions of distinguishing tests.

The situation becomes more interesting when both probabilistic and nondeterministic choices are present. Consider the probabilistic automaton in Figure 4.6. If we repeatedly run the trace distribution machine with this automaton inside, the ratio between the various traces does not need to converge to a fixed value. However, when we run the machine sufficiently often we will observe that a weighted sum of the number of occurrences of traces ac and ad will approximately equal the number of occurrences of traces ab. Restricting attention to the cases where the left transition has been chosen, we observe $\frac{1}{2}\#[ac] \approx \#[ab]$. Restricting attention to the cases where the right transition has been chosen, we observe $\frac{1}{3}\#[ad] \approx \#[ab]$. Since in each execution either the left or the right transition will be selected, we have:

$$\frac{1}{2}\#[ac] + \frac{1}{3}\#[ad] \quad \approx \quad \#[ab].$$

[1] For this reason an *on/off* button does not occur in the testing scenarios of [Gla01]. An obvious alternative to the *on/off* button would be a *reset* button.

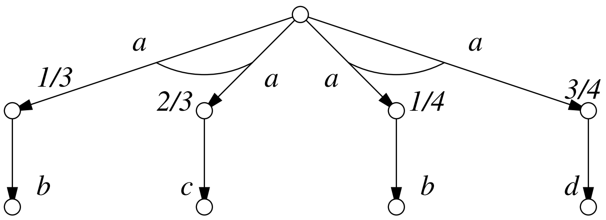


Figure 4.6: The combination of probabilistic and nondeterministic choice.

Even though our testing scenario is trivial, the combination of nondeterministic and probabilistic choice makes it far from easy to characterize the behavioral equivalence on probabilistic automata which it induces.

The main technical contribution of this chapter is a proof that the equivalence (preorder) on probabilistic automata induced by our testing scenario coincides with the trace distribution equivalence (preorder) proposed by Segala [Seg95a]. A technical result that we need to establish on the way is an *Approximation Induction Principle (AIP)* (cf. [BK86, BBK87]) for probabilistic processes. This principle says that if two finitely branching processes are equivalent up to any finite depth, then they are equivalent.

Being a first step, this chapter limits itself to a very simple class of probabilistic processes and to observers with limited capabilities. First of all, only *sequential* processes are investigated: processes capable of performing at most one action at a time. Furthermore, we only study *concrete* processes in which no internal actions occur. Finally, observers can only interact with machines in an extremely limited way: apart from observing termination and the occurrence of actions, the only way in which they can influence the course of events is via the *on/off* button[2]. It will be interesting to extend our result to richer classes of processes and more powerful observers, and to consider for instance a probabilistic version of the failure trace machine described earlier in this introduction.

Related work Several testing preorders and equivalences for probabilistic processes have been proposed in the literature [Chr90a, Seg96, CDSY99, JY01]. These papers all consider testing equivalences and preorders in the style of De Nicola and Hennesy [DNH84]. That is, they define a test as a (probabilistic) process that interacts with a system via shared actions and that can report success in some way, for instance via success state or success actions. When a test is run on a system, one can compute the probability on success or, if nondeterminism is present in either the test or the system, a set of these. By comparing the probabilities on success, one can say whether or not two systems are testing equivalent. For instance, two systems $\mathcal{A}$ and $\mathcal{B}$ are testing equivalent in the sense of [JY01] if and only if for all tests T the maximal probability on success in $\mathcal{A} \parallel T$ is less or equal to the maximal probability on success in $\mathcal{B} \parallel T$. The different equivalences and preorders in the mentioned papers arise by considering different kinds of probabilistic systems, by studying tests with different power (purely nondeterministic tests, finite trees or no restrictions) and by using different ways to compare two systems under test (e.g. may testing versus must testing). All of the mentioned papers provide alternative characterizations of the testing equivalences (preorders, respectively) they introduce in terms of trace distribution equivalence–like relations (trace

[2]This ensures that our testing scenario truly is a "button pushing experiment" in the sense of Milner [Mil80]!

distribution inclusion–like relations), that is in terms of an extension of the trace equivalence relation (trace inclusion relation) to probabilistic processes.

Thus, the testing relations defined in the mentioned works are button pushing experiments in the sense that a test interacts with a system via synchronization on shared actions. We argue that these relations are not entirely observational because it is not described how the probability on success of a system can be observed.

Our work is most related to the paper by Larsen & Skou [LS91]. This chapter defines a notion of tests for reactive probabilistic processes (that is, processes in which all outgoing transitions of a state have different labels) which allow one to make arbitrary copies of any state. Using those tests, a fully observable characterization of probabilistic bisimulation based on hypothesis testing is given. (We note that copies of tests can both serve to discover the branching structure of a system – as in the nondeterministic case – and to repeat a certain experiment a number of times.)

Each test T in [LS91] gives rise to a set of observations O_T. Tests allow certain properties to be tested with arbitrary confidence $\alpha \in [0,1]$, the so–called level of significance. More precisely, a property Φ is said to be testable if for every level of significance α, there is a test T and a partition of the level of observations O_T into $(E_\Phi, O_T \setminus E_\Phi)$ satisfying the following. If Φ holds in a state s and T is run in s, then it is likely that we observe an element from E_Φ: $\mathbf{P}[E_\Phi] \geq 1 - \alpha$. On the other hand, if Φ does not hold, then the probability to observe an element in E_Φ is small: $\mathbf{P}[E_\Phi] \leq \alpha$. Thus, by checking whether the outcome of the test is in E_Φ or not, we can find out whether s satisfies Φ, where probability that the judgment is wrong is less than α. Using the terminology from hypothesis testing, Φ is the null hypothesis and E_Φ the critical section.

Then is it shown that two states in a system that satisfies the minimal derivation assumption are probabilistically bisimilar if and only if they satisfy exactly the same testable properties. Here the minimal derivation assumption requires that any probability occurring in the system is an integer multiple of some value ε. (Technically, this result is achieved via the probabilistic modal logic PML.) Thus, although not explicitly phrased in these terms, one can say Larsen & Skou present a button pushing scenario for probabilistic processes.

Our work differs from the approach in [LS91] in the following aspects.

- We present our results in the more general PA model, whereas [LS91] considers the reactive model. As a consequence, the probability on a set of observations of a system under a test in [LS91] is uniquely determined. This is not the case for the observations we consider, which therefore give rise to a more generic notion of observation.

- The main result of this chapter, which is the characterization of trace distribution inclusion as a testing scenario, is established for all finitely branching systems, which is much less restrictive than the minimal derivation assumption needed for the results in [LS91].

- The possibility in the testing scenario of Larsen & Skou to make copies of processes in any state (at any moment), is justified for instance in the case of a sequential system where one can make core dumps at any time. But for many distributed systems, it is not possible to make copies in any but the initial state. Therefore, it makes sense to study scenarios in which copying is not possible, as done in this chapter.

Overview Even though readers may not expect this after our informal introduction, the rest of this chapter is actually quite technical. We start in Section 4.2 with some mathematical preliminaries concerning functions, sequences and probability theory. In Section 4.2.3 we recall the definitions of probabilistic automata and their behavior. Section 4.3 is entirely devoted to the proof of the AIP for probabilistic processes. Section 4.4, finally, presents the characterization of the testing preorder induced by the trace distribution machine as trace distribution inclusion.

4.2 Preliminaries

4.2.1 Functions

If f is a function, then we denote the domain of f by $Dom(f)$. The *range* of f, notation $Ran(f)$, is the set $\{f(x) \mid x \in X\}$. If X is a set, then the *restriction* of f to X, notation $f \restriction X$, is the function g with $Dom(g) = Dom(f) \cap X$ satisfying $g(x) = f(x)$ for each $x \in Dom(g)$. We say that a function f is a *subfunction* of a function g, and write $f \subseteq g$, if $Dom(f) \subseteq Dom(g)$ and $f = g \restriction Dom(f)$. A function f is called *finite* if $Dom(f)$ is finite.

4.2.2 Sequences

Let X be any set. A *sequence* over X is a function σ from a downward closed subset of the natural numbers to X. So the domain of a sequence is either the set $\mathbb{N}$ of natural numbers, or of the form $\{0, \ldots, k\}$, for some $k \in \mathbb{N}$, or the empty set. In the first case we say that the sequence is infinite, otherwise we say it is finite. The sets of finite and infinite sequences over X are denoted by X^* and X^∞, respectively. We will sometimes write σ_n rather than $\sigma(n)$. The symbol ε denotes the empty sequence, and the sequence containing one element $x \in X$ is denoted by x. Concatenation of a finite sequence with a finite or infinite sequence is denoted by juxtaposition. We say that a sequence σ is a *prefix* of a sequence ρ, denoted by $\sigma \sqsubseteq \rho$, if $\sigma = \rho \restriction Dom(\sigma)$. Thus $\sigma \sqsubseteq \rho$ if either $\sigma = \rho$, or σ is finite and $\rho = \sigma\sigma'$ for some sequence σ'. If σ is a nonempty sequence then $\mathit{first}(\sigma)$ denotes the first element of σ and, if σ is also finite, then $\mathit{last}(\sigma)$ denotes the last element of σ. Finally, $\mathit{length}(\sigma)$ denotes the length of a finite sequence σ. A *subsequence* of an infinite sequence σ is an infinite sequence ρ that is obtained by removing finitely or infinitely many elements from σ. Formally, ρ is a subsequence of σ if there is an *index function*, that is a function $\iota : \mathbb{N} \to \mathbb{N}$ such that (a) ι is strictly monotone (i.e., $n < m$ implies $\iota(n) < \iota(m)$), and (b) $\rho = \sigma \circ \iota$.

An elementary (but fundamental) result from Analysis is the following theorem from Bolzano–Weierstraß.

Theorem 4.2.1 (Bolzano–Weierstraß) *Every bounded infinite sequence in $\mathbb{R}^n$ has a convergent subsequence.*

Let $f_0, f_1, f_2, \ldots$ be an infinite sequence of functions in $X \to [0, 1]$, where X is a finite set. Then this sequence can be seen as a sequence over $[0, 1]^n$, where n is the cardinality of X. Applying the Bolzano–Weierstraß Theorem to $f_0, f_1, f_2 \ldots$ yields that this sequence has a convergent subsequence, i.e. there exists an index function ι such that $f_{\iota(0)}, f_{\iota(1)}, f_{\iota(2)}, \ldots$ has a limit (in $[0, 1]^n$).

4.2.3 Probabilistic Automata

In this chapter, we use the PA model introduced in Chapter 2. However, we want to consider only systems with a unique initial state and with external actions only, taken from a fixed, finite set Act. Moreover, we find it for technical reasons convenient to assume a special element $\delta \in Act$, referred to as the *halting action*. Hence, the automata in this chapter have the following structure. They are triples $\mathcal{A} = (S, s^0, \Delta)$ such that

- S is a set of states,
- $s^0 \in S$ is the initial state, and
- $\Delta \subseteq S \times Act \times \mathrm{Distr}(S)$ is a transition relation.

It is clear that the PAs in this section are special cases of the ones introduced previously and, hence, all terminology from Chapter 2 also applies to the PAs in this chapter. The remainder of this section introduces some additional notation.

Definition 4.2.2 A PA $\mathcal{A}$ is *finitely branching* if for each state s, the set $\{(a, \mu, t) \mid s \overset{a,\mu}{\rightsquigarrow} t\}$ is finite.

Thus, each state in a finitely branching PA has finitely many outgoing transitions and the target distribution of each transition has a finite support.

Definition 4.2.3 The *halting extension* of $\mathcal{A}$ is the PA $\delta\mathcal{A} = (S \cup \{\perp\}, s^0, \Delta')$, where Δ' is the least relation such that

1. $s \xrightarrow{\delta}_{\delta\mathcal{A}} \{\perp\mapsto 1\}$,
2. $s \xrightarrow{a}_{\mathcal{A}} \mu \implies s \xrightarrow{a}_{\delta\mathcal{A}} (\mu \cup \{(\perp\mapsto 0)\})$.

Here we assume that $\perp$ is fresh. The transitions with label δ are referred to as *halting* transitions.

The adversaries we consider in this chapter differ slightly from those in Chapter 2. Rather than yielding the value $\perp$, the adversaries here schedule a δ–transition to indicate that the execution is interrupted. Technically, this has the advantage that the maximal paths of these adversaries are exactly their infinite paths. Moreover, these adversaries allow an easy definition of what it means to halt after k steps.

Definition 4.2.4 A *(partial, randomized) adversary* E of $\mathcal{A}$ is a function

$$E : \mathit{Path}^*(\delta\mathcal{A}) \to \mathrm{Distr}(Act \times \mathrm{Distr}(S_{\delta\mathcal{A}}))$$

such that, for each finite path π, if $E(\pi)(a, \mu) > 0$ then $\mathit{last}(\pi) \xrightarrow{a}_{\delta\mathcal{A}} \mu$.

We say that E is *deterministic* if, for each π, $E(\pi)$ is a Dirac distribution. Adversary E *halts* on a path π if it extends π with the halting transition, i.e.,

$$E(\pi)(\delta, \{\perp\mapsto 1\}) \quad = \quad 1.$$

For $k \in \mathbb{N}$, we say that the adversary E *halts after k steps* if it halts on all paths with length greater than or equal to k. We call E *finite* if it halts after k steps, for some $k \in \mathbb{N}$.

The adversaries in this chapter are special cases of those in Chapter 2. Hence, each of the adversaries in this chapter can be assigned a probability space $(\Omega_E, \mathcal{F}_E, \mathbf{P}_E)$ and a trace distribution $(\Omega_H, \mathcal{F}_H, \mathbf{P}_H)$ as before, see Definition 2.3.13 on page 46 and Definition 2.3.16 on page 48 respectively.

This gives rise to a second notion of trace distribution for PAs: to define the set of trace distributions in a PA, one can either use the adversaries or the adversaries defined in this chapter. However, it is easy to see that they induce exactly the same notion of trace distribution inclusion. In the sequel, also the function $\mathbf{Q}^E(\pi)$, yielding the probability on the path π in the adversary E, plays an important role. Hence, the reader may have another look at Definition 2.3.10 on page 45.

In the next section, we focus on trace distributions generated from adversaries (those from Definition 4.2.4) that halt after k steps, for a given k. Hence, we introduce the following notation.

Notation 4.2.5 For $k \in \mathbb{N} \cup \{\infty\}$, denote the set of all paths of $\mathcal{A}$ of length k by $Path^k(\mathcal{A})$. The set of trace distributions of adversaries of $\mathcal{A}$ which halt after k steps is denoted $trd(\mathcal{A}, k)$. If $trd(\mathcal{A}, k) \subseteq trd(\mathcal{B}, k)$, then we write $\mathcal{A} \sqsubseteq^k_{\mathrm{TD}} \mathcal{B}$.

It is important to realize that for adversary E that halts after k steps, $\mathbf{P}_E$ is fully determined by $\mathbf{Q}^E \upharpoonright Path^k(\mathcal{A})$, i.e., the probability on the paths of length k.

Lemma 4.2.6 *Let $k \in \mathbb{N}$, let $\mathcal{A}$ be a finitely branching PA and let E be an adversary of $\mathcal{A}$ that halts after k steps. Then E can be written as a convex combination of deterministic adversaries that halt after k steps, i.e., there exists a probability distribution ν over the set $\mathcal{D}$ of deterministic adversaries of $\mathcal{A}$ that halt after k steps such that, for all π, a and μ,*

$$E(\pi)(a, \mu) \quad = \quad \sum_{D \in \mathcal{D}} \nu(D) \cdot D(\pi)(a, \mu).$$

PROOF: Since $\mathcal{A}$ is finitely branching, each adversary that halt after k steps is in fact a finite adversary. Hence, this result is a reformulation of Proposition 5.3.24(1). $\square$

4.3 The Approximation Induction Principle

This section is entirely devoted to a proof of an *Approximation Induction Principle (AIP)* (cf. [BK86, BBK87]) for probabilistic processes. We need this result to characterize the equivalence on probabilistic automata induced by the trace distribution machine in Section 4.4. The result and an informal proof sketch have also in appeared in [Seg96]. The proof below provides an explicit construction of the adversary needed and shows that this adversary meets the desired properties, thus filling in several nontrivial details which were not proven in [Seg96].

Theorem 4.3.1 (Approximation Induction Principle) *Let $\mathcal{A}$ and $\mathcal{B}$ be PAs and let $\mathcal{B}$ be finitely branching. Then*

$$\forall k[\mathcal{A} \sqsubseteq^k_{\mathrm{TD}} \mathcal{B}] \implies \mathcal{A} \sqsubseteq_{\mathrm{TD}} \mathcal{B}.$$

PROOF: Assume that $\mathcal{A} \sqsubseteq^k_{\mathrm{TD}} \mathcal{B}$, for all k. In order to prove $\mathcal{A} \sqsubseteq_{\mathrm{TD}} \mathcal{B}$, let H be a trace distribution of $\mathcal{A}$ and let E be an adversary of $\mathcal{A}$ with $H = trdistr(E)$. Via a number of sublemma's, we prove that $H \in trdistr(\mathcal{B})$.

For each $k \in \mathbb{N}$, define E_k by

$$E_k(\pi) = \begin{cases} E(\pi) & \text{if } |\pi| < k, \\ \{(\delta, \{\perp \mapsto 1\}) \mapsto 1\} & \text{otherwise.} \end{cases}$$

Clearly, E_k is an adversary of $\mathcal{A}$ that equals E on paths with length less than k and halts after k steps. Therefore $E_k \in \mathit{trdistr}(\mathcal{A}, k)$. By assumption, there is an adversary F_k of $\mathcal{B}$ such that $\mathit{trdistr}(E_k) = \mathit{trdistr}(F_k)$. We view F_k as a function in

$$\mathit{Path}^*(\delta\mathcal{B}) \times \mathit{Act} \times \mathrm{Distr}(S_{\delta\mathcal{B}}) \to [0, 1].$$

We will construct an adversary G of $\mathcal{B}$ with $\mathit{trdistr}(G) = H$ from the sequence of functions $F = F_0, F_1, F_2 \ldots$. The idea is that, since only the paths of length k matter, F_k is essentially a finite function and we can use the Bolzano–Weierstraß Theorem to obtain G as the limit of a convergent subsequence of F. However, this theorem cannot be applied immediately, because the finite domains of these functions are growing. Therefore, we will operate in several stages. The basic idea is to construct at stage $n + 1$ a convergent subsequence with index function ι_{n+1} of F_0^n, F_1^n, $F_2^n \ldots$, where F_k^n is the restriction of F_k to paths of length n. This sequence consists of finite functions with the same, finite domain and a bounded range (viz. $[0, 1]$) and has therefore a convergent subsequence. We define G_n as the limit of ι_n. Thus, we will obtain a sequence of increasing subfunctions $G_1 \subseteq G_2 \subseteq G_3 \ldots$ and we take G to be its limit. We will need several technical lemma's to ensure that everything is as expected and to prove finally that $\mathit{trdistr}(G) = \mathit{trdistr}(E)$.

Throughout this proof, we use the following notations.

$$\begin{aligned} P_n &= \mathit{Path}^n(\delta\mathcal{B}) \\ D_n &= \{\mu \in \mathrm{Distr}(S_{\delta\mathcal{B}}) \mid \mu \text{ occurs in some } \pi \in P_{n+1}\} \end{aligned}$$

$$\begin{aligned} P &= \mathit{Path}^*(\delta\mathcal{B}) \\ D &= \mathrm{Distr}(S_{\delta\mathcal{B}}) \end{aligned}$$

Note that $P_n \subseteq P_{n+1}$, $D_n \subseteq D_{n+1}$, $P = \cup_n P_n$ and $D \supseteq \cup_n D_n$. In fact, D may contain distributions that are not contained in any D_n. Observe also that $\pi \in P_n$ and $\pi \overset{a,\mu}{\rightsquigarrow} s$ implies $\mu \in D_n$. Since $\mathcal{B}$ is finitely branching, there are only finitely many paths of length at most n and hence P_n and D_n are both finite. Recall that Act is finite by definition. Therefore, the following function F_k^n is finite:

$$\begin{aligned} F_k^n &: P_n \times \mathit{Act} \times D_n \to [0, 1] \\ F_k^n &= F_k \restriction P_n \times \mathit{Act} \times D_n. \end{aligned}$$

Claim 1 $F_k^n \subseteq F_k^{n+1}$ for all k, n.

PROOF: Easy verification. ⊠

For each n, let ρ_n be the sequence

$$\rho_n \quad = \quad F_0^n \ F_1^n \ F_2^n \ F_3^n \ldots$$

and let ι_n be the index function defined inductively as follows:

- ι_0 is the identity function.
- Let ι be the index function of a convergent subsequence of $\rho_n \circ \iota_n$ (such a subsequence exists by the Bolzano–Weierstraß Theorem). Then $\iota_{n+1} = \iota_n \circ \iota$.

We define function $G_n : P_n \times \mathit{Act} \times D_n \to [0,1]$ by

$$G_n = \lim(\rho_n \circ \iota_{n+1}),$$

i.e. , G_n is the limit of the convergent subsequence just defined. Note that G_n is in fact the point wise limit of a sequence of functions. (It needs not to be an adversary).

Claim 2 $G_n \subseteq G_{n+1}$.

PROOF: Clearly, $\mathrm{Dom}(G_n) \subseteq \mathrm{Dom}(G_{n+1})$. Let $(\pi, a, \mu) \in \mathrm{Dom}(G_n)$. Then

$$\begin{aligned} G_n(\pi)(a,\mu) &= \lim_{k\to\infty} F^n_{\iota_{n+1}(k)}(\pi)(a,\mu) && \{\mathrm{Ran}(\iota_{n+2}) \subseteq \mathrm{Ran}(\iota_{n+1}\} \\ &= \lim_{k\to\infty} F^n_{\iota_{n+2}(k)}(\pi)(a,\mu) && \{\text{Claim 1}\} \\ &= \lim_{k\to\infty} F^{n+1}_{\iota_{n+2}(k)}(\pi)(a,\mu) \\ &= G_{n+1}(\pi)(a,\mu). \end{aligned}$$

⊠

Let $G' = \cup_n G_n$, i.e. for $\pi \in P_n$ and $\mu \in D_n$, $G'(\pi)(A,\mu) = G_n(\pi)(a,\mu)$. Then G' is a function in $\cup_n P_n \times \mathit{Act} \times D_n \to [0,1]$. We extend G' to a function G in $P \times \mathit{Act} \times D \to [0,1]$ as follows

$$G(\pi)(a,\mu) = \begin{cases} G'(\pi)(a,\mu) & \text{if } \exists n : \pi \in P_n \wedge \mu \in D_n, \\ 0 & \text{otherwise.} \end{cases}$$

The rest of this proof is concerned with showing that G is an adversary with $\mathit{trdistr}(G) = H$, which is exactly what we are after.

Claims 3 and 4 show that G is an adversary of $\mathcal{B}$. Claim 3 proves that G respects the transition relation of $\delta\mathcal{B}$, and Claim 4 establishes that G has the required type, i.e.

$$G : \mathit{Path}^*(\delta\mathcal{B}) \to \mathrm{Distr}(\mathit{Act} \times \mathrm{Distr}(S_{\delta\mathcal{B}})).$$

Claim 3 Suppose $\pi \in P$, $a \in \mathit{Act}$, $\mu \in D$ and $G(\pi)(a,\mu) > 0$. Then $\mathit{last}(\pi) \xrightarrow{a} \mu$ is a transition of $\delta\mathcal{B}$.

PROOF: Since $G(\pi)(a,\mu) > 0$, it follows from the definition of G that $G(\pi)(a,\mu) = G'(\pi)(a,\mu)$. Hence, by definition of G', there is an n such that $G'(\pi)(a,\mu) = G_n(\pi)(a,\mu)$. Then

$$\begin{aligned} 0 &< G_n(\pi)(a,\mu) && \{\text{def. } G_n\} \\ &= \lim_{k\to\infty} F^n_{\iota_{n+1}(k)}(\pi)(a,\mu) && \{\text{def. } F^n_i\} \\ &= \lim_{k\to\infty} F_{\iota_{n+1}(k)}(\pi)(a,\mu) \end{aligned}$$

This implies that $F_{\iota_{n+1}(k)}(\pi)(a,\mu) > 0$ for large k. Since $F_{\iota_{n+1}(k)}$ is an adversary of $\mathcal{B}$, $\mathit{last}(\pi) \xrightarrow{a} \mu$ is a transition of $\mathcal{B}$. ⊠

Claim 4 For all $\pi \in P$, $\sum_{a \in Act, \mu \in D} G(\pi)(a, \mu) = 1$.

PROOF: Choose $\pi \in P$ and let $n = |\pi|$. Then $\pi \in P_n$ and

$$
\begin{aligned}
& \sum_{a \in Act, \mu \in D} G(\pi)(a, \mu) && \{\text{def. } G\} \\
&= \sum_{a \in Act, \mu \in D_n} G'(\pi)(a, \mu) && \{\text{def. } G'\} \\
&= \sum_{a \in Act, \mu \in D_n} G_n(\pi)(a, \mu) && \{\text{def. } G_n\} \\
&= \sum_{a \in Act, \mu \in D_n} \lim_{k \to \infty} F^n_{\iota_{n+1}(k)}(\pi)(a, \mu) && \{\text{def. } F_i^n\} \\
&= \sum_{a \in Act, \mu \in D_n} \lim_{k \to \infty} F_{\iota_{n+1}(k)}(\pi)(a, \mu) && \{\text{finite sum}\} \\
&= \lim_{k \to \infty} \sum_{a \in Act, \mu \in D_n} F_{\iota_{n+1}(k)}(\pi)(a, \mu) && \{\text{def. } Act, D_n\} \\
&= \lim_{k \to \infty} \sum_{a \in Act, \mu \in D} F_{\iota_{n+1}(k)}(\pi)(a, \mu) && \{F_i \text{ adversary}\} \\
&= \lim_{k \to \infty} 1 \\
&= 1
\end{aligned}
$$

⊠

Note that the following claim concerns G and F_i, which are adversaries. In contrast, G_n and F^n_k are just functions, not adversaries.

Claim 5 $\mathbf{Q}^G(\pi) = \lim_{k \to \infty} \mathbf{Q}^{F_{\iota(n)(k)}}(\pi)$ for all $\pi \in Path^*(\delta\mathcal{B})$ with $|\pi| = n$.

PROOF: By induction on n.

- Then case $n = 1$ follows immediately from the fact that $\mathbf{Q}^E(s_0) = 1$ for all adversaries E.
- Case $n + 1$. Let π' be a path of length $n + 1$ and write $\pi' = \pi \overset{a,\mu}{\rightsquigarrow} s$. Then $\pi \in P_n$, $a \in Act$, $\mu \in D_n$ and

$$
\begin{aligned}
& \mathbf{Q}^G(\pi') \\
&= \mathbf{Q}^G(\pi \overset{a,\mu}{\rightsquigarrow} s) && \{\text{def. } \mathbf{Q}\} \\
&= \mathbf{Q}^G(\pi) \cdot G(\pi)(a, \mu) \cdot \mu(s) && \{\text{IH}, |\pi| = n\} \\
&= \lim_{k \to \infty} \mathbf{Q}^{F_{\iota(n)(k)}}(\pi) \cdot G_n(\pi)(a, \mu) \cdot \mu(s) && \{\text{def. } G_n\} \\
&= \lim_{k \to \infty} \mathbf{Q}^{F_{\iota(n)(k)}}(\pi) \cdot \lim_{k \to \infty} F_{\iota(n)(k)}(\pi)(a, \mu) \cdot \mu(s) \\
&= \lim_{k \to \infty} \mathbf{Q}^{F_{\iota(n)(k)}}(\pi) \cdot F_{\iota(n)(k)}(\pi)(a, \mu) \cdot \mu(s) && \{\text{def. } \mathbf{Q}\} \\
&= \lim_{k \to \infty} \mathbf{Q}^{F_{\iota(n+1)(k)}}(\pi \overset{a,\mu}{\rightsquigarrow} s) \\
&= \lim_{k \to \infty} \mathbf{Q}^{F_{\iota(n+1)(k)}}(\pi')
\end{aligned}
$$

⊠

Claim 6 $\mathbf{Q}^E(\pi) = \lim_{k\to\infty} \mathbf{Q}^{E_{\iota(n)(k)}}(\pi)$ for all n and π.

PROOF: Since $\iota(n)$ is an index function, we have $\lim_{k\to\infty} \iota(n)(k) = \infty$ and therefore $E_{\iota(n)(k)}[\pi] = E[\pi]$ for $\iota(n)(k) \geq |\pi|$. So, $\lim_{k\to\infty} \mathbf{Q}^{E_{\iota(n)(k)}}(\pi) = \mathbf{Q}^E(\pi)$. ⊠

The following is an immediate consequence of the previous claim.

Claim 7 $\mathbf{P}_E[C_\alpha] = \lim_{k\to\infty} \mathbf{P}_{E_{\iota(n)(k)}}[C_\alpha]$, for all α.

Claim 8 $\mathit{trdistr}(G) = \mathit{trdistr}(E)$.

PROOF: Let $H_1 = \mathit{trdistr}(G)$ and $H_2 = \mathit{trdistr}(E)$. It suffices to show that $\mathbf{P}_{H_1}[C_\alpha] = \mathbf{P}_{H_2}[C_\alpha]$ for all $\alpha \in Act^*$. Let $n = |\alpha|$.

$$
\begin{aligned}
\mathbf{P}_{H_1}[C_\alpha] &= \sum_{\pi \mid trace(\pi) \in C_\alpha} \mathbf{P}_G[\pi] \\
&= \sum_{\pi \mid trace(\pi) = \alpha \wedge |\pi| = n} \mathbf{P}_G[C_\pi] && \{\text{def. } C_\pi\} \\
&= \sum_{\pi \mid trace(\pi) = \alpha \wedge |\pi| = n} \mathbf{Q}^G(\pi) && \{\text{Claim 5, } |\pi|\text{=n}\} \\
&= \sum_{\pi \mid trace(\pi) = \alpha \wedge |\pi| = n} \lim_{k\to\infty} \mathbf{Q}^{F_{\iota(n)(k)}}(\pi) && \{\text{finite sum}\} \\
&= \lim_{k\to\infty} \sum_{\pi \mid trace(\pi) = \alpha \wedge |\pi| = n} \mathbf{Q}^{F_{\iota(n)(k)}}(\pi) && \{\text{def. } C_\alpha, \mathbf{P}_{F_i}\} \\
&= \lim_{k\to\infty} \mathbf{P}_{F_{\iota(n)(k)}}[C_\alpha] && \{\mathit{trdistr}(F_i) = \mathit{trdistr}(E_i)\} \\
&= \lim_{k\to\infty} \mathbf{P}_{E_{\iota(n)(k)}}[C_\alpha] && \{\text{Claim 7}\} \\
&= \mathbf{P}_E[C_\alpha]
\end{aligned}
$$

Note that the set $\{\pi \mid \mathit{trace}(\pi) = \alpha \wedge |\pi| = n\}$ is finite, because $\mathcal{A}$ is finitely branching and hence the summations above are all finite. ⊠

□

4.4 Characterization of Testing Preorder

The operational behavior of trace distribution machines described in Section 4.1 is specified accurately by the notion of a (partial, randomized) adversary, introduced in Definition 4.2.4. A trace distribution machine has to choose an execution path within some probabilistic automaton $\mathcal{A}$. For a given state, the choice which transition to take may depend on the execution history. Also, a probabilistic mechanism may be used to resolve the choice between the various transitions that are available. At any point during execution, there is a certain probability that the environment (the person doing the experiments) turns the machine off, i.e., aborts the execution via a halting transition.

Define an *observation* O *of depth* k *and width* m to be an element of $(Act^k)^m$, i.e., a sequence consisting of m sequences of actions of length k. An observation describes what an observer potentially may record when running m times an experiment of length k on the trace distribution machine. Note that if, during a run, the machine halts before k observable actions have been performed, we can still obtain a sequence of k actions by attaching a number of δ actions at the end.

We write $\mathit{freq}(O)$ for the function in $Act^k \to \mathbb{Q}$ that assigns to observation O a function which assigns to each sequence of actions of length k the number of times it occurs in O divided by m. Note that $\mathit{freq}(O)$ is a probability distribution . We will base our statistical analysis on $\mathit{freq}(O)$ rather than just O. This means we ignore some of the information contained in observations, information that more advanced statistical methods may want to explore. If, for instance, we consider the observation O of depth one and width 2000 that consists of 1000 $head$ actions followed by 1000 $tail$ actions, then it is quite unlikely that this will be an observation of a trace distribution machine implementing a fair coin. However, the frequency function $\mathit{freq}(O)$ can very well be generated by a fair coin.

Now choose a level of significance $\alpha \in (0,1)$. Let $E_1, \ldots, E_m$ be adversaries of $\mathcal{A}$ that halt after k steps. Let H_0 be the null hypothesis, which states that O has been generated using adversaries $E_1, \ldots, E_m$. We define the critical region $K_{E_1,\ldots,E_m}$ to be the complement of the largest hypercube K around the expected value for $\mathit{freq}(O)$, assuming H_0 and with $\mathbf{P}_{H_0}[K] \leq \alpha$.[3] Now we come to one of the key definitions of this chapter.

Definition 4.4.1 We say that O is an *observation of* $\mathcal{A}$ if

$$\exists E_1, \ldots, E_m \in Adv(\mathcal{A}, k).\mathit{freq}(O) \notin K_{E_1,\ldots,E_m}$$

Here $Adv(\mathcal{A}, k)$ denotes the set of adversaries of $\mathcal{A}$ that halt after k steps. We write $Obs(\mathcal{A})$ for the set of observations of $\mathcal{A}$ (for some depth k and width m).

Thus we only include those observations for which it is likely (given the chosen level of significance) that they are generated by some adversaries of $\mathcal{A}$. If O is an observation of $\mathcal{A}$ then its frequency distribution is contained in the hypercube with probability mass α surrounding the expected value of the frequency distribution for a (well-chosen) series of adversaries of $\mathcal{A}$. We can now prove our main characterization theorem.

Theorem 4.4.2 $Obs(\mathcal{A}) \subseteq Obs(\mathcal{B}) \;\Leftrightarrow\; \mathcal{A} \sqsubseteq_{\mathrm{TD}} \mathcal{B}$.

PROOF: (Sketch)
"$\Rightarrow$" Assume that $\mathcal{A} \not\sqsubseteq_{\mathrm{TD}} \mathcal{B}$. We show that $Obs(\mathcal{A}) \not\subseteq Obs(\mathcal{B})$. By Theorem 4.3.1, there exists a k such that $\mathcal{A} \not\sqsubseteq^k_{\mathrm{TD}} \mathcal{B}$. This means $\mathit{trd}(\mathcal{A}, k) \not\subseteq \mathit{trd}(\mathcal{B}, k)$. Let H be a trace distribution in $\mathit{trd}(\mathcal{A}, k)$ that is not a trace distribution in $\mathit{trd}(\mathcal{B}, k)$ and which is generated by an adversary E of $\mathcal{A}$ that halts after k steps. Using Lemma 4.2.6 we may assume, without loss of generality that E is a deterministic adversary. Also, by the same lemma, we may assume that adversaries that generate traces in $\mathit{trd}(\mathcal{B}, k)$ can be written as convex combinations of (a finite number of) deterministic adversaries. This means that there

[3] This choice of using hypercubes is actually quite impractical, but serves well the theoretical aims we have for this chapter.

is minimal distance d between H and *any* trace distribution in $trd(\mathcal{B}, k)$. By choosing m sufficiently large we obtain, using Chebychev's inequality, $\bar{K}_{E_1,\ldots,E_m} \cap \bar{K}_{F_1,\ldots,F_m} = \emptyset$, for $E_i = E$ and $F_1, \ldots, F_m$ arbitrary adversaries from $Adv(\mathcal{B}, k)$. In addition, we may assume that there exists an observation O with $freq(O) \in \bar{K}_{E_1,\ldots,E_m}$. Clearly, O is an observation of $\mathcal{A}$. However, by construction, O is not an observation of $\mathcal{B}$.

"$\Leftarrow$" Assume that $\mathcal{A} \sqsubseteq_{\mathrm{TD}} \mathcal{B}$. Suppose O is an observation of $\mathcal{A}$ with depth k and width m. Then, by definition, there exist adversaries $E_1, \ldots, E_m \in Adv(\mathcal{A}, k)$ such that $freq(O) \notin K_{E_1,\ldots,E_m}$. By the assumption, we can choose adversaries $F_1, \ldots, F_m \in Adv(\mathcal{B}, k)$ with, for all i, $trdistr(E_i) = trdistr(F_i)$. Since the critical region $K_{E_1,\ldots,E_m}$ can actually be computed from $trdistr(E_1), \ldots, trdistr(E_m)$ (and similarly for $K_{F_1,\ldots,F_m}$), it follows that $K_{E_1,\ldots,E_m} = K_{F_1,\ldots,F_m}$. Hence it follows that $O \in Obs(\mathcal{B})$, as required. $\square$

Acknowledgement The ideas worked out in this chapter were presented in preliminary form at the seminar "Probabilistic Methods in Verification", which took place from April 30 – May 5, 2000, in Schloss Dagstuhl, Germany. We thank the organizers, Moshe Vardi, Marta Kwiatkowska, Christoph Meinel and Ulrich Herzog, for inviting us to participate in this inspiring meeting.

CHAPTER 5
Probabilistic Bisimulation and Simulation

Abstract We consider probabilistic systems with nondeterminism and study several simulation and bisimulation relations on them. The purpose of this chapter is twofold. First, we give an overview of several types of *strong* and *weak (bi–)simulations* known in the literature. Second, we introduce a novel type of (bi–)simulation, called *delay (bi–)simulation* and show that it can be characterized by means of *norm functions* [GV98]. We discuss the connection between the several (bi–)simulation relations that result from the use of randomized versus deterministic adversaries and combining finitely versus infinitely many internal steps. While for the weak (bi-)simulations, the decidability is still an open problem, we present algorithms to decide the delay (bi-)simulation relations that run in polynomial time and space.

5.1 Introduction

Correctness proofs for concurrent systems are often based on simulation and bisimulation relations. These have been developed for and applied to various types of systems [LV95, AL91, RE98], including timed systems [LV96b] and probabilistic systems [LS91, HJ94, SL95].

Simulations and bisimulation relations are — for each type of systems mentioned — relations on the state space of the system that compare the stepwise behavior of related states. As we will formally define later in this chapter, *bisimulation relations* are equivalences on the state space of a system such that two related states exhibit the same stepwise behavior. In other words, states that are related via a bisimulation relation can be regarded as equivalent. *Simulation relations* are preorders $\sqsubseteq$ on the state space such that if $s \sqsubseteq t$, then t can mimic all stepwise behavior of s. The converse need not be true, so t may perform steps that cannot be mimicked by s. If we have $s \sqsubseteq t$, then we can say that s behaves as good as t. One can also use (bi–)simulation to compare systems, rather than states, by defining that two systems are (bi–)simulation if their start states are so.

Different notions of (bi–)simulation exist, depending on what one considers as the system's stepwise behavior. Typically, one distinguishes between strong and weak (bi–)simulation. Weak (bi–)simulation abstracts from internal computations, whereas strong (bi–)simulation takes all steps of the system into account, including internal ones.

This chapter focuses on probabilistic systems. We extensively study various simulation and bisimulation relations for these systems. We formulate the theory in a minor variation of the framework presented in Chapter 2. As before, the systems incorporate both probabilistic and nondeterministic choice. We consider several known (bi–)simulations from the literature for these systems, namely, strong, strong combined and weak (bi–)simulation. Moreover, we introduce a novel type of relations, which we call *delay (bi–)simulation*. We define two variants of them, *finitary* and ω–(bi–)simulation.

There are several ways in which simulation techniques can be used in verification proofs. We discuss the two most important ones. Firstly, simulation relations have proven to be very successful in showing that a lower level description of a system correctly implements a higher level specification of it. Chapter 2 proposed $\sqsubseteq_{\mathrm{PTD}}$ as an implementation relation for probabilistic labeled transition systems. That is, a probabilistic system $\mathcal{A}$ correctly implements another one $\mathcal{B}$ if and only if $\mathcal{A} \sqsubseteq_{\mathrm{PTD}} \mathcal{B}$. However, a direct proof showing $\mathcal{A} \sqsubseteq_{\mathrm{PTD}} \mathcal{B}$ is often very complex because it involves reasoning about the *global* behavior of two systems $\mathcal{A}$ and $\mathcal{B}$ in terms of trace distributions. Simulation relations, instead, involve reasoning about the *local* behavior of a system, that is, in terms of states and transitions. Establishing $\mathcal{A} \sqsubseteq \mathcal{B}$ is therefore often far more easy than establishing $\mathcal{A} \sqsubseteq_{\mathrm{PTD}} \mathcal{B}$. And since the former implies the latter, simulation techniques are sound for proving $\mathcal{A} \sqsubseteq_{\mathrm{PTD}} \mathcal{B}$ and often reduce the verification effort drastically.

Secondly, bisimulation equivalences $\approx$ can be used to reduce a system to a smaller one. To do so, one replaces each state in a system S by its equivalence class in $\approx$, yielding a new system $S/\approx$. Then one can verify $S/\approx$ instead of S, provided that the bisimulation $\approx$ preserves the properties one verifies. This is a gain, because $S/\approx$ is often significantly smaller that S, especially for weak bisimulation relations. Moreover, bisimulation relations do preserve many properties of interest, so this reduction technique can be used in many cases.

To be applicable in verification, the relations have to meet certain properties. Firstly, as we have already seen, the use of simulation relations depends on their soundness for establishing $\sqsubseteq_{\mathrm{PTD}}$. Similarly, we have seen that the use of bisimulations depends on their preservation of the properties one wishes to verify. We have also seen that the use of (bi–)simulation is based on the fact that they are relatively easy to establish. This can be done with deductive methods to be carried out by hand, but preferably, a (bi–)simulation is decidable. This opens the way to automatic verification, which is both faster and more reliable than manual proofs. Finally, we want the (bi–)simulations to support two important techniques for keeping a verification process manageable. That is, they should allow for *modular verification*, in which one derives the correctness of the whole system from the correctness of its components. Moreover, they should allow for *stepwise verification*, in which one derives the correctness of a system via several intermediate systems. Technically, this means that a (bi–)simulation relation should be transitive and substitutive (with respect to parallel composition). The former is trivial when we consider states, but not when comparing systems.

At the end of this chapter, we will discuss to what extend the relations we study meet the properties that make them useful in verification.

Related work Apart for bisimulation and simulation relations, several other types of preorders and equivalences for probabilistic systems have been proposed in the literature. Most of the standard relations defined for nonprobabilistic systems have been extended to the probabilistic ones: see e.g. [Seg95a] for a trace–based relation, [YL92, JY01] for testing equivalences and [LS91, HJ90, Han91, Yi94, SL95, Seg95a, SV99b] for several types of (bi–)simulations. In the fully probabilistic setting, that is, for systems with probabilistic choice only, the equivalences are studied under several aspects (compositionality, axiomatization, decidability, logical characterizations, etc.), see e.g. [JS90, CC92, BJ91, HT92, KL92, Chr90b, BH97].

On the other hand, the treatment of equivalences and preorders for probabilistic systems is less well–understood. Due to the combination of nondeterminism and probability, the definitions are more complicated than the corresponding notions for nonprobabilistic or fully probabilistic systems. Even though some important issues (like compositionality and axiomatization) have been addressed in the above mentioned literature, research on algorithmic methods to decide the equivalence of two systems or to compute the quotient space are less common. For strong bisimulation [LS91] and strong simulation [SL95], polynomial-time algorithms have been presented in [BEM00]. To the best of our knowledge, the work [PSL98] is the first attempt to formulate an algorithmic method that deals with a weak equivalence for probabilistic processes with nondeterminism. Decidability of weak bisimulation à la [SL95] was an open problem for a long time; it was conjectured to be decidable in exponential time. Segala [SC01] has proposed a polynomial time algorithm that decides weak bisimulation. A publication of these results is currently in preparation. We are not aware of any complexity (or even decidability) result for any linear time relation on probabilistic systems with nondeterminism, e.g. trace distribution equivalence [Seg95a]. Obviously, since the trace distribution preorder à la Segala is a conservative extension of usual trace inclusion, the PSPACE–completeness result for trace inclusion of nonprobabilistic systems [KS90] yields a PSPACE–hardness for the trace distribution preorder.

Our contribution: We present a comprehensive study of several notions of strong, combined strong and weak (bi–)simulation in the probabilistic setting. We simplify existing definitions and present several important results known from the literature, such as compositionality and decidability, in a single framework.

The main contribution of our work is, however, the introduction of novel notions of finitary and ω (bi–)simulation. These relations are (in some sense) insensitive to internal transitions. Moreover, they are conservative extensions of *delay bisimulation* equivalence [Wei89, Gla93] for nonprobabilistic systems. This equivalence relies on the assumption that the simulation of a step of a system $\mathcal{A}$ by another process $\mathcal{B}$ might happen with a certain delay (i.e. after a sequence of internal transitions). Our notions of delay bisimulation can be characterized by a probabilistic variant of *norm functions* in the style of [GV98]. Intuitively, the norm functions specify bounds on the number of internal transitions that can be used by a step of $\mathcal{B}$ when simulating a step of $\mathcal{A}$. In the probabilistic setting, where the combination of internal transitions leads to a tree rather than a linear chain, the norm functions yield conditions on the length of the paths in the trees corresponding to a "delay transition". Using a modification of the traditional splitter/partitioning technique [KS90, PT87], we present polynomial time algorithms for computing the quotient spaces. We briefly discuss some other aspects (compositionality w.r.t. parallel composition and preservation of linear time properties).

Organization of the chapter: In Section 5.2, we present some notations concerning relations, partitions and distributions that we use throughout the chapter. Section 5.3 introduces the model for probabilistic systems which we consider and its behavior. Section 5.4 recalls the definition of strong bisimulation [LS91] and the several variants of weak and branching bisimulations [SL95, Seg95b]. The novel delay (bi–)simulation relations and their characterization by norm functions are presented in Section 5.5. Section 5.6 treats several important properties, including compositionality and the interconnection between the (bi–)simulation relations. Section 5.7 summarizes the results on the decidability of the various relations that have been established in the literature and presents polynomial-time algorithms for computing the delay (bi–)simulation relations. In the conclusion (Section 5.8), we summarize the results and present some conclusions.

5.2 Preliminaries

This section briefly summarizes some standard notations that are used throughout this chapter.

The domain of a partial function: Let X, Y be sets. We denote the set of partial functions from X to Y by $X \hookrightarrow Y$. For a partial function $f : X \hookrightarrow Y$, let $\mathrm{Dom}(f)$ denote the *domain* of f, that is the set of all $x \in X$ such that $f(x)$ is defined.

Sequences: Let X be a set. We denote the set of finite sequences over X by X^*, the set of infinite sequences over X by X^ω and the set $X^* \cup X^\omega$ by X^∞. The empty sequence is written by ε. We denote the prefix relation on X^∞ by $\sqsubseteq^X$ and the proper prefix relation by $\sqsubset^X$. When X is clear from the context, we simply write $\sqsubseteq$ and $\sqsubset$.

Equivalences: If R is an equivalence relation on a set X, then X/R denotes the quotient space of X with respect to R and $[x]_R$ denotes the equivalence class of x with respect to R.

Partitions: Let X be a nonempty finite set and $x, x' \in X$. A *partition* of X is a set $\chi = \{X_1, \dots, X_l\}$ consisting of pairwise disjoint, nonempty subsets of X such that $X_1 \cup \dots \cup X_l = X$. The elements X_i of χ are called *blocks* of χ and $[x]_\chi$ denotes the unique block X_i of χ that contains x.

Clearly, there is a one–to–one correspondence between the equivalences on X and the partitions of X. For any equivalence relation R on X, the quotient space X/R is a partition of X. Vice versa, each partition χ induces an equivalence R_χ by identifying those elements that are contained in the same block of χ. A partition χ of X is called *finer* than a partition χ' of X (in which case we also say that χ' is *coarser* than χ) iff R_χ is finer than $R_{\chi'}$, which means that $R_{\chi'} \subseteq R_\chi$, or equivalently, any block in χ' can be written as a union of blocks in χ. The partition χ is called *strictly finer* than χ' (or χ' *strictly coarser* than χ) if χ is finer than but not equal to χ'.

Distributions: Let X be a nonempty set. A *(probability) distribution* on X is a function $\mu : X \to [0, 1]$ such that $\mu(x) > 0$ for countably[1] many elements $x \in X$ and $\sum_{x \in X} \mu(x) = 1$. Let $supp(\mu)$ denote the *support* of μ, i.e. the set of elements $x \in X$ with $\mu(x) > 0$. If $\emptyset \neq A \subseteq X$ then $\mu[A] = \sum_{x \in A} \mu(x)$. Moreover, we put $\mu[\emptyset] = 0$. For $x \in X$, μ_x^1 denotes the unique distribution on X with $\mu(x) = 1$, which is called the *Dirac distribution* over x. If $(p_i)_{i \in I}$ is a countable family of real numbers $p_i \in [0, 1]$ such that $\sum_{i \in I} p_i = 1$ and $(\mu_i)_{i \in I}$ a family of distributions on X, then the distribution given by $x \mapsto \sum_{i \in I} p_i \cdot \mu_i(x)$ is called a *convex combination* of the distributions μ_i, $i \in I$. We denote the collection of all distributions on X by $Distr(X)$.

The equivalence $\equiv_R$: If R is an equivalence on X, then the induced equivalence $\equiv_R$ on $Distr(X)$ is given by: $\mu \equiv_R \mu'$ iff $\mu[C] = \mu'[C]$ for all $C \in X/R$.

[1] In our definition, the countable sets subsume the finite sets.

Weight functions: Let R be a binary relation on X and μ, $\mu' \in \mathit{Distr}(X)$. A *weight function* for (μ, μ') with respect to R is a function $wgt : X \times X \to [0, 1]$ such that $wgt(x, x') > 0$ implies $(x, x') \in R$ and, for all x, $x' \in X$:

$$\sum_{y' \in X} wgt(x, y') \; = \; \mu(x), \quad \sum_{y \in X} wgt(y, x') \; = \; \mu'(x').$$

Intuitively, a weight function shows how we can distribute the probability $\mu(x)$ among the related states x' in such a way that $\mu'(x')$ equals the total amount of probability gets distributed this way. In fact, the function wgt is a probability distribution on $X \times X$ such that the probability to pick an element (x, x') in R is one, the probability to pick an element whose first component is x equals $\mu(x)$ and the probability to pick an element whose second component is x' equals $\mu'(x')$.

The relation $\sqsubseteq_R$ and the set $\mu\!\uparrow_R$: We write $\mu \sqsubseteq_R \mu'$ iff there exists a weight function for (μ, μ') with respect to R. We define $\mu\!\uparrow_R = \{\mu' \in \mathit{Distr}(X) : \mu \sqsubseteq_R \mu'\}$. We call $\mu\!\uparrow_R$ the *cone* above μ w.r.t. $\sqsubseteq_R$.

The following proposition collects some elementary properties of the relations $\equiv_R$ and $\sqsubseteq_R$.

Proposition 5.2.1 *Let μ, μ', $\mu'' \in \mathit{Distr}(X)$ and R, R_1, R_2 binary relations on X.*

1. *If R is an equivalence relation, then $\mu \equiv_R \mu' \iff \mu \sqsubseteq_R \mu'$.*
2. *If $\mu \sqsubseteq_{R_1} \mu'$ and $\mu' \sqsubseteq_{R_2} \mu''$, then $\mu \sqsubseteq_{R_1 \circ R_2} \mu''$.*[2]
3. *If $\mu \sqsubseteq_R \mu'$, then $\mu' \sqsubseteq_{R^{-1}} \mu$.*[3]
4. *Let $\mu = \sum_{i \in I} p_i \cdot \mu_i$ be a convex combination over the distributions μ_i. If $\mu_i \sqsubseteq_R \mu'$ for all $i \in I$, then $\mu \sqsubseteq_R \mu'$.*
5. *If $R_1 \subseteq R_2$, then $\mu \sqsubseteq_{R_1} \mu' \implies \mu \sqsubseteq_{R_2} \mu'$.*

PROOF:

1. ($\implies$) Let $\mu \equiv_R \mu'$. Define a function wgt for (μ, μ') with respect to R as follows. If $(x, x') \notin R$ then put $wgt(x, x') = 0$. If $(x, x') \in R$ then put

$$wgt(x, x') \; = \; \frac{\mu(x) \cdot \mu'(x')}{\mu[[x]_R]}.$$

Using that $\mu[[x]_R] = \mu'[[x]_R]$ for each x, one easily shows that wgt is indeed a weight function for (μ, μ').

[2]Recall that $R_1 \circ R_2$ consists of all pairs (x, x'') where $(x, x') \in R_1$ and $(x', x'') \in R_2$ for some $x' \in X$.
[3]Recall that R^{-1} consists of all pairs (x', x) where $(x, x') \in R$.

($\Longleftarrow$) Assume $\mu \sqsubseteq_R \mu'$. Let *wgt* be a weight function for (μ, μ'). Since $wgt(x, x') = 0$ for $(x, x') \notin R$, we have $\mu(x) = \sum_{y' \in C} wgt(x, y')$ and $\mu'(x') = \sum_{y \in C} wgt(y, x')$ for all x, x' and C with $x, x' \in C$. From this, we get

$$\begin{aligned} \mu[C] = \sum_{x \in C} \mu(x) &= \sum_{x \in C} \sum_{x' \in C} wgt(x, x') = \sum_{x' \in C} \sum_{x \in C} wgt(x, x') \\ &= \sum_{x' \in C} \mu'(x') = \mu'[C]. \end{aligned}$$

Thus, $\mu \equiv_R \mu'$.

2. Let wgt_1 and wgt_2 be weight functions for (μ, μ') with respect to R_1 and for (μ', μ'') with respect to R_2 respectively. It is easy to see that

$$wgt(x, x'') = \sum_{x' \in supp(\mu')} \frac{wgt_1(x, x') \cdot wgt_2(x', x'')}{\mu'(x')}$$

is a weight function for (μ, μ'') with respect to $R_1 \circ R_2$.

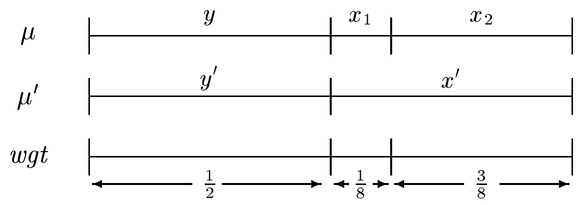
3–5. Easy verification.

□

Example 5.2.2 Let $X = \{x_1, x_2, x', y, y'\}$ and let R be the smallest equivalence relation on X which identifies the elements x', x_1 and x_2 and the elements y and y' (i.e. $X/R = \{C_x, C_y\}$ where $C_x = \{x_1, x_2, x'\}$ and $C_y = \{y, y'\}$). Let μ and μ' be the distributions with

$$\mu'(x') = \mu'(y') = \tfrac{1}{2} \text{ and } \mu(x_1) = \tfrac{1}{8}, \mu(x_2) = \tfrac{3}{8}, \mu(y) = \tfrac{1}{2}\,.$$

Then, $\mu[C_x] = \mu(x_1) + \mu(x_2) + \mu(x') = \frac{1}{2} = \mu'(x_1) + \mu'(x_2) + \mu'(x') = \mu'[C_x]$ and $\mu[C_y] = \mu(y) + \mu(y') = \frac{1}{2} = \mu'(y) + \mu'(y') = \mu'[C_y]$. Thus, $\mu \equiv_R \mu'$ and therefore also $\mu \sqsubseteq_R \mu'$. A weight function for (μ, μ') with respect to R is given by $wgt(y, y') = 1/2$, $wgt(x_1, x') = 1/8$, $wgt(x_2, x') = 3/8$ (and $wgt(\cdot) = 0$ in all other cases), see the picture below.



5.3 Probabilistic Systems

This section introduces the probabilistic systems we study and their behavior. The basic ideas behind this model have been explained in Chapter 2. This chapter treats more technical details, although several notions are repeated.

Throughout the paper, we deal with a fixed set of action symbols including a special action symbol τ that stands for any internal activity, i.e. any activity that is not observable by the environment.

Notation 5.3.1 (The action set Act_τ) Let Act_τ be a finite set of actions containing a special action τ (called the *internal* action). The other actions are called *visible*. We write $Act = Act_\tau \setminus \{\tau\}$. We define an embedding $\widehat{\ }: Act_\tau \to Act_\varepsilon$, by $\widehat{\tau} = \varepsilon$ and $\widehat{a} = a$ for any visible action $a \in Act$.

Definition 5.3.2 (Probabilistic system) A *probabilistic system* $\mathcal{S}$ is a pair (S, Steps) where S is a finite[4] set of states and

$$\mathit{Steps} : S \to 2^{Act_\tau \times \mathit{Distr}(S)}$$

a function which assigns to each state $s \in S$ a finite set $\mathit{Steps}(s)$ of pairs (a, μ) consisting of an action $a \in Act_\tau$ and a distribution μ on S. We define $\mathit{Steps}_a(s) = \{\mu : (a, \mu) \in \mathit{Steps}(s)\}$. The function *Steps* induces a transition relation $\longrightarrow \subseteq S \times Act_\tau \times \mathit{Distr}(S)$ which is defined by $s \xrightarrow{a} \mu$ iff $(a, \mu) \in \mathit{Steps}(s)$. A state s is called *terminal* iff there is no possible next step in s, i.e. if $\mathit{Steps}(s) = \emptyset$. We often write $s \xrightarrow{a} t$ for $s \xrightarrow{a} \mu_t^1$.

The function *Steps* represents the nondeterministic alternatives in each state: each transition $s \xrightarrow{a} \mu$ represents the possibility to move from state s with an a–action to a next state t, where the probability to reach t is given by $\mu(t)$.

Remark 5.3.3 ((Non-probabilistic) labeled transition systems) Any finite (nonprobabilistic) labeled transition system $(S, \longrightarrow)$ (where $\longrightarrow \subseteq S \times Act_\tau \times S$) can be identified with the probabilistic system (S, Steps) where $\mathit{Steps}(s) = \{(a, \mu_t^1) : s \xrightarrow{a} t\}$.

Remark 5.3.4 (Probabilistic automata) A *probabilistic automaton* is a probabilistic system equipped with an initial state, i.e. a tuple $\mathcal{A} = (S, \mathit{Steps}, s_{init})$ consisting of a probabilistic system (S, Steps) and an initial state $s_{init} \in S$. Sections 5.4 and 5.5 define several relations on the state space S of a probabilistic system. These can easily be adapted to relations on probabilistic automata by defining that two probabilistic automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are related if and only if their initial states are related when we view them as states in disjoint union $\mathcal{A}_1 \uplus \mathcal{A}_2$[5].

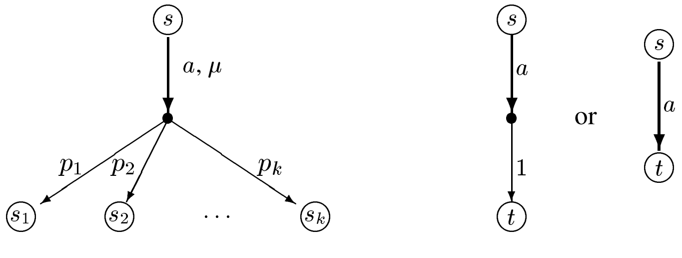


Figure 5.1: The transitions $s \xrightarrow{a} \mu$ and $s \xrightarrow{a} \mu_t^1$

We depict probabilistic systems as follows. We use circles for the states. Thick lines stand for the outgoing transitions from a state. The thick line corresponding to a transition $s \xrightarrow{a} \mu$

[4]For algorithmic purposes, we restrict our attention to finite probabilistic systems. Most of the results on adversaries and (bi-)simulation relations also hold for infinite systems.

[5]For $\mathcal{A}_i = (S_i, \mathit{Steps}_i, s_i)$, the disjoint union $\mathcal{A}_1 \uplus \mathcal{A}_2$ is the system (S, Steps). Here $S = S_1 \uplus S_2$ is the disjoint union of S_1 and S_2 and $\mathit{Steps} = \mathit{Steps}_1 \cup \mathit{Steps}_2$, where we identify each element in $(a, \mu) \in \mathit{Steps}_i$ with the element $(a, \mu') \in Act_\tau \times \mathit{Distr}(S)$ given by $\mu'(s) = \mu(s)$ if $s \in S_i$ and $\mu'(s) = 0$ if $s \notin S_i$.

is directed and ends in a small circle, which represents the probabilistic choice but which is not a state in the system. The picture on the left of Figure 5.1 stands for a transition $s \xrightarrow{a} \mu$ where $supp(\mu) = \{s_1, \ldots, s_k\}$ and $\mu(s_i) = p_i > 0$, $i = 1, \ldots, k$. Sometimes we omit the distribution and just write a rather than a, μ. Transitions of the form $s \xrightarrow{a} \mu$ are depicted as shown in the pictures on the right. In our examples, we often depict the unfolding of a probabilistic system into a tree-like structure and duplicate certain states. Moreover, we often omit the outgoing transitions of certain states if they are not important.

Example 5.3.5 [Simple communication system] Figure 5.2 shows a probabilistic system that describes a toy communication system consisting of a sender (which *produces* certain messages and tries to submit the messages via an unreliable medium) and a receiver (which acknowledges the receipt and *consumes* the received messages). The failure rate of the medium is 1%; more precisely, with probability $1/100$ the medium looses the messages in which case the sender resends the message. For simplicity, we assume that both the sender and the receiver work with mailing boxes that cannot hold more than one message at any time. Thus, if the sender has produced a message m then the next message cannot be produced before m has been delivered correctly; similarly, the medium cannot be activated as long as there is an unread message in the mailing box of the receiver (i.e. as long as the acknowledgement for the last message has not arrived yet). We use the following four states:

- s_{init}: the state where the sender produces a message and passes the message to the medium.
- s_{del}: the state in which the medium tries to deliver the message (via an internal action).
- s_{ok}: the state reached when the message is delivered correctly.
- s_{ack}: the state in which the receiver consumes the message (i.e. reads and processes the message and acknowledges the receipt).

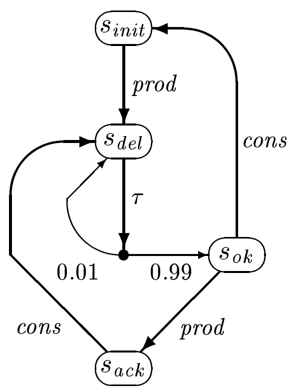


Figure 5.2: The simple communication system

5.3.1 Paths

The executions of probabilistic systems are given by the paths in the underlying nonprobabilistic LTS. Paths in probabilistic systems arise by resolving both the nondeterministic and probabilistic choices. We distinguish between finite paths (representing execution fragments) and fullpaths (representing complete executions). The latter are either finite paths that end up in a terminal state or infinite paths.

Definition 5.3.6 (paths) A *path* of a probabilistic system $\mathcal{S} = (S, \mathit{Steps})$ is an alternating, finite sequence

$$\sigma \;=\; s_0 \overset{a_1,\mu_1}{\leadsto} s_1 \overset{a_2,\mu_2}{\leadsto} s_2 \overset{a_3,\mu_3}{\leadsto} \ldots \overset{a_l,\mu_l}{\leadsto} s_l$$

or an alternating, infinite sequence

$$\sigma \;=\; s_0 \overset{a_1,\mu_1}{\leadsto} s_1 \overset{a_2,\mu_2}{\leadsto} s_2 \overset{a_3,\mu_3}{\leadsto} \ldots$$

where $s_i \in S$, $s_i \xrightarrow{a_{i+1}} \mu_{i+1}$, $s_{i+1} \in \mathit{supp}(\mu_{i+1})$. We often omit the distributions μ_i and just write $\ldots s_{i-1} \overset{a_i}{\leadsto} s_i \ldots$. We define the following notations for paths.

- Let $\mathit{length}(\sigma) = l$ if σ is finite and $\mathit{length}(\sigma) = \infty$ otherwise. Thus, the length of a path denotes its number actions.
- Let $\mathit{first}(()\sigma) = s_0$ and, for finite paths, $\mathit{last}(\sigma) = s_l$,
- We can retrieve respectively the i^{th} state, step and action by defining $\mathit{state}(\sigma, i) = s_i$, $\mathit{Step}(\sigma, i) = (a_i, \mu_i)$ and $\mathit{act}(\sigma, i) = a_i$,
- $\mathit{prefix}(\sigma, i)$ denotes the prefix of length i, i.e.

  $$\mathit{prefix}(\sigma, i) \;=\; s_0 \overset{a_1,\mu_1}{\leadsto} s_1 \overset{a_2,\mu_2}{\leadsto} s_2 \overset{a_3,\mu_3}{\leadsto} \ldots \overset{a_i,\mu_i}{\leadsto} s_i, \quad \text{if } i < \mathit{length}(\sigma),$$

  and otherwise we put $\mathit{prefix}(\sigma, i) = \sigma$.
- Let $\mathit{word}(\sigma) = a_1 a_2 \ldots$. be the finite or infinite word over Act_τ associated with σ.

Definition 5.3.7 (Fullpaths) A *fullpath* of a probabilistic system $\mathcal{S}$ is either an infinite path or a finite path σ, where $\mathit{last}(\sigma)$ is a terminal state. Let $\mathit{Path}^{\mathcal{S}}_{\mathit{ful}}$ denote the set of all fullpaths in $\mathcal{S}$ and $\mathit{Path}^{\mathcal{S}}_{\mathit{ful}}(s)$ the set of fullpaths starting in s (i.e. with $\mathit{first}(()\pi) = s$). Similarly, $\mathit{Path}^{\mathcal{S}}_{*}$ denotes the set of all finite paths in $\mathcal{S}$ and $\mathit{Path}^{\mathcal{S}}_{*}(s)$ the set of finite paths in $\mathcal{S}$ starting in s.

Example 5.3.8 Consider the probabilistic system in Figure 5.3. This system contains infinitely many finite paths and one infinite path starting in s. The paths starting in s are given by

$$\begin{array}{ll}
s, & s \overset{\tau,\nu}{\leadsto} s \overset{\tau,\nu}{\leadsto} s \ldots \overset{\tau,\nu}{\leadsto} s \overset{\tau,\nu}{\leadsto} s' \\
s \overset{a,\rho}{\leadsto} t & s \overset{\tau,\nu}{\leadsto} s \overset{\tau,\nu}{\leadsto} s \ldots \overset{\tau,\nu}{\leadsto} s \overset{\tau,\nu}{\leadsto} s' \overset{a,\mu}{\leadsto} u \\
s \overset{\tau,\nu}{\leadsto} s \overset{\tau,\nu}{\leadsto} s \ldots \overset{\tau,\nu}{\leadsto} s & s \overset{\tau,\nu}{\leadsto} s \overset{\tau,\nu}{\leadsto} s \overset{\tau,\nu}{\leadsto} s \ldots.
\end{array}$$

The only fullpaths starting in s are $s \overset{a,\rho}{\leadsto} t$ and $s \overset{\tau,\nu}{\leadsto} s \overset{\tau,\nu}{\leadsto} s \overset{\tau,\nu}{\leadsto} s \ldots$.

5.3.2 Adversaries

Adversaries (also called *schedulers* [Var85] or *policies* in the theory of MDPs) are entities that resolve the nondeterministic choices. Being in a state of the system, an adversary determines the next step to be taken. This step may depend on the history, i.e. the path leading to the current state. Moreover, the adversary may choose randomly which of the possible next steps is chosen. Thus, given a finite path π ending in a state s and each transition $s \xrightarrow{a} \mu$, a randomized partial adversary A yields a probability $A(\pi)(a,\mu)$, specifying the probability with which A schedules $s \xrightarrow{a} \mu$. The remaining value $A(\sigma)(\perp)$ is the probability for an interruption.

We define several special classes of adversaries. *Deterministic* adversaries are adversaries that schedule a unique next step with probability one; *total* adversaries, or just adversaries, schedule a step whenever possible, i.e. only choose $\perp$ in a terminal state of the system. Furthermore, we define the classes of *memoryless*, *finitary* and *almost finitary* adversaries.

Definition 5.3.9 ((Randomized) partial adversary) A *(randomized) partial adversary* of a probabilistic system $\mathcal{S}$ is a function

$$A : \mathit{Path}^* \to \mathit{Distr}\,(\mathit{Act}_\tau \times \mathit{Distr}(S) \,\cup\, \{\perp\})$$

such that, for each finite path σ: if $A(\sigma)(a,\mu) > 0$ then $(a,\mu) \in \mathit{Steps}(\mathit{last}(\sigma))$.

Definition 5.3.10 (Deterministic and total adversary) A partial adversary D is called *deterministic* if for all $\sigma \in \mathit{Path}_*^D$, $D(\sigma)$ yields a Dirac distribution. As before, we often write $D(\sigma) = x$ for $D(\sigma) = \mu_x^1$. A *total adversary* is a partial adversary A where $A(\sigma)(\perp) = 0$ for all finite paths σ where $\mathit{last}(\sigma)$ is nonterminal. Let $RAdv_\perp$ denote the set of all randomized partial adversaries, $RAdv$ the set of all randomized adversaries, $DAdv_\perp$ the set of all deterministic partial adversaries and finally $DAdv$ the set of all deterministic adversaries.

Since we deal mainly with partial adversaries, we will often leave out the adjective partial and simply speak of adversaries rather than of partial adversaries.

Definition 5.3.11 (Memoryless adversary) A partial adversary A is called *memoryless* if it depends only on the last state, i.e. $\mathit{last}(\sigma) = \mathit{last}(\sigma') \implies A(\sigma) = A(\sigma')$. Therefore, we write $A(\mathit{last}(\sigma))$ rather than of $A(\sigma)$ if A is memoryless.

Example 5.3.12 Consider Figure 5.3. We illustrate the definitions above by three adversaries.

- The function D on finite paths given by $D(s) = (a,\rho)$ and $D(\sigma) = \perp$ for $\sigma \neq s$ is a deterministic partial adversary. It is not total because $D(s') = \perp$, whereas s' is not terminal.
- Furthermore, define the function A by $A(s)(a,\rho) = \frac{1}{3}$, $A(s)(\tau,\nu) = \frac{1}{3}$, $A(s)(\perp) = \frac{1}{3}$ and $A(\sigma) = \perp$ for $\sigma \neq s$. Then A is a randomized partial adversary.
- Define the deterministic total adversary D_ω by

$$D_\omega(\sigma) = \begin{cases} (\tau,\nu) & \text{if } \mathit{last}(\sigma) = s, \\ (a,\mu) & \text{if } \mathit{last}(\sigma) = s', \\ \perp & \text{otherwise.} \end{cases}$$

Informally, D_ω keeps on scheduling the τ-transition until the τ-loop is exited. This eventually happens with probability 1.

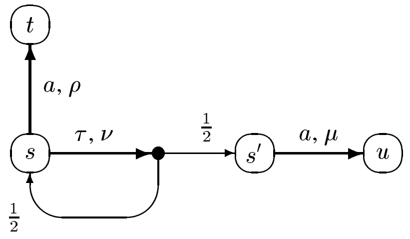


Figure 5.3: Paths in a probabilistic system

5.3.3 Paths in Adversaries

Apart from paths in a system, we can also identify paths in a partial adversary A. Informally, the latter are those paths in a system that are scheduled by A with a positive probability. Thus, these describe the execution sequences which are obtained when the nondeterministic choices are resolved according to A (and the probabilistic choices according to the transitions scheduled). The fullpaths in A are the fullpaths of the system. The maximal paths in A represent "completed" behavior in A and contain the infinite paths and the paths in which A can interrupt the execution (with a positive probability).

Definition 5.3.13 (Paths in partial adversaries) Let A be a partial adversary. A *path in A* is a finite or infinite path

$$\sigma \;=\; s_0 \overset{a_1,\mu_1}{\rightsquigarrow} s_1 \overset{a_2,\mu_2}{\rightsquigarrow} \ldots$$

with $A(\mathit{prefix}(\sigma, i))(\mathit{Step}(\sigma, i+1)) > 0$ for $0 \le i < \mathit{length}(\sigma)$. A *fullpath* in A is a fullpath of the system that is also path in A. The *maximal paths in A* are the infinite paths in A and the finite paths σ in A where $A(\sigma)(\perp) > 0$. $\mathit{Path}^A_*(s)$ denotes the set of all finite paths σ in A where $\mathit{first}(()\sigma) = s$. Similarly, we define $\mathit{Path}^A_{max}(s)$ (resp. $\mathit{Path}^A_{*,max}(s)$ or $\mathit{Path}^A_{ful}(s)$) as the set of maximal paths (resp. finite maximal paths or fullpaths) in A that start in s.

Note that all fullpaths in A are maximal paths in A, but not conversely. If A is partial, then $A(\sigma)(\perp) > 0$ is possible even if $\mathit{last}(\sigma)$ is nonterminal. Thus, maximality of a finite path σ in a partial adversary does not imply that σ is a fullpath.

Notation 5.3.14 (The generated words $\mathit{Words}(s, A)$) Let A be a randomized partial adversary and $s \in S$. We define $\mathit{Words}(s, A) \;=\; \left\{ \mathit{word}(\sigma) : \sigma \in \mathit{Path}^A_{*,max}(s) \right\}$.

Thus, $\mathit{Words}(s, A)$ is the set of all finite words in Act^*_τ which can be generated by a finite maximal path in A that starts in s. These correspond to the action sequences that occur in a finite path (starting in s) which is obtained when the nondeterministic choices are resolved according to A.

Example 5.3.15 Consider the adversaries in Example 5.3.12 again. Their paths are the following.

- The deterministic adversary D contains two paths starting in s: s and $s \overset{a,\rho}{\rightsquigarrow} t$. The latter is a maximal fullpath in D, the former is neither maximal nor a fullpath. The only word in $\mathit{Words}(s, D)$ is a.
- The paths in A starting in s are s, $s \overset{a,\rho}{\rightsquigarrow} t$, $s \overset{\varepsilon,\nu}{\rightsquigarrow} s$, $s \overset{\tau,\nu}{\rightsquigarrow} s'$. Each of these is maximal, but $s \overset{a,\rho}{\rightsquigarrow} t$ is the only fullpath. We have $\mathit{Words}(s, A) = \{\varepsilon, \tau, a\}$.
- The adversary D_ω contains infinitely many paths starting in s: the paths $s \overset{\tau,\nu}{\rightsquigarrow} s \overset{\tau,\nu}{\rightsquigarrow} s \ldots s$ and $s \overset{\tau,\nu}{\rightsquigarrow} s \overset{\tau,\nu}{\rightsquigarrow} s \ldots s \overset{\tau,\nu}{\rightsquigarrow} s'$ and the maximal fullpaths $s \overset{\tau,\nu}{\rightsquigarrow} s \overset{\tau,\nu}{\rightsquigarrow} s \ldots$ and $s \overset{\tau,\nu}{\rightsquigarrow} s \overset{\tau,\nu}{\rightsquigarrow} s \ldots s \overset{\tau,\nu}{\rightsquigarrow} s' \overset{a,\mu}{\rightsquigarrow} u$. As only the latter path is finite and maximal, we have $\mathit{Words}(s, D_\omega) = \tau^* a$.

5.3.4 The Probabilistic Behavior of an Adversary

Each adversary defines a probability space on the set of paths.

Notation 5.3.16 (The transition probabilities $\mathbf{P}^A(\cdot)$) Let A be a partial adversary. Define the function $\mathbf{Q}^A : \mathit{Path}_*^A \to [0, 1]$ inductively by

$$\mathbf{Q}^A(s) = 1 \text{ and } \mathbf{Q}^A(\sigma \overset{a,\mu}{\rightsquigarrow} t) = \mathbf{Q}^A(\sigma) \cdot A(\sigma)(a, \mu) \cdot \mu(t).$$

Moreover, let $\mathbf{P}^A(\sigma) = \mathbf{Q}^A(\sigma) \cdot A(\sigma)(\perp)$. For $s,\ t \in S$, $C \subseteq S$, $W \subseteq \mathit{Act}_\tau^*$ and $\alpha = a_1 a_2 \ldots a_l \in \mathit{Act}_\tau^*$ define

$$\begin{aligned}
\mathbf{P}^A(s, \alpha, t) &= \sum_{\sigma \in \mathit{Path}^A_{\mathit{finmax}}(s,\alpha,t)} \mathbf{P}^A(\sigma), \\
\mathbf{P}^A(s, \alpha, C) &= \sum_{t \in C} \mathbf{P}^A(s, \alpha, t) \\
\mathbf{P}^A(s, t) &= \sum_{\sigma \in \mathit{Path}^A_{\mathit{finmax}}(s,t)} \mathbf{P}^A(\sigma), \\
\mathbf{P}^A(s, C) &= \sum_{t \in C} \mathbf{P}^A(s, t) \\
\mathbf{P}^A(s, W, C) &= \sum_{\alpha \in W} \sum_{t \in C} \mathbf{P}^A(s, \alpha, t)
\end{aligned}$$

where $\mathit{Path}^A_{\mathit{finmax}}(s, \alpha, t) = \{\sigma \in \mathit{Path}^A_{\mathit{finmax}}(s) : \mathit{last}(\sigma) = t, \mathit{word}(\sigma) = \alpha\}$ and $\mathit{Path}^A_{\mathit{finmax}}(s, t) = \{\sigma \in \mathit{Path}^A_{\mathit{finmax}}(s) : \mathit{last}(\sigma) = t\}$.

Note that $\mathbf{Q}^D(s_0 \overset{a_1,\mu_1}{\rightsquigarrow} s_1 \ldots \overset{a_n,\mu_n}{\rightsquigarrow} s_n) = \mu_1(s_1) \cdot \ldots \cdot \mu_l(s_l)$ for deterministic adversaries D and that $\mathbf{P}^A(s, t) = \sum_{\alpha \in \mathit{Act}_\tau^*} \mathbf{P}^A(s, \alpha, t)$ for all adversaries A.

The value $\mathbf{Q}^A(\sigma)$ for a finite path σ is the probability for an execution to start with the prefix σ when the nondeterministic choices are resolved according to A. $\mathbf{P}^A(\sigma)$ stands for the probability for σ to be a maximal finite path in A. The *transition probabilities* $\mathbf{P}^A(s, \alpha, t)$ denote the probability for state s to move to state t via a finite maximal path in A labeled by the word $\alpha \in \mathit{Act}_\tau^*$. $\mathbf{P}^A(s, t)$ denotes the probability to move from s to t in A via any finite maximal path in A.

Example 5.3.17 Consider the adversaries in Example 5.3.12 once more. We compute their transition probabilities.

- For the partial adversary D, we have $\mathbf{P}^D(s) = 0$, $\mathbf{P}^D(s \overset{a,\rho}{\rightsquigarrow} t) = 1$ and $\mathbf{P}^D(s,t) = 1$.
- For A we have $\mathbf{P}^A(s) = \frac{1}{3}$, $\mathbf{P}^A(s \overset{a,\rho}{\rightsquigarrow} t) = \frac{1}{3}$, $\mathbf{P}^A(s \overset{\tau,\nu}{\rightsquigarrow} s) = \frac{1}{6}$, $\mathbf{P}^A(s \overset{\tau,\nu}{\rightsquigarrow} s') = \frac{1}{6}$. Hence $\mathbf{P}^A(s,\tau^*,\{s,s'\}) = \mathbf{P}^A(s) + \mathbf{P}^A(s \overset{\tau,\nu}{\rightsquigarrow} s) + \mathbf{P}^A(s \overset{\tau,\nu}{\rightsquigarrow} s') = \frac{2}{3}$.
- For the paths in D_ω, for $l \in \mathbb{N}$ let σ_l denote the path of the form $s \overset{\tau,\nu}{\rightsquigarrow} s \overset{\tau,\nu}{\rightsquigarrow} s \ldots \overset{\tau,\nu}{\rightsquigarrow} s$ of length l, σ'_{l+1} the path $\sigma_l \overset{\tau,\nu}{\rightsquigarrow} s'$ and σ^a_{l+1} the path $\sigma_l \overset{\tau,\nu}{\rightsquigarrow} s' \overset{a,\mu}{\rightsquigarrow} u$. Then, for $l \leq 1$,

$$\begin{aligned}
&\mathbf{Q}^{\mathcal{D}_\omega}(\sigma'_l) = \mathbf{Q}^{\mathcal{D}_\omega}(\sigma_l) = \mathbf{P}^{\mathcal{D}_\omega}(\sigma^a_l) = \tfrac{1}{2^l},\\
&\mathbf{P}^{D_\omega}(s,\tau^*,\{s,s'\}) = 0,\\
&\mathbf{P}^{D_\omega}(s,s) = \mathbf{P}^{D_\omega}(s,s') = 0,\\
&\mathbf{P}^{D_\omega}(s,\tau^* a,u) = \mathbf{P}^{D_\omega}(s,u) = 1.
\end{aligned}$$

As in Chapter 2, we can associate a probability space to each adversary.

Notation 5.3.18 (The probability measure $Prob^A$) Let $A \in RAdv_\perp$ and $s \in S$. If $\sigma \in Path^{*A}(s)$ then the *basic cone* of σ in A is given by

$$\sigma{\uparrow}^A = \{\pi \in Path^A_{max}(s) : \sigma \sqsubseteq \pi\}.$$

Let $\mathcal{F}^A(s)$ denote the smallest sigma-field on $Path^A_{max}(s)$ that contains the basic cones $\sigma{\uparrow}^A$ and let $Prob^A$ denote the unique probability measure on $\mathcal{F}^A(s)$, where the probability measure of the basic cones is

$$Prob^A\left(\sigma{\uparrow}^A\right) = \mathbf{Q}^A(\sigma).$$

Then $(Path^A_{max}(s), \mathcal{F}_A(s), \mathbf{Q}^A)$ is a probability space for each $s \in S$.

5.3.5 Finitary and Almost Finitary Adversaries

Definition 5.3.19 ((Almost) finitary randomized partial adversaries) A randomized partial adversary A is called *finitary* iff all paths in A are finite, i.e. if $Path^A(s) = Path^A_{fin}(s)$ for all states s. We call A *almost finitary* iff almost all paths in A are finite, i.e. if $\mathbf{P}^A(s,S) = 1$.[6]. We denote the sets of finitary and almost finitary randomized adversaries by $RAdv^{fin}_\perp$ and $RAdv^\omega_\perp$ respectively.

Obviously, any finitary partial adversary is almost finitary. Finitary partial adversaries are those that do not have infinite fullpaths. In other words, the tree associated with them is finite. They describe the probabilistic effect of the combination of *finitely many* transitions. Almost finitary partial adversaries are those where the probability measure of the infinite fullpaths in them is 0 (but infinite fullpaths in them might exist).

Remark 5.3.20 For each almost finitary randomized partial adversary A and state s, the function $t \mapsto \mathbf{P}^A(s,t)$ is a probability distribution on S. Similarly, $(\alpha,t) \mapsto \mathbf{P}^A(s,\alpha,t)$ is a probability distribution on $Act^*_\tau \times S$.

[6]Note that it is sufficient that all (almost all, resp.) *maximal* paths in A are finite.

Example 5.3.21 First, we give an informal example to illustrate the difference between finitary and almost finitary adversaries. Flipping a coin three times in a row corresponds to a finitary adversary, because all maximal paths have length 3. Flipping a coin until heads come up corresponds to an almost finitary adversary, because the probability on termination (i.e. on a finite path) is one. It is not finitary because the infinite sequence in which only heads come up is a maximal path in the corresponding adversary.

For a more formal example, based on the same ideas, reconsider the system of Figure 5.3. Define the partial deterministic adversaries D_l $(l \in \mathbb{N})$ by

$$D_l(\sigma) = \begin{cases} (\tau, \nu) & \text{if } \mathit{last}(\sigma) = s \wedge \mathit{length}(\sigma) \leq l, \\ (a, \mu) & \text{if } \mathit{last}(\sigma) = s', \\ \perp & \text{otherwise.} \end{cases}$$

Note that D_l is not memoryless. For each l, the adversary D_l is finitary and interrupts the execution after the τ loop has been taken l times or more. This yields the transition probability

$$\mathbf{P}^{D_l}(s,s) = \left(\tfrac{1}{2}\right)^l, \mathbf{P}^{D_l}(s,u) = 1 - \left(\tfrac{1}{2}\right)^l, \mathbf{P}^{D_l}(s,s') = \mathbf{P}^{D_l}(s,t) = 0.$$

The adversary D_ω from Example 5.3.12 is almost finitary but not finitary as it yields the infinite path $s \overset{\tau}{\leadsto} s \overset{\tau}{\leadsto} s \overset{\tau}{\leadsto} \ldots$. As we have seen, the transition probabilities are

$$\mathbf{P}^{D_\omega}(s,u) = 1, \quad \mathbf{P}^{D_\omega}(s,s) = \mathbf{P}^{D_\omega}(s,s') = 0.$$

These cannot be obtained by any finitary partial adversary.

5.3.6 Properties of Adversaries

The following proposition states that each partial adversary A can be written as a pointwise limit of finitary adversaries A_l. We believe that, by defining a suitable metric on adversaries, one can also write A as a uniform limit in this metric.

Proposition 5.3.22 *Let A be partial randomized (deterministic) adversary.*

1. *Then there are finitary randomized (deterministic) partial adversaries A_l such that $A = \lim_{l\to\infty} A_l$, where* lim *denotes the pointwise limit of the functions A_l.*
2. *If $A = \lim_{l\to\infty} A_l$ is a pointwise limit, then $\mathbf{Q}^A(\sigma) = \lim_{l\to\infty} \mathbf{Q}^{A_l}(\sigma)$.*

PROOF:

1. Take $A_l(\sigma) = A(\sigma)$ if $\mathit{length}(\sigma) \leq l$ and $A_l(\sigma)(\perp) = 1$ otherwise.
2. Easy verification.

□

The following proposition shows that each finite randomized partial adversary can be written as a convex combination of deterministic partial adversaries. The proof given below crucially depends on the fact that the systems we consider are finite. The proof immediately carries over to finitely branching systems, but not to systems with an infinite branching structure.

Notation 5.3.23 (The set $\mathcal{D}(s, A)$) Let A be a finite partial randomized adversary and let $s \in S$. $\mathcal{D}(s, A)$ denotes the set of deterministic partial adversaries D where $Path^D_{max}(s) \subseteq Path^A_{max}(s)$.[7]

Proposition 5.3.24 *Let $A \in RAdv^{fin}_{\perp}$ and $s \in S$.*

1. *Then, there is a finite family $(D_i)_{i \in I}$ of deterministic finitary partial adversaries D_i and a family $(p_i)_{i \in I}$ of real numbers $p_i \in [0,1]$ such that $\sum_{i \in I} p_i = 1$ and*

$$A(\sigma)(a, \mu) \;=\; \sum_{i \in I} p_i \cdot D_i(\sigma)(a, \mu), \quad \textit{for all } \sigma \in Path^A(s),\ a,\ \mu.$$

2. *If $A \;=\; \sum_{i \in I} p_i \cdot A_i$, for adversaries $(A_i)_{i \in I}$, real numbers $(p_i)_{i \in I}$, $p_i \in [0,1]$ with $\sum_{i \in I} p_i = 1$ then*

$$\mathbf{Q}^A(\sigma) \;=\; \sum_{i \in I} p_i \cdot \mathbf{Q}^{A_i}(\sigma).$$

PROOF:

1. Let A be a finitary randomized partial adversary and $s \in S$. Since A is finitary, the set $\mathcal{D}(s, A)$ is finite and all $D \in \mathcal{D}(s, A)$ are finitary.

   The idea in the proof is as follows. Each D in $\mathcal{D}(s, A)$ can be seen as an adversary within A because, among the steps that A schedules with a positive probability, D schedules one with probability one. If we multiply all the probabilities that A assigns to steps taken in D (yielding a value $p(D, A)$), then A is obtained by selecting the adversary D with probability $p(D, A)$, as we show below.

   Let

$$p(D, A) = \prod_{\sigma \in Path^A(s)} A(\sigma)(D(\sigma)),$$

   where, as before, we write $D(\sigma) = (a, \mu)$ for $D(\sigma)(a, \mu) = 1$. Then

$$\sum_{D \in \mathcal{D}(s,A)} p(D, A) \;=\; \sum_{D \in \mathcal{D}(s,A)} \prod_{\sigma \in Path^A(s)} A(\sigma)(D(\sigma)) \;=\; 1,$$

   The last step holds because $\mathcal{D}(s, A)$ is finite and $\sum_x A(\sigma)(x) \;=\; 1$ for all σ. For the same reason we have for all ρ, a, μ that

$$\begin{aligned}
&\sum_{D \in \mathcal{D}(s,A)} p(D, A) \cdot D(\rho)(a, \mu) \;= \\
&\sum_{D \in \mathcal{D}(s,A)} \prod_{\sigma \in Path^A(s)} A(\sigma)(D(\sigma)) \cdot D(\rho)(a, \mu) \;= \\
&\sum_{D \in \mathcal{D}(s,A), D(\rho) = (a,\mu)} \prod_{\sigma \in Path^A(s)} A(\rho)(a, \mu) \cdot 1 \;= \\
&A(\rho)(a, \mu).
\end{aligned}$$

[7]Thus, $D \in \mathcal{D}(s, A)$ iff, for all $\sigma \in Path^D_{fin}(s)$, $D(\sigma) = x$ implies $A(\sigma)(x) > 0$.

2. By induction on the length of σ.

   - If $\sigma = s$ has length 0, then the statement follows immediately.
   - Let $\sigma = \sigma' \overset{a,\mu}{\leadsto} t$, then

$$\begin{aligned}\mathbf{Q}^A(\sigma' \overset{a,\mu}{\leadsto} t) &= \mathbf{Q}^A(\sigma') \cdot A(\sigma')(a,\mu) \cdot \mu(t) = \\ &= \sum_{i \in I} p_i \cdot \mathbf{Q}^{A_i}(\sigma') \cdot \sum_{j \in I} p_j \cdot A_j(\sigma')(a,\mu) \cdot \mu(t) \ = \\ &= \sum_{i \in I} p_i \cdot \mathbf{Q}^{A_i}(\sigma') \cdot A_i(\sigma')(a,\mu) \cdot \mu(t) \ = \\ &= \sum_{i \in I} p_i \cdot \mathbf{Q}^{A_i}(\sigma' \overset{a,\mu}{\leadsto} t).\end{aligned}$$

□

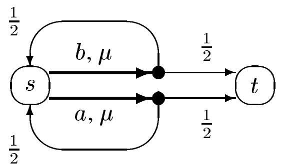


Figure 5.4: Almost finitary adversaries need not be a convex combination of deterministic ones.

The following example shows that the result of Proposition 5.3.24 cannot be generalized immediately to almost finitary partial adversaries. This should not be too much of a surprise because, even for finite systems, there are uncountably many maximal paths in an almost finitary adversary, whereas there are only countably many paths in a finitary adversary. Hence, there are countably many maximal paths in a convex combination of finitary adversaries. In fact, the natural generalization of Proposition 5.3.24 to almost finitary adversaries would be to define a measure space over the set of adversaries and to express an arbitrary almost finitary adversary as an integral over deterministic ones. The investigation whether it is indeed possible to generalize Proposition 5.3.24 along these line is a topic for future research.

Example 5.3.25 Consider the system in Figure 5.4 and define a randomized partial adversary A which in state s schedules the a and the b–transition each with probability $\frac{1}{2}$, i.e. if $last(\sigma) = s$ then $A(\sigma)(a,\mu) = A(\sigma)(b,\mu) = \frac{1}{2}$ and $A(\sigma)(x) = \perp$ otherwise.

It is not difficult to see that A is almost finitary: the probability of ending up in t after exactly k steps is $\frac{1}{2^k}$.

Every deterministic partial adversary D of this system can be identified uniquely by its longest maximal path σ, which is the longest maximal path in D. We write D_σ for the deterministic partial adversary whose longest maximal path is σ. Note that, $\mathbf{Q}^{D_\rho}(\sigma) > 0$ iff $s \neq \sigma \sqsubseteq \rho$ and in that case $\mathbf{Q}^{D_\rho}(\sigma) = \frac{1}{2^{length(\sigma)}}$.

Now assume that A is a convex combination of deterministic partial adversaries as in Proposition 5.3.24 [8]. This means that there is a countable set C of (in)finite maximal path and a family $\{p_\rho\}_{\rho\in C}$ of real numbers in $[0,1]$ such that $\sum_{\rho\in C} p_\rho = 1$ and

$$A(\sigma)(a,\mu) \;=\; \sum_{\rho\in C} p_\rho \cdot D_\rho(\sigma)(a,\mu) \quad \text{for all } \sigma \in Act^*_\tau \text{ and } t \in S.$$

Let σ be a maximal path of length $n > 0$. Then $\mathbf{Q}^A(\sigma) \;=\; \frac{2}{4^n}$ and hence

$$\sum_{\rho\in C} p_\rho \cdot \mathbf{Q}^{D_\rho}(\sigma) = \sum_{\rho\in C,\sigma\sqsubseteq\rho} p_\rho \cdot \mathbf{Q}^{D_\rho}(\sigma) = \sum_{\rho\in C,\sigma\sqsubseteq\rho} p_\rho \cdot \frac{1}{2^n} = \frac{1}{4^n}.$$

Therefore, we have for all $n > 0$ and all σ of length n, $\sum_{\rho\in C,\sigma\sqsubseteq\rho} p_\rho = \frac{1}{2^n}$.

$$\sum_{\rho\in C, length(\rho)\geq n} p_\rho = \sum_{\sigma\in\{a,b\}^n} \sum_{\sigma\sqsubseteq\rho} p_\rho = \sum_{\sigma\in\{a,b\}^n} \frac{1}{2^n} = 2^n \cdot \frac{1}{2^n} = 1 = \sum_{\rho\in C} p_\rho.$$

This means that $p_\rho = 0$ for all $\rho \in C$ with length $< n$ and, since this holds for all $n > 0$, we have $p_\rho = 0$ for all finite ρ. Now we claim that $p_\sigma = 0$ for infinite sequences σ. Let $\sigma \in \{a,b\}^\infty$ and consider $\sigma = \mathit{Prefix}(\sigma, n)$. Then for all $n > 0$

$$\frac{1}{2^n} = \sum_{\rho\in C,\sigma\sqsubseteq\rho} p_\rho \cdot \frac{1}{2^n} > p_\sigma.$$

Thus, $p_\sigma = 0$ and A is not the convex combination of countably many deterministic partial adversaries.

5.4 Strong and Weak (Bi–)simulation

This section recalls several notions of simulation and bisimulation for probabilistic systems that appear in the literature. More specifically, we treat strong, combined strong and weak (bi–)simulation. These relations have been introduced by Larsen & Skou [LS91], Hansson & Jonsson [HJ94] and Segala & Lynch [SL95] and they are the natural extensions of the corresponding notions for nonprobabilistic labeled transition systems [Mil80, Par81, Mil89, GW89]. Although few of the results in this section are new, we believe that we have contributed by simplifying existing definitions.

Both for nonprobabilistic and for probabilistic systems, simulations and bisimulations are relations on the state space of the system that compare the stepwise behavior of the states. *Bisimulation relations* are equivalences on the state space such that related states exhibit the same stepwise behavior. *Simulation relations* are preorders on the state space (i.e. transitive and reflexive) such that if a state s is related to a state t, then t can mimic all behavior of s. The converse is however not required: t may perform steps that cannot be exhibited by s.

Different notions of (bi–)simulation exist, depending on what is meant by "exhibiting the same stepwise behavior." The basic scheme for (bi–)simulation in the nonprobabilistic case is as follows. A relation R is a simulation iff

[8] In this proposition we only needed the convex combination of finitely many partial adversaries. In general a convex combination can be an infinite sum.

$$\text{If } sRs' \wedge s \xrightarrow{a} t \text{ then there is a state } t' \text{ such that } s' \overset{a}{\dashrightarrow} t' \wedge tRt'.$$

A bisimulation is a simulation relation that is an equivalence. Here, $s \overset{a}{\dashrightarrow} t$ refers to some kind of step from s to t whose visible behavior is a or, if the relation abstracts from internal steps, $\widehat{a}$. The different types of (bi–)simulations are obtained by different choices for $\dashrightarrow$. If we have $s \xrightarrow{a} t$ and $s' \overset{a}{\dashrightarrow} t'$ with sRs' and tRt', then we say that the transition $s \xrightarrow{a} t$ is *matched*, *simulated* or *mimicked* by the step $s' \overset{a}{\dashrightarrow} t'$.

The situation in the probabilistic case is similar. However, we cannot use the relation R to compare the target of two transitions, because these are now probability distributions instead of states. Therefore, we lift R to the set of probability distributions, yielding the relation $\sqsubseteq_R$. In the sequel, we explain why $\sqsubseteq_R$ is a reasonable choice for comparing the target distributions. Thus, the scheme becomes:

$$\text{If } sRs' \wedge s \xrightarrow{a} \mu \text{ then there is a distribution } \mu' \text{ such that } s' \overset{a}{\dashrightarrow} \mu' \wedge \mu \sqsubseteq_R \mu'.$$

Again, we get the different types of (bi–)simulations by taking different instances of $\dashrightarrow$. The ideas behind the variants considered in this chapter are the following.

- *Strong (bi–)simulation* does not abstract from internal transitions. That is, it treats a τ–transition as any other transition. It is obtained by taking the transition relation $\rightarrow$ for $\dashrightarrow$. In other words, strong (bi–)simulation requires each transition to be matched with a single step.

- *Strong combined (bi–)simulation* does not abstract from internal transitions either. But unlike strong simulation relations, combined strong bisimulations allow for a single step to be mimicked by a convex combination of steps. This is formalized by the combined transition relation $\rightarrowtail$.

- *Weak (bi–)simulation* does abstract from internal computation. This means that one a–transition can be mimicked by a combination of τ transitions and one a–transition. This is formalized by the weak transition relation $\Longrightarrow$, whose definition depends on schedulers. These tell which τ and a–transitions are taken. The various types of schedulers (randomized, deterministic, finitary, ω) give rise to different types of weak transitions and, hence, different types of weak (bi–)simulation (notably randomized, deterministic, finitary, ω (bi–)simulation).

In Section 5.5.2 the notion of delay bisimulation is introduced.

- *Delay (bi–)simulation* also abstracts from internal computation, but in a more limited way than weak (bi–)simulation. What we gain is a polynomial algorithm. Delay (bi–)simulation also depends on schedulers and, here too, different kinds of schedulers give rise to different notions of delay (bi–)simulation.

Another well–known relation from the literature is branching bisimulation. This relation is strictly finer than weak bisimulation and takes into account the branching structure of a system. We do not treat it here, but it can be introduced along the exactly same lines as weak bisimulation. The only difference is that the adversaries have to take into account the branching structure. Different types of schedulers give rise to different branching bisimulations. Moreover, one can introduce delay variants of branching bisimulation. We conjecture that these can be decided in polynomial time and space by similar algorithms as the ones introduced in Section 5.7.4.

5.4.1 Strong (Bi–)simulation

Strong (bi–)simulation, introduced by [LS91], requires that each step leaving from a state s can be matched by step from a related state.

Definition 5.4.1 (Strong (bi–)simulation) A relation $R \subseteq S \times S$ is a *strong simulation* iff for all $(s, s') \in R$:

$$\text{If } s \xrightarrow{a} \mu \text{ then there is a transition } s' \xrightarrow{a} \mu' \text{ with } \mu \sqsubseteq_R \mu'.$$

A *strong bisimulation* is a strong simulation which is an equivalence. We say that the state s_2 *strongly simulates* s_1 and write $s_1 \preceq_{ssim} s_2$ iff there exists a strong simulation R such that $(s_1, s_2) \in R$. If there exists a bisimulation containing (s_1, s_2), then s_1 and s_2 are called *strongly bisimilar*, denoted by $s_1 \approx_{sbis} s_2$.

The definition of strong bisimulation can be understood as follows. First, recall that $\equiv_R$ and $\sqsubseteq_R$ coincide for equivalences (see part (a) of Proposition 5.2.1). Therefore, we may replace $\sqsubseteq_R$ with $\equiv_R$ in the definition of a strong *bi*simulation. As bisimilar states are interchangeable, it does not matter which state within the same bisimulation equivalence class is reached. Therefore, we are interested in the probability to reach an equivalence class, rather than in the probability to reach an individual state. Thus, when we match two transitions in a bisimulation relation, we require that they reach each equivalence class with the same probability. This is exactly the requirement $\mu \equiv_R \mu'$. Before explaining the simulation relations, we give an example of a strong bisimulation relation.

Example 5.4.2 The states s and s' in the system in Figure 5.5 are strongly bisimilar and so are y and y' and so are x_1, x_2 and x'. Indeed μ and μ' assign exactly the same probabilities to each of the equivalence classes $\{s, s'\}$, $\{z\}$, $\{y, y'\}$ and $\{x_1, x_2, x\}$.

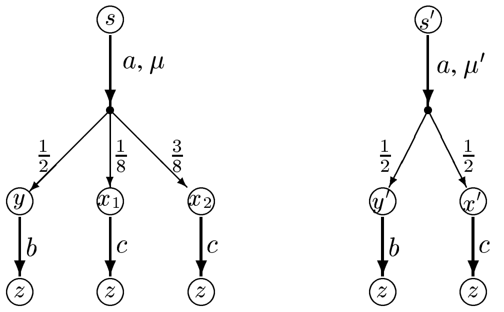


Figure 5.5: $s \approx_{sbis} s'$

The situation for strong simulation is similar, except that now we cannot compare the target probability distributions of two transitions by comparing the probabilities they assign to the equivalence classes. Instead, $\mu \sqsubseteq_R \mu'$ is established by a weight function. This function quantifies to which extend the probability $\mu(x)$ contributes to the probability $\mu'(x')$ of a related state.

Example 5.4.3 Now, consider the system in Figure 5.6. The relation $R = \{(s, s'), (t, t'), (u, u'), (u, t'), (v, v')\}$ is a strong simulation. For instance, the state u is related to the states t' and u'. We can distribute $\mu(u) = \frac{2}{3}$ over t' and u' by having $wgt(t, t') = 1/3$, $wgt(u, u') = 1/2$. Moreover, take $wgt(u, t') = 1/6$ and $wgt(\cdot, \cdot) = 0$ in the other cases. This shows $\mu \sqsubseteq_R \mu'$.

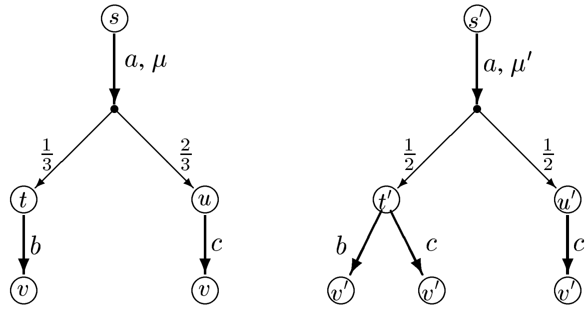


Figure 5.6: $s \preceq_{ssim} s'$

We refer the reader to Section 5.6 for some basic properties of strong bisimulation relations.

5.4.2 Strong Combined (Bi–)simulation

In [Seg95b], Segala argues that strong (bi–)simulation is too strong in certain cases. He argues that it is more natural to allow an a–transition to be matched by a convex combination of a–transitions, rather than by a single a–transition. This leads to a notion which we call *combined bisimulation*.

Definition 5.4.4 We write $s \overset{a}{\rightarrowtail} \mu$ iff there is a countable family of transitions $s \xrightarrow{a} \mu_i$, such that μ is a convex combination of the distributions μ_i.

Definition 5.4.5 (Strong combined (bi–)simulation) A *strong combined simulation* is a relation R on S such that for all $(s, s') \in R$:

If $s \xrightarrow{a} \mu$ then there is a transition $s' \overset{a}{\rightarrowtail} \mu'$ with $\mu \sqsubseteq_R \mu'$.

A *strong combined bisimulation* is a strong combined simulation which is an equivalence. We write $s_1 \preceq_{scsim} s_2$ ($s_1 \approx_{scbis} s_2$) iff there exists a strong combined (bi–)simulation which contains (s_1, s_2).

Simulation relations are often used to establish that one state (or system) correctly implements another one. The example below shows that, unlike strong simulations, strong *combined* simulations allow a nondeterministic choice to be implemented by a probabilistic choice.

Example 5.4.6 Consider the systems in Figure 5.7. The relation $\{(s_1, t_1), (s_2, t_2), (s_3, t_3)\}$ is a strong combined simulation, but not a strong simulation. The transition $t_1 \overset{a}{\rightarrowtail} \nu$ with $\nu(t_2) = \nu(t_3) = \frac{1}{2}$ is obtained as a convex combination of the steps $t_1 \xrightarrow{a} \{t_2 \mapsto 1\}$ and $t_1 \xrightarrow{a} \{t_3 \mapsto 1\}$ with $\nu = \frac{1}{2} \cdot \{t_2 \mapsto 1\} + \frac{1}{2} \cdot \{t_3 \mapsto 1\}$.

Similar systems show the difference between strong bisimulation and strong combined bisimulation: in Figure 5.7, we have $t_1 \approx_{scbis} u_1$ but $t_1 \not\approx_{sbis} u_1$.

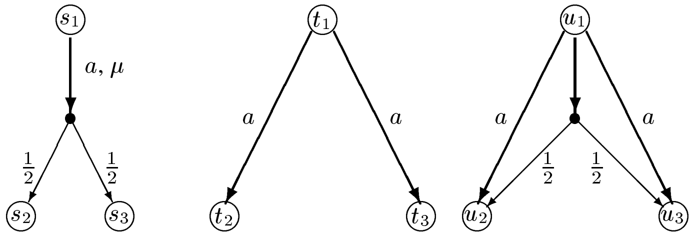


Figure 5.7: Simulating a probabilistic choice by a nondeterministic one.

5.4.3 Weak (Bi–)simulation

Unlike strong (bi–)simulation and strong combined (bi–)simulation, weak (bi–)simulation abstracts from internal computation. This means that τ–transitions, being invisible for the environment, may always be used to mimic another step. More precisely, a τ–transition leaving from a state s may be mimicked by any number (zero or more) of τ transitions in a related state. An a–transition in s may be matched by first taking some (zero or more) τ transitions, then an a–transition and some τ–transitions again.

In the nonprobabilistic setting, a weak simulation relation R is required to satisfy for all s, s' such that sRs'

$$\text{if } s \xrightarrow{a} u \text{ then there is some } u' \text{ with } s' \overset{a}{\Longrightarrow} u' \text{ and } s'Ru'.$$

A bisimulation is a simulation that is an equivalence [Mil89]. Here, the double arrow relation $\Longrightarrow \; \subseteq \; S \times Act_\varepsilon \times S$ is defined via the transitive reflexive closure of the internal steps. Thus, $s \overset{\varepsilon}{\Longrightarrow} u$ iff $s \xrightarrow{\tau} \ldots \xrightarrow{\tau} u$ and $s \overset{a}{\Longrightarrow} u$ iff $s \xrightarrow{\tau} \ldots \xrightarrow{\tau} v \xrightarrow{a} t \xrightarrow{\tau} \ldots \xrightarrow{\tau} u$ for any visible action a. Note that $s \overset{\widehat{a}}{\Longrightarrow} t$ iff there is an execution starting at s, leading to t, whose word is an element of $\tau^* \widehat{a} \tau^*$.

For probabilistic systems, the situation is similar, but more complex. The definitions of weak (bi-)simulations are adapted for the probabilistic setting using suitable modifications of the relation $\Longrightarrow$ [SL95, Seg95b]. A weak transition $s \overset{\widehat{a}}{\Longrightarrow} \mu$ corresponds to an execution over the words $\tau^* \widehat{a} \tau^*$. Since executions for probabilistic systems are formalized as adversaries, $s \overset{\widehat{a}}{\Longrightarrow} \mu$ corresponds to an adversary A with $Words(s, A) \subseteq \tau^* \widehat{a} \tau^*$ such that μ equals the associated distribution $\mathbf{P}^A(s, \cdot)$. The use of randomized versus deterministic and almost finitary versus finitary adversaries yields four different notions of weak transitions, and hence, four notions of weak (bi–)simulation. However, we will see that the use of deterministic adversaries induces a notion of (bi–)simulation with completely different properties than the other (bi–)simulations. For instance, the corresponding (bi–)simulations are not transitive.

Notation 5.4.7 [The partial adversary classes $\mathcal{C}[s, W]$] Let $\mathcal{C}$ be a class of partial adversaries, let $s \in S$ and $W \subseteq Act_\tau^\infty$ be a set of finite and infinite words. We define $\mathcal{C}[s, W] = \{A \in \mathcal{C} : Words(s, A) \subseteq W\}$.

In the sequel, we consider adversaries over the set words $\tau^* a \tau^*$ and, in particular, we consider the sets

- $RAdv_\perp^{fin}[s, \tau^* \widehat{a} \tau^*]$ consisting of finitary, randomized, partial adversaries starting in s over the words $\tau^* \widehat{a} \tau^*$,

- $DAdv_{\perp}^{fin}[s, \tau^*\widehat{a}\tau^*]$ consisting of finitary, deterministic, partial adversaries starting in s over the words $\tau^*\widehat{a}\tau^*$,
- $RAdv_{\perp}^{\omega}[s, \tau^*\widehat{a}\tau^*]$ consisting of almost finitary, randomized, partial adversaries starting in s over the words $\tau^*\widehat{a}\tau^*$,
- $DAdv_{\perp}^{\omega}[s, \tau^*\widehat{a}\tau^*]$ consisting of almost finitary, deterministic, partial adversaries starting in s over the words $\tau^*\widehat{a}\tau^*$.

For a visible action a, the adversaries in $RAdv_{\perp}^{fin}[s, \tau^*\widehat{a}\tau^*]$ and $DAdv_{\perp}^{fin}[s, \tau^*\widehat{a}\tau^*]$ correspond to trees in which each path is finite and in which each path to a leaf contains exactly one a–transition and zero or more τ–transitions. The adversaries in $RAdv_{\perp}^{\omega}[s, \tau^*\widehat{a}\tau^*]$ and $DAdv_{\perp}^{\omega}[s, \tau^*\widehat{a}\tau^*]$ are trees that might have infinite paths, but the total probability measure on these infinite paths is zero. Each finite path to a leaf contains exactly one a–transition and zero or more τ–transitions. The infinite paths contain at most one a–transition and infinitely many τ–transitions. The latter is a consequence of an adversary being almost finitary, it does not hold for arbitrary adversaries over the words $\tau^*\widehat{a}\tau^*$.

Recall that for an adversary A, the probability to reach a state t from a state s via a maximal path in A is given by $\mathbf{P}^A(s,t)$. Thus, $\mathbf{P}^A(s,t)$ is the probability to end up in t in the execution generated by A, which is exactly what we need when defining the weak transition relation $\Longrightarrow$ for probabilistic systems.

Each class $\mathcal{C}$ of adversaries yields a notion of weak transition $\Longrightarrow^{\mathcal{C}}$ defined by

$$s \overset{\widehat{a}}{\Longrightarrow}{}^{\mathcal{C}} \mu \text{ iff there exists } A \in \mathcal{C}[s, \tau^*\widehat{a}\tau^*] \text{ with } \mu = \mathbf{P}^A(s, \cdot).$$

Thus, the classes defined above yield four different types of weak transitions.

Definition 5.4.8 Let $s \in S$, $a \in Act_\tau$ and $\mu \in Distr(S)$. Define

$$s \overset{\widehat{a}}{\Longrightarrow}{}^{d,fin} \mu \text{ iff there exists } A \in DAdv_{\perp}^{fin}[s, \tau^*\widehat{a}\tau^*] \text{ with } \mu = \mathbf{P}^A(s, \cdot).$$

If A is an element of $DAdv_{\perp}^{fin}[s, \tau^*\widehat{a}\tau^*]$ with $\mu = \mathbf{P}^A(s, \cdot)$, we say that A *establishes* the transition $s \overset{\widehat{a}}{\Longrightarrow}{}^{d,fin} \mu$. The weak relations $\overset{\widehat{a}}{\Longrightarrow}{}^{r,fin}$, $\overset{\widehat{a}}{\Longrightarrow}{}^{\omega,d}$ and $\overset{\widehat{a}}{\Longrightarrow}{}^{\omega,r}$ are defined by taking the corresponding classes of adversaries.

It is clear that $s \overset{\widehat{a}}{\Longrightarrow}{}^{fin,d} \mu$ implies both $s \overset{\widehat{a}}{\Longrightarrow}{}^{fin,r} \mu$ and $s \overset{\widehat{a}}{\Longrightarrow}{}^{\omega,d} \mu$. The latter two are incomparable and each imply $s \overset{\widehat{a}}{\Longrightarrow}{}^{\omega,r} \mu$. The following example shows that every weak transition extends the relation $\rightarrow$ and that the randomized variants extend $\rightarrowtail$. Moreover, the example shows that, even though the systems we consider are finite, the number of finitary, weak transitions can be infinite.

Example 5.4.9
- The deterministic adversary $D_\perp$ with $D_\perp(\sigma) = \perp$ for all σ yields the trivial weak transition $s \overset{\varepsilon}{\Longrightarrow} s$.
- If $s \xrightarrow{a} \mu$ then $s \overset{\widehat{a}}{\Longrightarrow}{}^{fin,d} \mu$. This is established by the finitary, deterministic adversary D with $D(s) = (a, \mu)$ and $D(\sigma) = \perp$ if $\sigma \neq s$. (cf. the adversary D in Example 5.3.12 on page 94.)
- Assume $s \overset{a}{\rightarrowtail} \mu$. Let $s \xrightarrow{a} \mu_i$ be transitions such that $\sum_i p_i \cdot \mu_i = \mu$. Then the finitary, randomized adversary A with $A(s)(a, \mu_i) = p_i$ and $A(\sigma)(\perp) = 1$ in all other cases establishes that $s \overset{\widehat{a}}{\Longrightarrow}{}^{fin,r} \mu$.

- Now, consider the system in Figure 5.3 on page 95. The deterministic, almost finitary adversary D_ω (Example 5.3.12, page 94) establishes $s \stackrel{a}{\Longrightarrow}^{\omega,d} u$. As there is no finitary adversary A with $\mathbf{P}^A(s,\cdot) = \{u \mapsto 1\}$, we do not have $s \stackrel{a}{\Longrightarrow}^{\mathit{fin},d} u$.
- Let D' be the convex combination of the adversaries D and D_ω (both from Example 5.3.12 again) given by $D'(\sigma)(x) = \frac{1}{2}D_\omega(\sigma)(x) + \frac{1}{2}D(\sigma)(x)$. Then D' shows $s \stackrel{a}{\Longrightarrow}^{\omega,r} \rho'$, where $\rho'(t) = \frac{2}{3}$ and $\rho'(u) = \frac{1}{3}$. We clearly do not have $s \stackrel{a}{\Longrightarrow}^{\omega,d} \rho'$.
- The deterministic, finitary adversary D'_l given by

$$D'_l(\sigma) = \begin{cases} (\tau, \nu) & \text{if } \mathit{last}(\sigma) = s \wedge \mathit{length}(\sigma) \leq l, \\ (a, \rho) & \text{if } \mathit{last}(\sigma) = s \wedge \mathit{length}(\sigma) = l + 1, \\ (a, \mu) & \text{if } \mathit{last}(\sigma) = s', \\ \perp & \text{otherwise.} \end{cases}$$

  Thus, D'_l tries exiting the τ–loop l times and if it has not succeeded after l steps, it takes the a–transition leading to t. Then D'_l shows that $s \stackrel{a}{\Longrightarrow}^{d,\mathit{fin}} \mu_l$, where $\mu_l(t) = \frac{1}{2^{l+1}}$ and $\mu_l(u) = 1 - \frac{1}{2^{l+1}}$.
- For the simple communication system of Example 5.3.5 on page 92, we have that $s_{del} \stackrel{cons}{\Longrightarrow}^{\omega,d} s_{init}$ and $s_{del} \stackrel{prod}{\Longrightarrow}^{\omega,d} s_{ack}$.

Now, we can define the notions of (bi-)simulation induced by the weak transition relations. The ω-variants of randomized, weak (bi-)simulations coincide with those introduced by Segala & Lynch [SL95]. The deterministic and the finitary variants have not been defined in literature. We will see in Section 5.6 that deterministic variants lead to rather questionable relations.

Definition 5.4.10 ((bi-)simulation, cf. [SL95]) A relation $R \subseteq S \times S$ is called a *finitary, randomized, weak simulation* iff for all $(s, s') \in R$:

$$s \stackrel{a}{\to} \mu \text{ implies } s' \stackrel{\widehat{a}}{\Longrightarrow}^{r,\mathit{fin}} \mu' \text{ for some } \mu' \text{ with } \mu \sqsubseteq_R \mu'.$$

A *finitary, randomized, weak bisimulation* is a finitary, randomized, weak simulation R that is an equivalence. The finitary, randomized, weak simulation preorder $\preceq^{r,\mathit{fin}}_{wsim}$ is defined by $s \preceq^{r,\mathit{fin}}_{wsim} s'$ iff $(s, s') \in R$ for some finitary, randomized, weak simulation R. We define finitary, randomized, weak bisimulation equivalence by $s \approx^{r,\mathit{fin}}_{wbis} s'$ iff there exists a finitary, randomized, weak bisimulation R with $(s, s') \in R$. In a similar way, we define the finitary variants of the deterministic, weak (bi–)simulation relations $\preceq^{d,\mathit{fin}}_{wsim}$, and $\approx^{d,\mathit{fin}}_{wbis}$. The randomized and deterministic variants of ω-(bi-)simulation are denoted by $\preceq^{r,\omega}_{wsim}$, $\approx^{r,\omega}_{wsim}$ and $\preceq^{d,\omega}_{wsim}$, $\approx^{d,\omega}_{wsim}$ respectively.

Recall that the relations $\equiv_R$ and $\sqsubseteq_R$ coincide for equivalences and that we may therefore replace $\sqsubseteq_R$ with $\equiv_R$ when dealing with bisimulations.

Example 5.4.11 Reconsider the system in Figure 5.3 on page 95. Let R be the smallest equivalence relation R identifying s and s' and identifying t and u. Then R is a weak, deterministic, finitary bisimulation. The step $s \stackrel{\tau}{\to} \nu$ in s is matched by the step $s' \stackrel{\tau}{\Longrightarrow}^{\mathit{fin},d} s'$ in s', since we have $\nu \equiv_R \{s' \mapsto 1\}$. The step $s \stackrel{a}{\to} \rho$ is matched by $s' \stackrel{a}{\to} \mu$ and vice versa, since $\mu \equiv_R \rho$. Hence $s \approx^{d,\mathit{fin}} s'$.

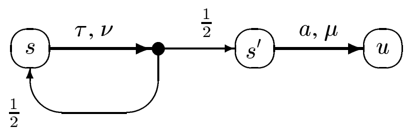


Figure 5.8: $s \approx^{d,\omega} s'$, but $s \not\approx^{d,fin} s'$

Example 5.4.12 Now consider the system in Fig 5.8. Let R be the smallest equivalence relation identifying s and s'. Then R is a deterministic, almost finitary, weak bisimulation. The step $s \xrightarrow{\tau} \nu$ is matched by $s' \stackrel{\tau}{\Longrightarrow}^{\omega,d} s'$ as before and the step $s' \xrightarrow{a} \mu$ is matched by $s \stackrel{a}{\Longrightarrow}^{\omega,d} \mu$. Therefore $s \approx^{d,\omega} s'$. Since we do not have $s \stackrel{a}{\Longrightarrow}^{fin,d} u$, we have $s \not\approx^{d,fin} s'$.

Example 5.4.13 Consider the system in Figure 5.9. We assume that u_i and u'_i are bisimilar. Then the relation $R = \{(s, s'), (u_1, u'_1), (u_2, u'_2), (u_3, u'_3), (u_4, u'_4)\}$ is a weak, deterministic, finitary simulation. The transition $s \xrightarrow{a} \mu$ is matched by $s' \stackrel{a}{\Longrightarrow}^{fin,d} \nu'$, where $\nu'(u'_i) = \frac{1}{4}$ for $i = 1, 2, 3, 4$ and clearly $\mu \sqsubseteq_R \nu'$. This illustrates the fact that weak bisimulation takes the cumulative probabilistic effect of several transitions into account.

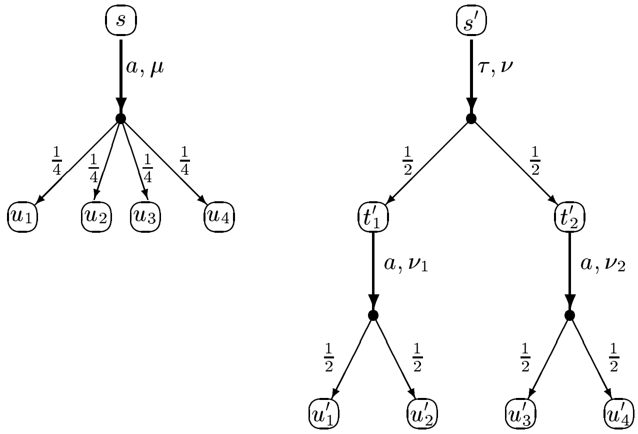


Figure 5.9: $s \preceq^{d,fin} s'$

5.4.4 Axiomatization

This section provides an alternative characterization of the finitary weak transition relations from Section 5.4.3 by means of axioms.

The nonprobabilistic, weak transition relation $\Longrightarrow$ is the smallest relation satisfying the following conditions (TR) (which yields the *trivial weak transition*), (PR) (which yields *proper transitions*), (WL) and (WR) (which assert that *weak* transitions can be combined with τ-transitions on the *left* and *right* respectively.

(TR) $s \stackrel{\varepsilon}{\Longrightarrow} s$.

(PR) If $s \xrightarrow{a} u$ then $s \stackrel{\widehat{a}}{\Longrightarrow} u$.

(TR) $s \overset{\widehat{\varepsilon}}{\Longrightarrow} \mu_s^1$

(PR) If $s \overset{a}{\longrightarrow} \mu$ then $s \overset{\widehat{a}}{\Longrightarrow} \mu$

(CC) If $s \overset{\widehat{a}}{\Longrightarrow} \mu_i$, $i = 1, \ldots, l$, then

$$s \overset{\widehat{a}}{\Longrightarrow} \mu \text{ for any convex combination } \mu \text{ of } \mu_1, \ldots, \mu_l$$

(WL) If $s \overset{\tau}{\to} \nu$ and $\forall t \in supp(\nu)[s \overset{a}{\Longrightarrow} \mu_t]$, then $s \overset{\widehat{a}}{\Longrightarrow} \sum_{t \in supp(\nu)} \nu(t) \cdot \mu_t$

(WR) If $s \overset{\widehat{a}}{\Longrightarrow} \mu'$ and $\forall t \in supp(\mu')[\nu_t = \mu_t^1 \vee t \overset{\tau}{\longrightarrow} \nu_t]$, then

$$s \overset{\widehat{a}}{\Longrightarrow} \sum_{t \in supp(\mu')} \mu'(t) \cdot \nu_t.$$

Figure 5.10: Axioms and rules for the finitary, weak transitions

(WL) $s \overset{\widehat{a}}{\Longrightarrow} u$ if there exist a transition $s \overset{\tau}{\Longrightarrow} t$ and a weak transition $t \overset{\widehat{a}}{\Longrightarrow} u$.

(WR) $s \overset{\widehat{a}}{\Longrightarrow} u$ if there exist a weak transition $s \overset{\widehat{a}}{\Longrightarrow} t$ and a transition $t \overset{\tau}{\Longrightarrow} u$.

Figure 5.10 shows how these rules can be lifted to the probabilistic setting, yielding the *finitary* variants.

Proposition 5.4.14 *The finitary weak transition relations can be characterized as the smallest subsets of $S \times Act_\varepsilon \times Distr(S)$ satisfying the following axioms and rules of Figure 5.10.*

- *for the finitary, deterministic, weak transitions:* (TR), (PR), (WL), (WR).
- *for the finitary, randomized, weak transitions:* (TR), (PR), (CC), (WL), (WR).

PROOF: (sketch) On the one hand, we have to prove that every finitary, deterministic, weak transition can be derived from the rules (TR), (PR), (WL) and (WR). This is a routine proof by induction on $\max\{length(\sigma) \mid \sigma \in Path^A_{*,max}(s)\}$, where A is a partial adversary that establishes $s \overset{a}{\Longrightarrow} \mu$. Since A is finitary, Königs Lemma yields that this maximum is finite. For the randomized, finitary, weak transition relations, one uses the result for the deterministic variants together with Proposition 5.3.24.

On the other hand, we have to prove that each transition that can be derived from the rules (TR), (PR), (WL) and (WR) (and from the rules (TR), (PR), (WL), (WR) and (CC), respectively) is a finitary, deterministic (randomized, respectively) transition. This is a routine proof by induction on structure of the derivation that establishes the transition. □

Example 5.4.15 Consider Figure 5.7. In Example 5.4.6, we have seen that $t_1 \overset{a}{\rightarrowtail} \nu$ with $\nu(t_2) = \nu(t_3) = \frac{1}{2}$. Then we also have $t_1 \overset{a}{\Longrightarrow}{}^{fin,r} \nu$. The latter is obtained by the following derivation.

$$\dfrac{\dfrac{t_1 \overset{a}{\to} \{t_2 \mapsto 1\}}{t_1 \overset{a}{\Longrightarrow} \{t_2 \mapsto 1\}}\,(\mathrm{PR}) \qquad \dfrac{t_1 \overset{a}{\to} \{t_3 \mapsto 1\}}{t_1 \overset{a}{\Longrightarrow} \{t_3 \mapsto 1\}}\,(\mathrm{PR})}{t_1 \overset{a}{\Longrightarrow} \frac{1}{2} \cdot \{t_2 \mapsto 1\} + \frac{1}{2} \cdot \{t_3 \mapsto 1\} = \nu}\,(\mathrm{CC})$$

Example 5.4.16 Consider Figure 5.9. Example 5.4.13 has shown that $s' \stackrel{a}{\Longrightarrow}^{fin,d} \nu'$, where $\nu'(u'_i) = \frac{1}{4}$ for $i = 1, 2, 3, 4$. This is also obtained by the following derivation.

$$\dfrac{s \xrightarrow{a} \nu \quad supp(\nu) = \{t'_1, t'_2\} \quad \dfrac{t'_1 \xrightarrow{a} \nu_1}{t'_1 \stackrel{a}{\Longrightarrow} \nu_1}\,(\mathrm{PR}) \quad \dfrac{t'_1 \xrightarrow{a} \nu_2}{t'_1 \stackrel{a}{\Longrightarrow} \nu_2}\,(\mathrm{PR})}{s' \stackrel{a}{\Longrightarrow} \nu(t_1)\cdot\nu_1 + \nu(t_1)\cdot\nu_2 = \frac{1}{2}\cdot\nu_1 + \frac{1}{2}\cdot\nu_2 = \nu'}\,(\mathrm{WL})$$

5.5 Delay Simulation and Delay Bisimulation

5.5.1 Delay (Bi–)simulation

This section introduces the new notions of *delay simulation* and *delay bisimulation*. These refine the (randomized) weak (bi–)simulation relations, but still abstract from internal moves in a certain way. We follow the lines of Section 5.4.3 and base the delay (bi–)simulation relations on *delay transitions*, which in their turn depend on certain classes of adversaries. As in Section 5.4.3, we consider finitary, almost finitary, randomized and deterministic variants. Unlike for weak transitions, the use of randomized and deterministic schedulers gives rise to the same notion of delay transition. The use of finitary and almost finitary adversaries, however, still yields two different notions of delay (bi–)simulation.

There are two ways in which the delay (bi–)simulations are more restrictive than the weak (bi–)simulations. Firstly, delay transitions are established by adversaries over the words in $\tau^*\widehat{a}$ rather than over words in $\tau^*\widehat{a}\tau^*$. Secondly, for a visible action a, a delay simulation relation R requires each step $s \xrightarrow{a} \mu$ to be matched by a delay transition $s \stackrel{a}{=\!=\!\Rightarrow} \rho$. The latter is established by an adversary A which can only schedule a step (a, ν) if $\mu \sqsubseteq_R \nu$. In order words, $A(\sigma)(a, \nu) > 0$ implies $\nu \in \mu\!\uparrow_R$. For the matching of τ–transitions $s \xrightarrow{\tau} \mu$, similar conditions are required.

Definition 5.5.1 [The partial adversary classes $\mathcal{C}[s, W, M]$] Let $\mathcal{C}$ be a class of partial adversaries, let $s \in S$, let $W \subseteq Act_\tau^\infty$ be a set of finite and infinite words, and let M be a set of distributions. Then set $\mathcal{C}[s, W, M]$ is defined as the set of adversaries A such that

- $A \in \mathcal{C}$,
- $Words(s, A) \subseteq W$ and
- for all maximal paths $\sigma = s_0 \overset{a_1,\mu_1}{\leadsto} s_1 \ldots \overset{a_n,\mu_n}{\leadsto} s_n$ in A, we have $\{s_0 \mapsto 1\} \in M$ if $n = 0$ and $\mu_n \in M$ if $n > 0$.

In the sequel, we consider adversaries over the set of words τ^*a. In particular, we work with the sets of adversaries $RAdv^{fin}[s, \tau^*\widehat{a}, \mu\uparrow_R]$, $DAdv^{fin}[s, \tau^*\widehat{a}, \mu\uparrow_R]$, $RAdv^{\omega}[s, \tau^*\widehat{a}, \mu\uparrow_R]$ and $DAdv^{\omega}[s, \tau^*\widehat{a}, \mu\uparrow_R]$. For a visible action a, the sets $RAdv_\perp^{fin}[s, \tau^*\widehat{a}, \mu\uparrow_R]$ and $DAdv_\perp^{fin}[s, \tau^*\widehat{a}, \mu\uparrow_R]$ correspond to finite trees whose paths are labeled by elements from τ^*a. The adversaries in $RAdv_\perp^{\omega}[s, \tau^*\widehat{a}, \mu\uparrow_R]$ and $DAdv_\perp^{\omega}[s, \tau^*\widehat{a}, \mu\uparrow_R]$ might have infinite paths, which are then labeled by τ–actions, but the probability measure on the set of infinite paths is zero. The finite maximal paths in these adversaries are labeled by elements from τ^*a, as before. As a consequence, each adversary in one of the classes above reaches some of the states s' with an outgoing a–transition $s' \xrightarrow{a} \nu$ such that $\mu \sqsubseteq_R \nu$ with probability one. The proof of the following proposition is routine.

Proposition 5.5.2 *For all partial adversaries A in $RAdv^{\omega}_{\perp}[s, \tau^*\widehat{a}, \mu\uparrow_R]$, the distribution $x \mapsto \mathbf{P}^A(s, x)$ is an element in $\mu\uparrow_R$.*

This result holds in particular for adversaries in $RAdv^{fin}_{\perp}[s, \tau^*\widehat{a}, \mu\uparrow_R]$, $DAdv^{fin}_{\perp}[s, \tau^*\widehat{a}, \mu\uparrow_R]$ and $DAdv^{\omega}_{\perp}[s, \tau^*\widehat{a}, \mu\uparrow_R]$, because these sets are all subsets of $RAdv^{\omega}_{\perp}[s, \tau^*\widehat{a}, \mu\uparrow_R]$. Now, we can define the several variants of delay transition and delay (bi–)simulation in the same way as we defined the weak transition and (bi–)simulation relations.

Definition 5.5.3 (Delay transitions) Define the deterministic, finitary delay transition relation $\Longrightarrow^{fin,d}$ as follows. For $s \in S$, $a \in Act_\tau$, $\mu \in Distr(S)$ and $M \subseteq Distr(S)$,

1. define $s \stackrel{\widehat{a}}{\Longrightarrow}{}^{fin,d}_M \mu$ iff there exists $A \in DAdv^{fin}_{\perp}[s, \tau^*\widehat{a}, M]$ with $\mu = \mathbf{P}^A(s, \cdot)$ and
2. define $s \stackrel{\widehat{a}}{\Longrightarrow}{}^{fin,d} M$ iff there exists $\mu \in M$ with $s \stackrel{\widehat{a}}{\Longrightarrow}{}^{fin,d}_M \mu$.

The delay transitions relations $\Longrightarrow^{fin,r}_M$, $\Longrightarrow^{\omega,d}_M$, $\Longrightarrow^{\omega,r}_M$, $\Longrightarrow^{fin,r} M$, $\Longrightarrow^{\omega,d} M$ and $\Longrightarrow^{\omega,r} M$ are defined by taking the corresponding classes of adversaries.

The following proposition is an immediate consequence of Proposition 5.5.2.

Proposition 5.5.4 *For all s, a and μ*

$$s \stackrel{\widehat{a}}{\Longrightarrow}{}^{fin,d} \mu\uparrow_R \textit{ iff } DAdv^{fin}_{\perp}[s, \tau^*\widehat{a}, \mu\uparrow_R] \neq \emptyset.$$

For the relations $\stackrel{\widehat{a}}{\Longrightarrow}{}^{fin,r} \mu\uparrow_R$, $\stackrel{\widehat{a}}{\Longrightarrow}{}^{\omega,d} \mu\uparrow_R$ and $\stackrel{\widehat{a}}{\Longrightarrow}{}^{\omega,r} \mu\uparrow_R$, we have similar results.

The following proposition states that the use of deterministic and randomized adversaries gives rise to the same notion of delay transitions if we consider target sets of the form $\mu\uparrow_R$. Therefore, we will omit the subscripts r and d in the delay transitions reaching such sets.

Proposition 5.5.5 *For all s, a and μ we have*

1. $s \stackrel{\widehat{a}}{\Longrightarrow}{}^{\omega,d} \mu\uparrow_R$ *iff* $s \stackrel{\widehat{a}}{\Longrightarrow}{}^{\omega,r} \mu\uparrow_R$.
2. $s \stackrel{\widehat{a}}{\Longrightarrow}{}^{fin,d} \mu\uparrow_R$ *iff* $s \stackrel{\widehat{a}}{\Longrightarrow}{}^{fin,r} \mu\uparrow_R$.

PROOF:

1. The "only-if" part is clear. For the "if"-part, we use Proposition 5.5.4. It is easy to see that if $A \in RAdv^{fin}_{\perp}[s, \tau^*\widehat{a}, \mu\uparrow_R]$, then every D in $\mathcal{D}(s, A)$ (where $\mathcal{D}(s, A)$ is as in Notation 5.3.23) is an element of $DAdv^{fin}_{\perp}[s, \tau^*\widehat{a}, \mu\uparrow_R]$.
2. Similarly.

□

The following proposition, which is an immediate consequence of the definitions, is used in the correctness arguments for the decidability algorithms for the delay (bi–)simulation relations.

Proposition 5.5.6 *For all s, a and M we have*

1. $s \stackrel{\widehat{a}}{\Longrightarrow}{}^{\omega,d} M$ *iff* $s \stackrel{\widehat{a}}{\Longrightarrow}{}^{\omega,d} M \cap \bigcup_{a,t} Steps_a(t)$.

2. $s \overset{\widehat{a}}{=\!=\!\Rightarrow}^{\mathit{fin},d} M$ *iff* $s \overset{\widehat{a}}{=\!=\!\Rightarrow}^{\mathit{fin},d} M \,\cap\, \bigcup_{a,t} \mathit{Steps}_a(t)$.

Example 5.5.7 Consider the systems in Figure 5.11. Let R be the smallest equivalence over S that identifies the elements s, t and t'.

- For the leftmost system, we have that $s \overset{\widehat{a}}{=\!=\!\Rightarrow}^{\mathit{fin}} \nu \uparrow_R$. Note that $\nu = \nu'$ and therefore $\nu \equiv_R \nu'$. This delay transition is established by the deterministic scheduler that in each state takes the unique outgoing transition with probability one.
- For the rightmost system in the same figure, we do not have $s \overset{\widehat{a}}{=\!=\!\Rightarrow}^{\mathit{fin}} \rho \uparrow_R$ because $\mathit{RAdv}^{\mathit{fin}}[s, \tau^* a, \rho \uparrow_R] = \emptyset$. Each scheduler over $\tau^* a$ has to schedule the ρ'–transition with a positive probability, but $\rho \not\equiv_R \rho'$.

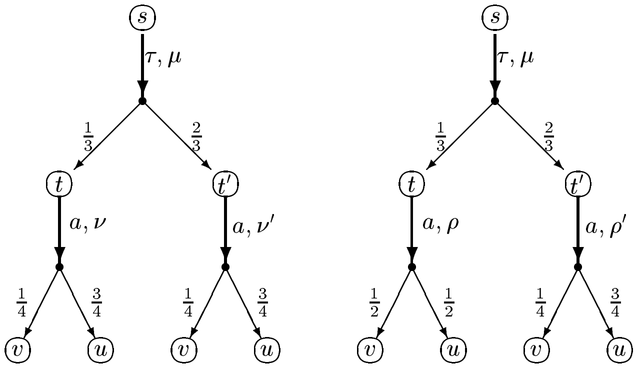


Figure 5.11: Delay transitions

Example 5.5.8 Reconsider the system in Figure 5.3 on page 95. Let R be the smallest equivalence relation identifying s and s' and identifying t and u. Since $\mu \equiv_R \rho$, the adversary D from Example 5.3.12 on page 94 is an element of $\mathit{DAdv}^{\mathit{fin}}[s, a, \mu \uparrow_R]$. Hence, D establishes that $s \overset{a}{=\!=\!\Rightarrow}^{\mathit{fin}} \mu \uparrow_R$. Similarly, we have that $s' \overset{a}{=\!=\!\Rightarrow}^{\mathit{fin}} \rho \uparrow_R$. Note that the step $s' \overset{\varepsilon}{=\!=\!\Rightarrow}^{\mathit{fin}} \{s' \mapsto 1\}$ is also a finitary delay transition.

Example 5.5.9 Now consider the system in Fig 5.8 on page 108. Let R be the smallest equivalence relation identifying s and s'. The adversary D_ω from Example 5.3.12 (page 94) is an element of $\mathit{DAdv}^{\omega}[s, a, \mu \uparrow_R]$ and hence establishes that $s \overset{a}{=\!=\!\Rightarrow}^{\omega} \mu \uparrow_R$. Note that $\mathit{DAdv}^{\mathit{fin}}[s, a, \mu \uparrow_R]$ is empty, so we do not have $s \overset{a}{=\!=\!\Rightarrow}^{\mathit{fin}} \mu \uparrow_R$.

Example 5.5.10 Consider the system in Figure 5.9 on page 108. Let $R = \{(s, s'), (u_1, u_1'), (u_2, u_2'), (u_3, u_3'), (u_4, u_4')\}$. Example 5.4.13 (page 5.4.13) has shown that $s' \overset{a}{\Longrightarrow}^{\mathit{fin},d} \nu' \uparrow_R$ for ν' with $\nu'(u_i') = \frac{1}{4}$. Since $\nu' \not\sqsubseteq_R \nu_1$, we have $\mathit{DAdv}^{\mathit{fin}}[s, a, \nu' \uparrow_R] = \emptyset$ and hence $s' \not\overset{a}{=\!=\!\Rightarrow}^{\mathit{fin}} \nu' \uparrow_R$. (For the same reason, we do not have $s' \overset{a}{=\!=\!\Rightarrow}^{\omega} \nu' \uparrow_R$ either.) This illustrates the fact that, unlike for weak transitions, the target distribution of a delayed transition cannot be obtained by "multiplying probabilities along the maximal paths."

Definition 5.5.11 (Delay (bi–)simulation) A binary relation R on S is called a *delay ω–simulation* iff for all $(s, s') \in R$ and $\mu \in \mathit{Distr}(S)$

$$\text{if } s \xrightarrow{a} \mu \text{ then } s' \overset{\widehat{a}}{=\!=\!\Rightarrow}{}^{\omega} \mu \uparrow_R.$$

A *delay ω–bisimulation* is an equivalence which is a delay ω–simulation. We write $s \preceq_{del}^{\omega} s'$ iff there exists a delay ω–simulation R such that $(s, s') \in R$. In that case, we say that s' *delay ω–simulates* s. We write $s \approx_{del}^{\omega} s'$ iff there exists a delay ω–bisimulation containing (s, s'). In that case s and s' are said to be *delay ω–bisimilar*. Now the finitary delay simulation relation $\approx_{del}^{fin}$ and the finitary delay bisimulation equivalence $\preceq_{del}^{fin}$ are defined in a similar way.

Example 5.5.12 Reconsider the system in Figure 5.3 on page 95. Let R be the smallest equivalence relation R identifying s and s' and identifying t and u. Then R is a finitary delay bisimulation. The step $s \xrightarrow{\tau} \nu$ in s is matched by the step $s' \overset{\varepsilon}{=\!=\!\Rightarrow}{}^{fin} \{s' \mapsto 1\}$. The step $s' \xrightarrow{a} \mu$ is matched by $s \overset{a}{=\!=\!\Rightarrow}{}^{fin} \mu \uparrow_R$ and the step $s \xrightarrow{a} \rho$ is matched by $s' \overset{a}{=\!=\!\Rightarrow}{}^{fin} \rho \uparrow_R$. Thus, $s \approx_{del}^{fin} s'$.

Example 5.5.13 Now consider the system in Fig 5.8 on page 108. Let R be the smallest equivalence relation R identifying s and s'. We claim that R is an almost finitary, delay bisimulation. The step $s \xrightarrow{\tau} \nu$ is matched by $s' \overset{\varepsilon}{=\!=\!\Rightarrow}{}^{\omega} \nu \uparrow_R$ as before and the step $s' \xrightarrow{a} \mu$ is matched by $s \overset{a}{=\!=\!\Rightarrow}{}^{\omega} \mu \uparrow_R$. Therefore, $s \approx_{del}^{\omega} s'$. Since $s \;\not\overset{a}{=\!=\!\Rightarrow}{}^{fin} \mu \uparrow_R$, we do not have $s \approx_{del}^{fin} s'$.

Example 5.5.14 Consider the system in Figure 5.9. Then the relation $R = \{(s, s'), (u_1, u_1'), (u_2, u_2'), (u_3, u_3'), (u_4, u_4')\}$ is not a delay simulation, because the transition $s \xrightarrow{a} \mu$ cannot be matched in the state s', since there is no outgoing delay transition labeled by a. Recall from Example 5.4.13 (page 108) that s' does weakly simulate s.

5.5.2 Alternative Characterization with Norm Functions

We now show that the several types of the delay relations can be characterized by probabilistic variants of *norm functions*, introduced by Griffioen & Vaandrager [GV98]. Our formulation differs from [GV98] in two ways. Firstly, we work with norm functions whose codomain are the natural numbers, whereas in [GV98], the range of a norm function can be any well–ordered set. Moreover, we adapt the formulation such that we characterize the delay variants, whereas [GV98] treats branching (bi-)simulations. As remarked before, it is not difficult to adapt the theory presented here for branching bisimulation.

In the nonprobabilistic case, a norm function for a binary relation R is a partial function $norm : S \times Act_\tau \times S \times S \hookrightarrow \mathbb{N}$ such that[9]

1. If $s \xrightarrow{a} u$ and $(s, s') \in R$, then $(s, a, u, s') \in dom(norm)$,
2. If $(s, a, u, s') \in dom(norm)$, then $s \xrightarrow{a} u$ and the following holds.
   - If $norm(s \xrightarrow{a} u, s') = 0$, then $a = \tau$ and $(u, s') \in R$,
   - if $norm(s \xrightarrow{a} u, s') = 1$, then there is a transition $s' \xrightarrow{a} u'$ where $(u, u') \in R$,
   - otherwise, there is a transition $s' \xrightarrow{\tau} t'$ such that

$$(s, a, u, t') \in dom(norm) \text{ and } norm(s \xrightarrow{a} u, t') < norm(s \xrightarrow{a} u, s').$$

[9] Recall that $dom(norm)$ denotes the domain of $norm$ (Section 5.2). We write $norm(s \xrightarrow{a} \mu, s')$ rather than $norm(s, a, \mu, s')$.

Intuitively, the value $norm(s \xrightarrow{a} u, s')$ is an upper bound for the number of τ transitions that are needed to establish a weak transition $s' \overset{a}{\Longrightarrow} u'$ such that $(u, u') \in R$. More precisely, if $(s, s') \in R$ and $norm(s \xrightarrow{a} u, s') = n$, then there is a path $s' = s_0 \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \dots s_n \xrightarrow{a} t'$ which consists of exactly n τ–transitions and ends with an a–transition such that $(s', t') \in R$. Thus, this path shows that $s' \overset{a}{\Longrightarrow} t'$ for some t' with $(s', t') \in R$. Using this property one can show that an equivalence R on the state space S is a weak bisimulation iff there is a norm function for R with

$$dom(norm) = \{(s, a, u, s') : s \xrightarrow{a} u \ \wedge \ (s, s') \in R\}.$$

Thus, norm functions provide an alternative characterization of weak[10] bisimulations.

For probabilistic systems, there are two alternative ways to generalize the third clause in condition (2) above. Firstly, given a transition $s \xrightarrow{a} \mu$ and a pair $(s, s') \in R$, we may require the existence of a transition $s' \xrightarrow{\tau} \nu'$ such that *for all states* $t' \in supp(\nu')$ the value $norm(s \xrightarrow{a} \mu, t')$ is defined and strictly less than $norm(s \xrightarrow{a} \mu, s')$. Second, we may use a weaker condition and require the existence of a transition $s' \xrightarrow{\tau} \nu'$ such that for all states $t' \in supp(\nu')$ the value $norm(s \xrightarrow{a} \mu, t')$ is defined and *for some state* $t' \in supp(\nu')$ $norm(s \xrightarrow{a} \mu, t')$ is strictly less than $norm(s \xrightarrow{a} \mu, s')$. We will see that norm functions of the former type provide a characterization of the finitary delay transitions, whereas the latter characterize the ω-variants.

Definition 5.5.15 (Norm functions) Let R be a binary relation on S. A *norm function* for R is partial function

$$norm : S \times Act_\tau \times Distr(S) \times S \rightharpoonup \mathbb{N}$$

which satisfies the following conditions.

1. If $s \xrightarrow{a} \mu$ and $(s, s') \in R$ then $(s, a, \mu, s') \in dom(norm)$.
2. If $(s, a, \mu, s') \in dom(norm)$ then $s \xrightarrow{a} u$ and one of the following conditions holds.
   - If $norm(s \xrightarrow{a} \mu, s') = 0$ then $a = \tau$ and $\mu \sqsubseteq_R \mu^1_{s'}$. [11]
   - If $norm(s \xrightarrow{a} \mu, s') = 1$ then there exists a transition $s' \xrightarrow{a} \mu'$ with $\mu \sqsubseteq_R \mu'$.
   - If $norm(s \xrightarrow{a} \mu, s') > 1$ then there exists a transition $s' \xrightarrow{\tau} \nu'$ such that

$$\forall t' \in supp(\nu')[(s, a, \mu, t') \in dom(norm)] \qquad \text{(WN)}$$

$$\exists t' \in supp(\nu')[norm(s \xrightarrow{a} \mu, t') < norm(s \xrightarrow{a} \mu, s')] \qquad \text{(N}\exists\text{)}$$

A norm function $norm$ for R is called *strict* iff condition (N$\exists$) is replaced by the following stronger condition (N$\forall$).

$$\forall t' \in supp(\nu')[norm(s \xrightarrow{a} \mu, t') < norm(s \xrightarrow{a} \mu, s')]. \qquad \text{(N}\forall\text{)}$$

[10] If we add the condition $(s, t') \in R$ in the third clause of (2), then we get a characterization of branching bisimulation.

[11] The condition $\mu \sqsubseteq_R \mu^1_{s'}$ is equivalent to the requirement that $supp(\mu) \subseteq \{t \in S : (t, s') \in R\}$.

Example 5.5.16 Consider the leftmost system in Figure 5.11. Let $R = \{(t, s), (t', s), (u, u), (v, v)\}$. A strict norm function for R is given by

$$norm(t \xrightarrow{a} \nu, s) = 2$$
$$norm(t' \xrightarrow{a} \nu', s) = 2$$
$$norm(t \xrightarrow{a} \nu, t) = 1$$
$$norm(t' \xrightarrow{a} \nu, t') = 1$$

and undefined for all other elements. The values for the function $x \mapsto norm(s' \xrightarrow{a} \nu, x)$ are depicted in Figure 5.12.

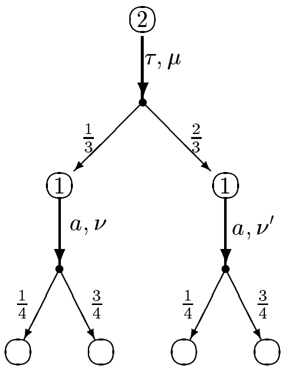


Figure 5.12: The function $x \mapsto norm(s \xrightarrow{a} \nu, x)$

Example 5.5.17 Reconsider the system in Figure 5.3 on page 95. Let R be the the smallest equivalence relation R identifying s and s' and identifying t and u. Then a strict norm function for R is given by

$$norm(s' \xrightarrow{a} \rho, s) = 1$$
$$norm(s' \xrightarrow{a} \rho, s') = 1$$
$$norm(s' \xrightarrow{a} \mu, s) = 1$$
$$norm(s' \xrightarrow{a} \mu, s') = 1$$
$$norm(s \xrightarrow{\tau} \nu, s) = 1$$
$$norm(s \xrightarrow{\tau} \nu, s') = 0$$

and undefined for all other elements. The values for the function $x \mapsto norm(s' \xrightarrow{a} \mu, x)$ are depicted in Figure 5.13.

Example 5.5.18 Now consider the system in Fig 5.8. Let R be the the smallest equivalence relation R identifying s and s'. A norm for R is given by

$$norm(s' \xrightarrow{a} \mu, s) = 2$$
$$norm(s' \xrightarrow{a} \mu, s') = 1$$
$$norm(s \xrightarrow{\tau} \nu, s) = 1$$
$$norm(s \xrightarrow{\tau} \nu, s') = 0$$

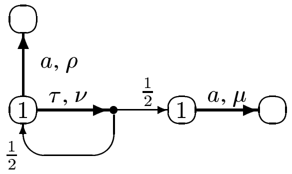


Figure 5.13: The function $x \mapsto \mathit{norm}(s' \xrightarrow{a} \mu, x)$

and undefined for all other elements. The values for the function $x \mapsto \mathit{norm}(s' \xrightarrow{a} \mu, x)$ are depicted in Figure 5.14.

Note that there is no strict norm function for R.

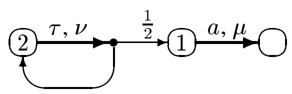


Figure 5.14: The function $x \mapsto \mathit{norm}(s' \xrightarrow{a} \mu, x)$

The reader may have noticed that the examples above correspond to Examples 5.5.8, 5.5.9 and 5.5.10, which showed the existence of delay transitions. This is not a coincidence. The following two propositions provide a connection between norm functions and the delay transition relations.

Proposition 5.5.19 *Let R be a binary relation on S and norm a norm function for R. Then, we have:*

1. *If $(s, a, \mu, s') \in \mathit{dom}(\mathit{norm})$ then $s' \overset{\widehat{a}}{\Longrightarrow}^{\omega} \mu{\uparrow}_R$.*
2. *If norm is strict and $(s, a, \mu, s') \in \mathit{dom}(\mathit{norm})$ then $s' \overset{\widehat{a}}{\Longrightarrow}^{\mathit{fin}} \mu{\uparrow}_R$.*

PROOF: Assume that $(s, a, \mu, s') \in \mathit{dom}(\mathit{norm})$ then $s \xrightarrow{a} \mu$ is a transition. We define a deterministic partial adversary D in $\mathit{DAdv}^{\omega}[s', \tau^* a, \mu{\uparrow}_R]$ and hence $s \overset{\widehat{a}}{\Longrightarrow}^{\omega} \mu{\uparrow}_R$.

First, define an auxiliary function $B : S \to \mathit{Act}_\tau \times \mathit{Distr}(S) \cup \{\bot\}$ as follows.

- If $(s, a, \mu, s') \notin \mathit{dom}(\mathit{norm})$, put $B(s') = \bot$,

and for $(s, a, \mu, s') \in \mathit{dom}(\mathit{norm})$, define $B(s')$ by

- if $\mathit{norm}(s \xrightarrow{a} \mu, s') = 0$, define $B(s') = \bot$,
- if $\mathit{norm}(s \xrightarrow{a} \mu, s') = 1$, choose $s' \xrightarrow{a} \mu'$, where $\mu \sqsubseteq_R \mu'$ and put $B(s') = (a, \mu')$,
- if $\mathit{norm}(s \xrightarrow{a} \mu, s') > 1$, then choose a transition $s' \xrightarrow{\tau} \nu'$ such that conditions (WN) and (N$\exists$) of Definition 5.5.15 are fulfilled and put $B(s') = (\tau, \nu')$. For strict normed functions we require condition (N$\forall$) instead of (N$\exists$).

Let $\Sigma = \{\sigma \in \mathit{Path}(s') \mid \mathit{word}(\sigma) \in \tau^*\}$. Now define the adversary D by

$$D(\sigma) = \begin{cases} B(\mathit{last}(\sigma)) & \text{if } \sigma \in \Sigma, \\ \bot & \text{otherwise.} \end{cases}$$

Below, we show that D is almost finitary and then it is easy to see that $D \in DAdv^{\omega}[s', \widehat{a}, \mu\!\uparrow_R]$. Then Proposition 5.5.4 implies $s' \stackrel{\widehat{a}}{\Longrightarrow}{}^{\omega} \mu\!\uparrow_R$ and we are done for statement 1. For part 2., we will prove below that, if the function *norm* is strict, then $D \in DAdv^{fin}[s', \widehat{a}, \mu\!\uparrow_R]$ and then we are also done by Proposition 5.5.4.

By the results of [Bai98] (or using the results of [BK98]) it follows that D is almost finitary. This can be seen as follows. The adversary D induces a finite Markov chain $(U, \mathbf{Q})$ with state space

$$U \; = \{u \in S : (s \xrightarrow{a} \mu, u) \in dom(norm)\}$$

and the transition probabilities $\mathbf{Q}$ given by

$$\mathbf{Q}(u,u') = \begin{cases} \nu(u') & \text{if } norm(s \xrightarrow{a} \mu, u) \geq 2 \text{ and } B(u) = (\tau, \nu), \\ 1 & \text{if } norm(s \xrightarrow{a} \mu, u) \leq 1 \text{ and } u = u', \\ 0 & \text{otherwise.} \end{cases}$$

The absorbing states in this Markov chain are given by the set $U_0 = \{u \mid norm(s \xrightarrow{a} \mu, u) \in \{0,1\}\}$. Let $Reach(u)$ denote the set of states that are reachable from u. Clearly, $Reach(u) \cap U_0 \neq \emptyset$ for all $u \in U$. Let $\mathbf{Q}^*(s', u')$ denote the probability for state s' to reach the state $u' \in U_0$ in this Markov chain. Then, an elementary result in the theory of Markov chains yields that the probability to reach a state in U_0 is 1, no matter from which state $u \in U$ one starts. It is not difficult to see that $\mathbf{P}^D(s, s') = \mathbf{Q}^*(s, s')$. As a consequence,

$$\mathbf{P}^D(s', S) \;=\; \sum_{u' \in U_0} \mathbf{Q}^*(s', u') \;=\; 1,$$

in other words, D is almost finitary.

Now, assume that *norm* is strict. Then, the adversary D defined above is finitary. This can be seen as follows. For any path $s' \stackrel{\tau}{\rightsquigarrow} s'_1 \stackrel{\tau}{\rightsquigarrow} \ldots \stackrel{\tau}{\rightsquigarrow} s'_l$ in D, we have

$$norm(s \xrightarrow{a} \mu, s') \;>\; norm(s \xrightarrow{a} \mu, s'_1) \;>\; \ldots \;>\; norm(s \xrightarrow{a} \mu, s'_l).$$

It then follows that any maximal path in D is finite and, therefore, D is finitary. □

Thus, the existence of a norm function for R implies that R is a delay ω-simulation. Similarly, if R has a strict norm function then R is a finitary delay simulation. The next proposition shows the converse, i.e. that any delay ω–simulation has a norm function and any delay finitary simulation has a strict norm function.

Proposition 5.5.20 *Let R be a binary relation on S.*

1. *If R is ω–simulation, then there exists a norm function norm such that $(s, a, \mu, s') \in dom(norm)$ whenever $(s, s') \in R$ and $s \xrightarrow{a} \mu$.*
2. *If R is finitary simulation, then there exists a strict norm function norm such that $(s, a, \mu, s') \in dom(norm)$ whenever $(s, s') \in R$ and $s \xrightarrow{a} \mu$.*

Proof: Let R be a binary subset of S, $a \in Act_\tau$ and $\rho \in Distr(S)$. We define $\mathcal{U}[s, a, \rho, R]$ as the set of deterministic almost finitary partial adversaries D that induces a delay ω–delay transition from s to $\rho\!\uparrow_R$. We define

$$n_R(s, a, \rho) \;=\; \min\,\{\; \ell(s, D) \;:\; D \in \mathcal{U}[s, a, \rho, R] \;\}$$

where $\min \emptyset = \infty$ and $\ell(s, D) = \min \left\{ \mathit{length}(\sigma) : \sigma \in \mathit{Path}^D_{\mathit{fmax}}(s) \right\}$ is the length of a shortest finite maximal path in D with starting state s. If $n_R(s', a, \rho) < \infty$ then we put

$$\mathit{norm}_R \left(s \xrightarrow{a} \mu, s' \right) = n_R(s', a, \mu).$$

It is easy to see that norm_R is a norm function for R provided that R is a delay ω-simulation. For part (b), we replace $\mathcal{U}[s, a, \rho, R]$ by $\mathcal{U}^{\mathit{fin}}[s, a, \rho, R]$, which is the set of all partial adversaries $D \in \mathcal{U}[s, a, \rho, R]$ that are finitary. Moreover, we replace $\ell(s, D)$ by $\ell^{\mathit{fin}}(s, D) = \max \left\{ \mathit{length}(\sigma) : \sigma \in \mathit{Path}^D_{\mathit{fmax}}(s) \right\}$. (Since D is finitary, this maximum is finite.) Hence, we get that norm_R is strict if R is a delay simulation. □

5.5.3 Axiomatization

The finitary delay transition can be characterized by axioms and rules in a similar way as the finitary weak transitions (Section 5.4.4). We adapt the rules (TR) and (PR) by requiring that they yield distributions in the class M.

Note that there are no counterparts to the rules (WR) and (CC). The counterpart of (WR) is missing because delay transitions do not combine τ-transitions on the right and the rule (CC) may be skipped since Proposition 5.5.5 states that the choice between deterministic and randomized partial adversaries is irrelevant for the delay transitions.

Proposition 5.5.21 *Consider the axioms and rules of Figure 5.15. The finitary delayed transition relation can be characterized as the smallest subsets of $S \times \mathit{Act}_\varepsilon \times \mathit{Distr}(S)$ satisfying the axioms and rules* (DelTR)*,* (DelPR) *and* (DelWL).

PROOF: By induction on the structure of the derivation, one easily shows that every transition that can be derived from the axioms is indeed a finitary delay transition. Conversely, to show that every delay transition can be derived from the axioms, we use an argument similar to the proof of Proposition 5.5.20. For $s \overset{a}{=\!=\!\Rightarrow} M$, define $\mathcal{U}^{\mathit{fin}}[s, a, M]$ as the set of deterministic finitary partial adversaries D that induce a deterministic delay transition from s to M and where $D(s) \neq \bot$. For every state s and every finitary deterministic adversary D, let $\ell(s, D) = \max\{\mathit{length}(\sigma) : \sigma \in \mathit{Path}^D_{\mathit{fmax}}(s)\}$. Since D is finitary, this maximum is finite. By induction on $\ell(s, D)$, it can be shown that for all deterministic adversaries D that establish $s \overset{a}{=\!=\!\Rightarrow} M$ we also have a derivation from the axioms. □

Example 5.5.22 Reconsider the leftmost system in Figure 5.11. Let R be the smallest equivalence over S that identifies the elements s, t and t'. Example 5.5.7 has shown that $s \overset{\widehat{a}}{=\!=\!\Rightarrow}{}^{\mathit{fin}} \nu\!\uparrow_R$. This can also be obtained by the following derivation, in which the final step is justified by the axiom (DelPR).

$$\dfrac{s \xrightarrow{\tau} \mu \;\; \mathit{supp}(\mu) = \{t, t'\} \quad \dfrac{t \xrightarrow{a} \nu\!\uparrow_R}{t \overset{a}{=\!=\!\Rightarrow}{}^{\mathit{fin}} \nu\!\uparrow_R}\,(\mathrm{DelPR}) \quad \dfrac{t' \xrightarrow{a} \nu'\!\uparrow_R = \nu\!\uparrow_R}{t' \overset{a}{=\!=\!\Rightarrow}{}^{\mathit{fin}} \nu\!\uparrow_R}}{s \overset{a}{=\!=\!\Rightarrow}{}^{\mathit{fin}} \nu\!\uparrow_R}\,(\mathrm{DelWL})$$

(DelTR) If $\mu_s^1 \in M$ then $s \stackrel{\varepsilon}{\Longrightarrow} M$.

(DelPR) If $s \xrightarrow{a} \mu$ and $\mu \in M$ then $s \stackrel{\widehat{a}}{\Longrightarrow} M$.

(DelWL) If $s \xrightarrow{\tau} \nu$ and $t \stackrel{a}{\Longrightarrow} M$ for all $t \in supp(\nu)$, then $s \stackrel{\widehat{a}}{\Longrightarrow} M$.

Figure 5.15: Axioms and rules for the finitary delay transitions

5.6 Basic Properties

This section lists several basic properties for simulation and bisimulation relations defined in previous sections. We focus on those properties that are important for the decidability algorithms in Section 5.7.

An important conclusion from this section is that the deterministic weak (bi–)simulation relations have properties that make them unsuitable for practical verification. All the other (bi–)simulations enjoy properties that are natural generalizations of the nonprobabilistic setting. In particular, this holds for the strong (bi–)simulation. Since strong (bi–)simulation can be considered as the deterministic variant of strong combined (bi–)simulation, the effects of determinism versus randomization differ in the strong and the weak case.

To avoid tedious repetitions, the results in this section have – quite arbitrarily – been formulated for ω–delay (bi–)simulation. However, the corresponding results hold and are also proven for the following variants.

- finitary delay (bi–)simulation,
- randomized, finitary (bi–)simulation,
- randomized, ω (bi–)simulation,
- strong (bi–)simulation,
- strong combined (bi–)simulation.

That is, in the propositions in this section, one can substitute the ω–delay (bi–)simulation with any of the variants listed above, provided that one also takes the corresponding transition relations. We provide several counter examples, showing that the results do not hold for the deterministic, weak ω and finitary (bi–)simulation.

In the nonprobabilistic case, it is well–known that one can replace the transition relation $\rightarrow$ in the definition of weak (bi–)simulation, by the weak transition relation $\Longrightarrow$. This characterization also holds in the probabilistic case when we deal with the randomized variants of simulations, but fails for the deterministic variants. This result plays an important role in the proofs for several other results. Note that, for the strong simulation relation, the proposition is exactly the same as the definition.

Proposition 5.6.1 *Let R be a binary relation (an equivalence relation, respectively) on S. Then R is a delay ω–simulation relation (a delay ω–bisimulation) if and only if for all $(s, s') \in R$ and all $\mu \in \mathrm{Distr}(S)$*

$$s \stackrel{a}{\Longrightarrow}^{\omega} \mu\!\uparrow_R \text{ implies } s' \stackrel{\widehat{a}}{\Longrightarrow}^{\omega} \mu\!\uparrow_R .$$

Remark 5.6.2 The corresponding result of Proposition 5.6.1 does not hold for the deterministic (bi-)simulation relations. For example, consider the states s, s' of Figure 5.16 and the smallest equivalence R that identifies the states s, s', t and u. Then s and s' are weakly, deterministically bisimilar. However, the weak, deterministic transition $s \overset{a}{\Longrightarrow}^{\mathit{fin},d}\mu$, where $\mu(x) = 1/3$, $\mu(y) = 2/3$, cannot be matched by a weak, deterministic transition leaving from s'.

The following result is essential for the correctness of the decidability procedures based on partition refinement.

Proposition 5.6.3 *1. The relation $\preceq^{\omega}_{del}$ is a preorder.*

2. The relation $\preceq^{\omega}_{del}$ is the coarsest delay ω–simulation relation.

PROOF: As in the nonprobabilistic case, using Proposition 5.6.1. □

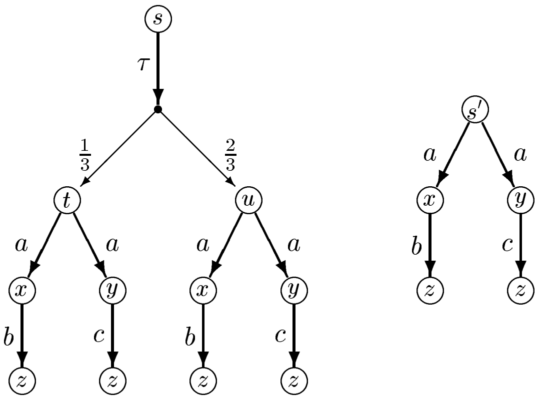


Figure 5.16: $s \approx^{d,\mathit{fin}} s'$ and $s \overset{a}{\Longrightarrow}^{\mathit{fin},d}\mu$, but not $s' \overset{a}{\Longrightarrow}^{\mathit{fin},d}\mu$.

Proposition 5.6.4 *1. The relation $\approx^{\omega}_{del}$ is an equivalence.*

2. The relation $\approx^{\omega}_{del}$ is the coarsest delay ω–bisimulation relation.

PROOF: As in the nonprobabilistic case, using Proposition 5.6.1. □

The following example shows that the deterministic simulations are not transitive.

Example 5.6.5 Consider the system in Figure 5.17. We assume that s_2 is not simulated by t_3, s_3 not by s_2, t_2 not by u_3, t_3 not by u_2, due to transitions not shown in the picture.

Let

$$R = \{(s_1, t_1), (s_2, t_2), (s_3, t_3), (s_4, t_4), (s_5, t_5)\}$$
$$T = \{(t_1, u_1), (t_1', u_1), (t_1'', u_1), (t_2, u_2), (t_3, u_3), (t_4, u_4), (t_5, u_5)\}.$$

Then R shows that $s_1 \preceq^{d}_{wbis} t_1$ and T establishes $t_1 \preceq^{d}_{wbis} u_1$. Due to the absence of randomized adversaries, we *do not* have $s_1 \preceq^{d}_{wbis} u_1$.

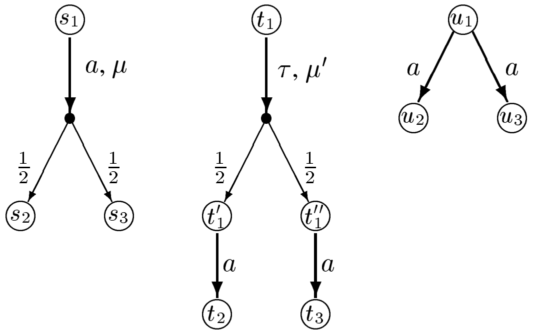


Figure 5.17: $s_1 \preceq^d_{wbis} t_1$, $t_1 \preceq^d_{wbis} u_1$, $s_1 \not\preceq^d_{wbis} u_1$ and $s_1 \not\preceq_{dwbis} u_1$.

The following proposition states that our notions of probabilistic (bi–)simulation are conservative extensions of (bi–)simulation on nonprobabilistic systems.

Proposition 5.6.6 *When applied to a nonprobabilistic labeled transition system $(S, \longrightarrow)$, the notions of strong and strong combined (bi–)simulation coincide with the classical notions of strong (bi–)simulation on nonprobabilistic systems. Similarly, the notion of delay (bi–)simulation coincides with the classical notion of delay (bi–)simulation on nonprobabilistic systems. Finally, the notion of weak (bi–)simulation coincides with the classical notion of delay weak (bi–)simulation on nonprobabilistic systems [Mil80].*

Proposition 5.6.7 *The deterministic weak (bi-)simulation relations are strictly coarser than the randomized ones, the finitary (bi-)simulation relations are strictly coarser than the ω–ones. The strong (bi-)simulation simulation relations are strictly finer than the strong combined ones. The delay weak (bi-)simulation variants are strictly between the strong and the weak (bi-)simulation. The strong combined (bi-)simulation variants and the delay variants are incomparable.*

Proposition 5.6.8 (cf. [Seg95b]) *The $\preceq^{\omega}_{del}$ is a precongruence and the $\approx^{\omega}_{del}$ is a congruence with respect to the parallel composition operator.*

The following theorem states the soundness of the simulations for trace distribution inclusion $\sqsubseteq_{\mathrm{TD}}$ and of the bisimulations for trace distribution equivalence $\equiv_{\mathrm{TD}}$. Recall that $\sqsubseteq_{\mathrm{TD}}$ and $\equiv_{\mathrm{TD}}$ were defined in Definition 2.4.1 on page 49 and that the (bi–)simulation relations can be lifted to PAs via their start states.

Theorem 5.6.9 *1. If $\mathcal{A} \preceq^{\omega}_{del} \mathcal{B}$, then $\mathcal{A} \sqsubseteq_{\mathrm{TD}} \mathcal{B}$.*

2. If $\mathcal{A} \approx^{\omega}_{del} \mathcal{B}$, then $\mathcal{A} \equiv_{\mathrm{TD}} \mathcal{B}$.

PROOF: (sketch) For the proof of part (1) of this theorem, we refer to Segala [Seg95b]. Since any of the simulation relations considered in this chapter is finer than the weak, randomized ω–simulations and since the latter are proven to be sound for trace distribution inclusion, all the relations from this chapter are also sound for it. Then the proof of part (2) follows immediately. □

5.7 Decidability Algorithms for Simulation and Bisimulation Relations

In the previous sections, we treated the notions of strong, strong combined, weak and delayed (bi–)simulation. This section is concerned with the decidability and complexity results for these (bi–)simulation relations.

A decidability algorithm for a bisimulation relation $\approx$ is an algorithm that takes a system $(S, \rightarrow)$ and two states s and s' in S and yields whether or not $s \approx s'$. The algorithms in this section for deciding bisimulations are all based on a so–called *partition refinement technique*. This means that the algorithm obtains the partition $S/\approx$ by starting from the trivial partition $\{S\}$ and successively refining it into finer partitions until the result $S/\approx$ is obtained. Then one can check whether two states are bisimilar by verifying whether they are in the same block in $S/\approx$. As observed by [Her99], computing the entire quotient space just for deciding the bisimilarity of two states may seem too much work, but is in fact not, because bisimilarity intrinsically involves all states of the system.

The algorithms for deciding simulation preorders work similarly. These compute the pairs contained in a simulation relation $\sqsubseteq$ by starting with the trivial preorder S^2 and successively remove pairs which are not in the simulation preorder until the relation $\sqsubseteq$ is obtained.

In this section, we will firstly summarize the main existing algorithmic methods for computing the (bi–)simulation relations, both for the probabilistic and the nonprobabilistic case. We concentrate on those aspects relevant for the algorithms to compute the delay (bi–)simulations. We treat polynomial time algorithms for deciding strong and weak (bi–)simulation in the nonprobabilistic case and for strong (bi–)simulation in the probabilistic case. To the best of our knowledge, there are no published results on the decidability of strong combined and weak (bi–)simulation relations from Section 5.4, but these are strongly conjectured to be decidable in polynomial time [SC01].

Most of this section is devoted to the algorithms for deciding the delay simulation and bisimulation relations defined in Section 5.5. All these algorithms run in polynomial time and space.

In the sequel, we assume a finite probabilistic system $(S, Steps)$ with n states and m transitions (i.e. $n = |S|$ and $m = \sum_{s \in S} |Steps(s)|$).

5.7.1 Decidability of Strong (Bi–)simulation

The algorithms to compute the strong (bi–)simulations are the basis for the algorithms computing several other bisimulation relations.

Strong bisimulation The main idea for computing the strong bisimulation equivalence classes in nonprobabilistic systems is the use of a *partitioning technique* as proposed by Kanellakis & Smolka [KS90] (and improved by Paige & Tarjan [PT87]). This technique is sketched in Figure 5.18: We start with the trivial partition $\mathcal{X} = \{S\}$ of the state space S and then successively refine $\mathcal{X}$ by splitting the blocks B of $\mathcal{X}$ into subblocks, until splitting is not possible anymore. In that case, the partition $\mathcal{X}$ equals the quotient space $S/\approx_{sbis}$ and we are done.

The refinement operator depends on a *splitter* of $\mathcal{X}$. Intuitively, a splitter is a pair $\langle a, A \rangle$ consisting of an action a and a block $A \in \mathcal{X}$ that prevents the induced equivalence $R_{\mathcal{X}}$ to fulfill the condition of a strong bisimulation. That is, a splitter is a pair $\langle a, A \rangle$ such that

Computing the bisimulation equivalence classes

Input: a finite (nonprobabilistic or probabilistic) system with state space S

Output: $S/\approx_{bis}$ for a (strong or weak) bisimulation equivalence $\approx_{bis}$

Method:

While there exists a splitter candidate sc of $\mathcal{X}$ do
$\mathcal{X} := \mathit{Refine}(\mathcal{X}, sc)$;
Return $\mathcal{X}$.

Figure 5.18: Schema for computing the bisimulation equivalence classes

there are states s, $s' \in S$ that belong to the same block C of $\mathcal{X}$), but where $s \xrightarrow{a} A$ while $s \not\xrightarrow{a} A$. s' cannot move via a to a state of A[12]. Then we know that s and s' are not bisimilar and therefore we split the block C into two subblocks, $C_1 = \{s \in C \mid s \xrightarrow{a} A\}$ and $C_2 = \{s \in C \mid s \not\xrightarrow{a} A\}$. More precisely, if $\mathit{Pre}(a, A) = \{s \in S \mid s \xrightarrow{a} A\}$ denotes the set of a-predecessors of A, then $C_1 = C \cap \mathit{Pre}(a, A)$ and $C_2 = C \setminus \mathit{Pre}(a, A)$. Thus, define

$$\mathit{Refine}(C, a, A) \;=\; \{C \cap \mathit{Pre}(a, A), C \setminus \mathit{Pre}(a, A)\} \setminus \{\emptyset\}.$$

and

$$\mathit{Refine}(\mathcal{X}, a, A) \;=\; \bigcup_{C \in \mathcal{X}} Refine(C, a, A)$$

Note that, if the *Refine*–operator is applied, it is not checked whether an action/block pair $\langle a, A\rangle$ is indeed a splitter. This can be done because if $\langle a, A\rangle$ is not a splitter, then we have $\mathit{Refine}(\chi, a, C) = \chi$ and therefore splitting is harmless. It follows now Proposition 5.6.4 that $\mathcal{X} = S/{\approx_{sbis}}$ iff no more splitters are available. Using an efficient organization of the splitter candidates, this method can be implemented in time $\mathcal{O}(m \log n)$ [PT87] (see also [Fer90]).

Strong bisimulation in probabilistic systems As mentioned by Huynh & Tian [HT92], the splitter/partitioning technique can easily be modified to fully probabilistic systems and to reactive systems. The complexity for these systems is $\mathcal{O}(nk \log n)$, where k is the number of nonzero entries in the transition probability matrix.

In [BEM00], a modification of the partitioning technique is given to compute the strong bisimulation equivalence classes in time $\mathcal{O}(nm(\log m + \log n))$ and space $\mathcal{O}(nm)$. The main idea is no longer to split with respect to action/block pairs but w.r.t pairs $\langle a, M\rangle$, where a is an action and M an equivalence class of $\equiv_\chi$. We will also use this idea in our algorithms.

Strong simulation Henzinger, Henzinger & Kopke [HHK95] present an algorithm for computing the strong simulation preorder in nonprobabilistic systems that has time complexity $\mathcal{O}(nm)$. The schema of this method is sketched in Figure 5.19.

[12]Here, we write $s \xrightarrow{a} A$ iff $s \xrightarrow{a} u$ for some $u \in A$.

Computing the simulation preorder

Input: a finite (probabilistic or nonprobabilistic) system with state space S

Output: the simulation preorder $\preceq_{sim}$ on S

Method:

$R \ := \ S \times S$;
While there exists $(s, s') \in R$ with $s \not\sqsubseteq_R s'$ do $R := R \setminus \{(s, s')\}$;
Return R.

Figure 5.19: General schema for computing the simulation preorder

We start with the trivial preorder $R = S \times S$ and then successively remove those pairs (s, s') from R, iff $s \not\sqsubseteq_R s'$, in other words, if s has a transition that cannot be "simulated" by a transition of s'. More precisely, in the nonprobabilistic case, we have

$$s \sqsubseteq_R s' \text{ iff for each transition } s \xrightarrow{a} t \text{ there is a transition } s' \xrightarrow{a} t' \text{ with } (t, t') \in R.$$

In [Bai96, Bai98], it is shown that the schema of Figure 5.19 is also applicable for the strong simulation preorder $\preceq_{ssim}$ in probabilistic systems. Here, the relation $\sqsubseteq_R$ as a relation on the state space S of a probabilistic system is given by:

$$s \sqsubseteq_R s' \text{ iff for each transition } s \xrightarrow{a} \mu \text{ there is a transition } s' \xrightarrow{a} \mu' \text{ with } \mu \sqsubseteq_R \mu'.$$

The algorithm proposed in [Bai96, Bai98] requires polynomial time and space, namely, time $\mathcal{O}((mn^6 + m^2n^3)/\log n)$ and space $\mathcal{O}(mn + n^2)$. Another important insight in [Bai96], which we also need in our algorithms, is that the relation $\sqsubseteq_R$ can be decided via a maximal flow in a network.

Strong combined (bi–)simulation As far as the authors know, there are no algorithms for deciding the strong combined simulation and bisimulation from Section 5.4.2. The main problem here is that the relation $\rightarrowtail$ can be infinite, even for finite systems. These, however, could be solved by the methods proposed by Segala [SC01].

5.7.2 Decidability of Weak (Bi–)simulation

The key to the algorithms for deciding weak bisimilarity is that the weak bisimulation equivalence classes in a nonprobabilistic system $(S, \longrightarrow)$ agree with the strong bisimulation equivalence classes in the system $(S, \Longrightarrow)$. In the latter system, the action set is Act_ε and the transition relation $\Longrightarrow \subseteq S \times Act_\varepsilon \times S$ has been given in Section 5.4.3. Thus, for computing the weak bisimulation equivalence classes in nonprobabilistic systems, one first calculates the transitive closure of the τ-labeled transitions from which the weak transition relation $\Longrightarrow$ can be derived. Then, the above mentioned splitter/partitioning technique can be applied to the system $(S, \Longrightarrow)$. The efficiency of this method crucially depends on the transitive closure operation. Using the method of [CW87], one gets an $\mathcal{O}(n^{2.3})$ algorithm for computing the weak transition relation $\Longrightarrow$ and the weak bisimulation equivalence classes. For further details see [BS87].

As for weak bisimulation equivalence, the weak simulation preorder in $(S, \longrightarrow)$ agrees with the strong simulation preorder in the system $(S, \Longrightarrow)$. Hence, the methods of [HHK95] combined with a method for computing the relation $\Longrightarrow$ yields an algorithm for computing the strong simulation preorder in nonprobabilistic systems.

Weak (bi–)simulation in probabilistic systems Weak (bi–)simulation equivalence in a probabilistic system (S, Steps) is the same as strong bisimulation equivalence in the probabilistic system $(S, \widehat{\mathit{Steps}})$. In the latter, we deal with the action set $Act_\varepsilon = Act \cup \{\varepsilon\}$ and $\widehat{\mathit{Steps}}(s) = \{(a, \mu) : s \stackrel{a}{\Longrightarrow} \mu\}$. However, an algorithmic reduction of the weak bisimulation problem to the strong bisimulation problem is not possible since the system $(S, \widehat{\mathit{Steps}})$ might be infinite. As far as the authors know, the question of whether (any type of) weak or branching bisimulation equivalence is decidable for probabilistic systems in our sense is still open. The main problem with weak and branching bisimulation in probabilistic systems is that the number of weak transitions might be infinite.

To the best of our knowledge, there are no published results on the decidability of (any type of) weak (bi–)simulation from Section 5.4. The main problem here is that the number of weak transitions might be infinite. Hence, we cannot reduce the weak (bi–)simulation problem in a system (S, Steps) to the strong (bi–)simulation problem in $(S, \widehat{\mathit{Steps}})$, where $\widehat{\mathit{Steps}}(s) = \{(a, \mu) : s \stackrel{a}{\Longrightarrow} \mu\}$, as we did for weak (bi–)simulation in nonprobabilistic systems. Segala [SC01] has proposed a solution for this, using similar techniques as in [SC01], yielding a polynomial time algorithm for deciding weak, randomized bisimulation. A publication of these results is currently in preparation.

On the other hand, there exist decidability results for several other notions of weak bisimulation for probabilistic systems.

In [BH97], the notions of weak and branching bisimulation are proposed for fully probabilistic systems. It is shown that weak and branching bisimulation equivalence coincide and that the splitter/partitioning technique is applicable to get an $\mathcal{O}(n^3)$ algorithm for computing the weak/branching bisimulation equivalence.

5.7.3 The delay predecessor predicates

The algorithm sketched in Figure 5.20 for deciding strong bisimulation depends on the operator *Refine*, which in its turn depends on a set $\mathit{Pre}(a, A)$ and also the algorithm for deciding simulation depends on this predicate. The decidability algorithms for delay (bi–)simulation depend on a similar predicate, namely $\mathit{Pre}^{str}(a, M)$ for the finitary delay variants and $\mathit{Pre}^{\omega}(a, M)$ for the ω delay variants. This section is concerned with algorithms for those two sets.

Definition 5.7.1 (Strict and ω-predecessors) The sets of strict and ω–predecessors of an action a and a set of distributions M are defined respectively by

$$\mathit{Pre}^{str}(a, M) = \{s \in S \mid s \stackrel{\widehat{a}}{=\!=\!\Rightarrow}^{\mathit{fin}} M\}$$
$$\mathit{Pre}^{\omega}(a, M) = \{s \in S \mid s \stackrel{\widehat{a}}{=\!=\!\Rightarrow}^{\omega} M\}$$

We present the algorithms for computing the predecessors for actions $a \neq \tau$ because this is technically simpler and we do not need the predecessors of the τ action for deciding the (bi–)simulation relations. However, it is easy to adapt the given algorithms in such a way that these can also handle τ's (with the same complexity).

The strict predecessors

The fixed point characterization of the finitary delay predicates from Section 5.4.4 yields an (inefficient) algorithm to compute the set $Pre^{str}(a, M)$. We start from $P = \{s \mid s \xrightarrow{a} M\}$ and whenever we find $t \xrightarrow{\tau} \nu$ with $supp(\nu) \subseteq P$ and $s \notin P$, we add s to P, until a fixed point has been reached. In that case, $P = Pre^{str}(a, M)$.

We improve this algorithm. We basically simplify the test whether $t \xrightarrow{\tau} \nu$ is a transition satisfying $supp(\nu) \subseteq P$ by using information obtained in previous tests. The idea is as follows. For each distribution ν occurring in a transition $t \xrightarrow{\tau} \nu$, we keep a counter $c(\nu)$ yielding the number of states $s \in supp(\nu)$ for which the condition $s \in P$ has not been verified yet. Thus, initially, $c(\nu)$ equals the number of elements in $supp(\nu)$ and each time we find $s \in supp(\nu) \cap P$, we decrease the counter with 1 until $c(\nu) = 0$. In that case, we know that all t with $t \xrightarrow{\tau} \nu$ may be inserted into P, because all elements in $supp(\nu)$ are so. Obviously, adding t to P may cause other counters to decrease.

For an efficient handling of the counters, we wish to have easy access to all the distributions that have a certain element s in their supports and to all the states that have an outgoing τ transition leading to a given distribution ν. For this, consider the directed graph $G^{str}(S, Steps) = (V, E)$ with

$$N = \bigcup_{s \in S} Steps_{\tau}(s).$$

the vertex set V is $S \cup N$ where the set $E \subseteq S \times N \cup N \times S$ of edges is defined by

$$E = \left\{(\nu, s) \in N \times S : s \xrightarrow{\tau} \nu\right\} \cup \{(s, \nu) \in S \times N : s \in supp(\nu)\}.$$

Thus, $E(s)$ yields the set of distributions whose support contains the state s and $E(\nu)$ yields the set of states with an outgoing τ–transition leading to ν. The variable N_0 in the algorithm collects all distributions ν with $c(\nu) = 0$. Thus, N_0 can be considered as a wait list of distributions that are waiting to be processed.

Now consider the following algorithm in Figure 5.20. We assume that the transition system $(S, Steps)$ and the set M are represented by adjacency lists.

Example 5.7.2 Consider the system in Figure 5.21. We show how the algorithm computes the set $Pre^{str}(a, M)$, where we assume transitions $u \xrightarrow{a} M$ and $u' \xrightarrow{a} M$. In this example, if we only mention variable if its value has changed. Initially, we have

$$\begin{array}{ll} E(s) = \emptyset & E(\nu_1) = \{s\} \\ E(t) = E(t') = \{\nu_1\} & E(\nu_2) = \{t\} \\ E(u) = \{\nu_2, \nu_3\} & E(\nu_3) = \{t'\} \\ E(u') = \{\nu_1, \nu_2, \nu_3\} & \end{array}$$

$Pre^{str}(a, M) = \emptyset$, $N_0 = \emptyset$, $c(\nu_1) = 3$ and $c(\nu_2) = c(\nu_3) = 2$. Executing the first for–loop with $s = u_1$ yields $Pre^{str}(a, M) = \{u_1\}$ and $c(\nu_2) = c(\nu_3) = 1$ and executing

Input: probabilistic system $(S, Steps)$, $a \in Act$ and a set of distributions $M \subseteq \bigcup_s Steps_a(s)$.

Compute the adjacency lists $E(\cdot)$ of the graph $G^{str}(S, Steps)$;

Put $P := \emptyset$, $N_0 := \emptyset$ and $c(\nu) := |supp(\nu)|$ for all $\nu \in N$;

For all $s \in S$ where $s \xrightarrow{a} \mu$ for some $\mu \in M$ do:

    $P := P \cup \{s\}$;
    For all $\nu \in E(s)$ do
        $c(\nu) := c(\nu) - 1$;
        If $c(\nu) = 0$ then $N_0 := N_0 \cup \{\nu\}$;

While $N_0 \neq \emptyset$ do

    choose some $\nu_0 \in N_0$ and put $N_0 := N_0 \setminus \{\nu_0\}$;
    For all $s \in E(\nu_0) \setminus P$ do
        $P := P \cup \{s\}$;
        For all $\nu \in E(s) \setminus N_0$ do
            $c(\nu) := c(\nu) - 1$;
            If $c(\nu) = 0$ then $N_0 := N_0 \cup \{\nu\}$;

Return P.

Figure 5.20: Algorithm for computing $Pre^{str}(a, M)$

the same loop with $s = u_2$ yields $Pre^{str}(a, M) = \{u_1, u_2\}$ and $c(\nu_2) = c(\nu_3) = 0$ and $N_0 = \{\nu_2, \nu_3\}$. Then, for the second part of the algorithm, execution of the while–loop with $\nu_0 = \nu_2$ yields $Pre^{str}(a, M) = \{u_1, u_2, t\}$, $c(\nu_1) = 1$ and $N_0 = \{\nu_3\}$. Execution of the while–loop with $\nu_0 = \nu_3$ yields $Pre^{str}(a, M) = \{u_1, u_2, t, t'\}$, $c(\nu_1) = 0$ and $N_0 = \{\nu_1\}$. Finally, execution with $\nu_0 = \nu_3$ yields $Pre^{str}(a, M) = \{u_1, u_2, t, t', s\}$ and $N_0 = \emptyset$.

Proposition 5.7.3 *The algorithm in Figure 5.20 computes the set* $Pre^{str}(a, M)$.

Proposition 5.7.4 *The algorithm in Figure 5.20 runs in time and space* $\mathcal{O}(nm)$.

PROOF: The space complexity is clear. The time complexity follows easily from the following observations.

Building the graph $G^{str}(S, Steps)$ takes time $\mathcal{O}(nm)$. The first part (i.e. the first for–loop) of the algorithm clearly has complexity $\mathcal{O}(nm)$ because $|S| = n$ and $|E(s)| \leq m$ for each s.

For the second part of the algorithm, observe that each element ν_0 is inserted in the set N_0 only once: this follows from the fact that ν_0 is added to N_0 when its counter is set to value 0 and the fact that counters only decrease. Hence, for each $\nu \in N$ the while–loop

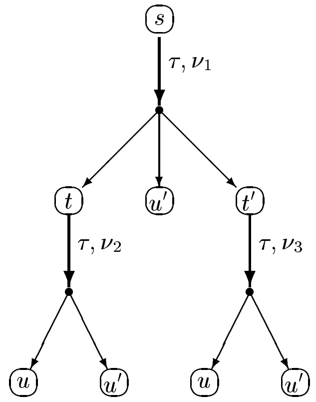


Figure 5.21: The computation of $Pre^{str}(a, M)$

in the algorithm is executed at most once. Then it is not difficult to see that, moreover, the combination of the while–loop and the outermost for–loop is executed at most once for each of the pairs (ν_0, s) such that $s \in E(\nu_0)$. Since there are at most m of those pairs the combination of those loops is executed at most m times. Since innermost for–loop is has complexity $\mathcal{O}(n)$, the entire algorithm runs in time $\mathcal{O}(nm)$. $\square$

The ω-predecessors:

Our algorithm to compute $Pre^{\omega}(a, M)$ is based on the following characterization.

Proposition 5.7.5

$$s \stackrel{a}{\Longrightarrow}^{\omega} \mu\!\uparrow_R \iff \mathbf{Q}^A(s, \tau^*, M) = 1, \textit{ for some adversary } A.$$

where $M = \{s' \mid s' \xrightarrow{a} \nu, \nu \in \mu\!\uparrow_R\}$.

Algorithms for computing the set $\{s \mid \mathbf{Q}^A(s, \tau^*, M), A \in DAdv^{\omega}_{\perp}\}$ can be found in [Var85]. They run in time $O(n^2m^2)$ and space $\mathcal{O}(nm)$. This implies the following.

Proposition 5.7.6 *The set* $Pre^{\omega}(a, M)$ *can be computed in time* $O(n^2m^2)$ *and space* $\mathcal{O}(nm)$.

5.7.4 Delay Bisimulation Equivalence

This section presents the algorithms for computing the delay bisimulation equivalences $\approx^{fin}_{del}$ and $\approx^{\omega}_{del}$; the algorithms for deciding the delay simulations preorders $\preceq^{fin}_{del}$ and $\preceq^{\omega}_{del}$ will be given in Section 5.7.5.

The algorithms in this section adapt the splitter/partitioning technique from Figure 5.18, which leads to the basic algorithm given in Figure 5.22. This algorithm is further refined in Figure 5.24. Operations appearing in these algorithms can be computed as given in Figures 5.25 and 5.26. We will first state the main result in this section, which is the polynomial time and space complexity of the algorithms for deciding $\preceq^{fin}_{del}$ and $\preceq^{\omega}_{del}$.

Theorem 5.7.7 *The delay bisimulation equivalence classes of the relation $\approx_{del}^{fin}$ can be computed in time $\mathcal{O}(nm(n+m^2))$ and space $\mathcal{O}(nm)$ and the classes of the relation $\approx_{del}^{\omega}$ can be computed in time $\mathcal{O}(nm(n+m^3))$ and space $\mathcal{O}(nm^2)$.*

In what follows, we briefly write $\approx_{del}$ to denote one of the delay bisimulation equivalences $\approx_{del}^{fin}$ and $\approx_{del}^{\omega}$; we write $\Longrightarrow$ for one of the transitions $\Longrightarrow^{fin}$ and $\Longrightarrow^{\omega}$ and $Pre(a, M)$ for one of the predecessor predicates $Pre^{str}(a, M)$ and $Pre^{\omega}(a, M)$.

As said before, the basic idea behind the computation of the delay equivalences is to adapt the partition refinement technique sketched in Figure 5.18, leading to the basic schema given in Figure 5.22: We start from the trivial partition $\chi = \{S\}$ and whenever we find a block C of χ, states $s, s' \in C$ an equivalence class M of $\equiv_\chi$ such that

$$s \stackrel{\widehat{a}}{\Longrightarrow} M \text{ but } s' \not\stackrel{\widehat{a}}{\Longrightarrow} M,$$

then we split C into two blocks, being $C \cap Pre(a, M)$ and $C \setminus Pre(a, M)$. Thus, as before, we define

$$\begin{aligned} Refine(C, a, M) &= \{C \cap Pre(a, M), C \setminus Pre(a, M)\} \setminus \{\emptyset\} \\ Refine(\chi, a, M) &= \bigcup_{C \in \chi} Refine(C, a, M). \end{aligned}$$

Recall that in the nonprobabilistic case, we dealt with $M \subseteq S$. A problem here is that the relation $\equiv_\chi$ is infinite. This can be solved by using Proposition 5.5.6, which basically states that $Pre(a, M) = Pre(a, M \cap \cup_t Steps_a(t))$. Thus, we can work with pairs $\langle a, M\rangle$ such that $a \in Act_\tau$ and $M \in \cup_t Steps_a(t)/\equiv_\chi$. Then M is a set of distributions that appear as targets of transitions in the system and that are equated by the relation $\equiv_\chi$. The pairs $\langle a, M\rangle$ are also called *step classes* induced by χ. The set of all step classes is called the *step partition* induced by χ and denoted by $\mathcal{M}_\chi$. The partition χ is in this context called a *state partition* and its blocks are called *state classes*.

For the obvious reasons of efficiency, the algorithm does not compute the step partition $\mathcal{M}_\chi$ all over again each time that χ is updated. Rather, we store $\mathcal{M}_\chi$ (in the variable $\mathcal{M}$) and update it accordingly when χ is updated. This leads to a partition refinement technique that works in two phases given in Figure 5.22. First, we update the state partition using the step classes from the latest step partition as splitters. Then, we update the step partition according to the state partition just obtained. Note that, initially, we have $\chi = \{S\}$ and $\mathcal{M}_{new} = \{\langle a, M_a\rangle \mid a \in Act_\tau\}$, where $M_a = \{\mu \mid s \xrightarrow{a} \mu\}$, which is indeed the step partition induced by $\{S\}$.

Updating $\mathcal{M}_{new}$ proceeds as follows. After updating, $\mathcal{M}$ should consist of pairs $\langle a, M\rangle$ such that for all blocks C of χ

$$\mu[C] = \mu'[C] \text{ for all } \langle a, M\rangle \in \mathcal{M}, \text{ and all distributions } \mu, \mu' \in M. \qquad (*)$$

This is achieved by replacing, for every block $C \in \chi$, each step class $\langle b, N\rangle \in \mathcal{M}$ by $\langle b, N_1\rangle$, $\langle b, N_2\rangle, \ldots,$ $\langle b, N_r\rangle$. Here, N_i collects all distributions ν in N that agree on the value $\nu[C]$. More precisely, $\{N_1, \ldots, N_r\}$ are the equivalence classes of the relation $\cong_C$ defined by $\nu_1 \cong_C \nu_2$ iff $\nu_1[C] = \nu_2[C]$. For $N/\cong_C = \{N_1, \ldots, N_r\}$ define

$$\begin{aligned} Split(N, C) &= \{N_1, \ldots, N_r\} \text{ and} \\ Split(\langle b, N\rangle, C) &= \{\langle b, N_1\rangle, \langle b, N_2\rangle, \ldots, \langle b, N_r\rangle\}. \end{aligned}$$

Thus, if we replace every step class $\langle b, N\rangle$ in $\mathcal{M}$ by $Split(\langle b, N\rangle, C)$, then $\mathcal{M}$ meets condition $(*)$ for the block C. That is, $\mathcal{M}$ is replaced by

$$Split(\mathcal{M}, C) \;=\; \bigcup_{\langle b,N\rangle \in \mathcal{M}} Split(\langle b, N\rangle, C).$$

Now, the procedure for updating $\mathcal{M}$ replaces $Split(\mathcal{M}, C)$ for every block C. It is obvious that we only have to consider blocks in $\chi_{new} \setminus \chi_{old}$ since for blocks C in $\chi_{old} \setminus \chi_{new}$, we have $Split(\mathcal{M}, C) = \mathcal{M}$.

$\chi_{new} := \{S\}$;

$\mathcal{M} := \{\langle a, M_a\rangle : a \in Act_\tau\}$ where $M_a = \bigcup_{s \in S} \; Steps_a(s)$;

Repeat

> $\chi_{old} := \chi_{new}$; $\chi_{new} := \emptyset$;
>
> (* Refine state partition w.r.t step partition*)
>
> for all $\langle a, M\rangle \in \mathcal{M}_{old}$ do $\chi_{new} := Refine(\chi_{new}, a, M)$ od;
>
> (* Refine step partition w.r.t state partition*)
>
> for all $C' \in \chi_{new} \setminus \chi_{old}$ do $\mathcal{M} := Split(\mathcal{M}, C')$ od;

until $\chi_{new} = \chi_{old}$;

Return χ_{new}.

Figure 5.22: The two–phased partitioning technique

We now describe how this two–phased partitioning algorithm can be implemented. We first give the skeleton in Figure 5.23; the most detailed algorithm is given in Figure 5.24.

The idea to merge the two refinement phases. So, rather than first performing the refine operator $Refine(C, a, M)$ for several blocks C and then performing the split operator $Split(\mathcal{M}, C')$ for several blocks C' as in Figure 5.22, the algorithm in Figure 5.23 works by performing $Refine(C, a, M)$ for a single block and immediately afterwards performing $Split(\mathcal{M}, C')$ for a block $C' \in Refine(C, a, M)$. Recall that, for a fixed splitter $\langle a, M\rangle$, the set $Refine(C, a, M)$ contains either one or two blocks. Moreover, if $Refine(C, a, M) = \{C_1, C_2\}$, we have that $Split(\langle b, N\rangle, C_1) = Split(\langle b, N\rangle, C_2)$ for all $\langle b, N\rangle \in \mathcal{M}$. This is because M contains step classes $\langle b, N\rangle$ such that $\nu_1[C] = \nu_2[C]$. Therefore, $\nu_1[C'] = \nu_2[C']$ iff $\nu_1[C \setminus C'] = \nu_2[C \setminus C']$. Finally, we remark that by definition $C_1 = C \setminus C_2$ and $C_2 = C \setminus C_1$, which immediately implies the desired result. Therefore, it suffices to perform $Split(\mathcal{M}, C')$ for only one block in C' in $Refine(C, a, M)$.

We further refine the algorithm in Figure 5.23, leading to the algorithm in Figure 5.24. Here, we organize the step partition $\mathcal{M}$ and the state partitions χ_{new} and χ_{old} by queues. We use the usual operations $head(Q)$ which returns first element of a queue Q and $tail(Q)$ removes the first element of Q (provided that Q is not empty) and $Add(Q, \alpha)$ which adds the element α at the end of Q. In the queue representing the step partition we use a special

$\chi_{new} := \{S\}$;

$\mathcal{M} := \{\langle a, M_a\rangle : a \in Act_\tau\}$ where $M_a = \bigcup_{s\in S} \ Steps_a(s)$;

Repeat

$\quad$ $\chi_{old} := \chi_{new}; \chi_{new} := \emptyset$;

$\quad$ for all $\langle a, M\rangle \in \mathcal{M}$

$\quad\quad$ for all $C \in \chi_{new}$ do

$\quad\quad\quad$ $\chi_{new} := \chi_{new} \cup Refine(C, a, M)$;

$\quad\quad\quad$ if $|Refine(C, a, M)| = 2$

$\quad\quad\quad$ then choose an element C' from $Refine(C, a, M)$;

$\quad\quad\quad\quad$ $\mathcal{M} := Split(\mathcal{M}, C')$

$\quad\quad\quad$ fi

$\quad\quad$ od

$\quad$ od

until $\chi_{new} = \chi_{old}$;

Return χ_{new}.

Figure 5.23: The two–phased partitioning technique more detailed

symbol @, which separates "old" step classes from "new" ones. More precisely, if we think of the queue $\mathcal{M}$ as a list ordered from the left to the right

$$\langle a, M\rangle \ \ldots \ \langle c, L\rangle \ @ \ \ldots \ \langle b, N\rangle \ldots$$

then all step classes $\langle b, N\rangle$ behind (on the right of) @–symbol in $\mathcal{M}$ have already been chosen as splitter candidates in the second command of the repeat–loop (the assignment $\alpha := head(\mathcal{M})$), whereas the step class $\langle c, L\rangle$ has not yet served as splitter candidate. In particular, if we split $\mathcal{M}$ in step (3.3.2) of the algorithm, then all the step classes are new and hence @ is the last element of $\mathcal{M}$. On the other hand, if @ is the first element of the list then all step classes are "old" and therefore we have for all step classes $\langle b, N\rangle$ behind @ that $\nu_1[C] = \nu_2[C]$ for all ν_1, $\nu_2 \in N$ and all blocks C in the current partition χ. Thus, when the algorithm terminates (and returns $\chi_{new} = \chi_{old}$) then in the last execution of the repeat–loop the step partition $\mathcal{M}$ has not changed and we have: if $\langle b, N\rangle$ is step class of $\mathcal{M}$ and $\nu_1, \nu_2 \in N$ then $\nu_1 \equiv_{R_\chi} \nu_2$.

In the initialization of $\mathcal{M}$, we assume a procedure as shown in Figure 5.25. Here, the assignment $\mathcal{M} := \emptyset$ stands for the operation which creates an empty queue. A more precise formulation of step (3.3.2) is given in Figure 5.26.

The splitting operator for the step classes: A naive procedure to calculate the operation $Split(N, C')$ is to compute the value $\nu[C']$ for each $\nu \in N$ and then to collect those distributions that agree on $\nu[C']$ in the same block.

Computing the delay bisimulation equivalence classes

Input: a probabilistic system $(S, Steps)$

Output: the set $S/\approx_{dbis}$ of delay bisimulation equivalence classes

Method:

$\chi_{new} := \{S\}$;
Create a queue $\mathcal{M}$ whose elements are the pairs

$$\langle a, M_a\rangle,\ a \in Act_\tau, \text{ where } M_a = \bigcup_{s\in S} Steps_a(s).$$

Insert the symbol @ at the end of $\mathcal{M}$.
Repeat
$\chi_{old} := \chi_{new}$;
$\alpha := head(\mathcal{M})$; $\mathcal{M} := tail(\mathcal{M})$; $Add(\mathcal{M}, \alpha)$;
If $\alpha = \langle a, M\rangle$ is a step class then
(1) $P := Pre(a, M)$;
(2) $\chi_{new} := \emptyset$;
(3) for all blocks $C \in \chi_{old}$ do
(3.1) If $a = \tau$ and $M \cap \{\nu \in Distr(S) : supp(\nu) \subseteq C\} \neq \emptyset$
then $C' := C$
else $C' := C \cap P$;
(3.2) If $C' = \emptyset$ or $C' = C$ then $\chi_{new} := \chi_{new} \cup \{C\}$;
(3.3) If $C' \neq \emptyset$ and $C' \neq C$ then
(3.3.1) $\chi_{new} := \chi_{new} \cup \{C', C \setminus C'\}$;
(3.3.2) Replace any $\langle b, N\rangle$ in $\mathcal{M}$ by the step classes $Split(\langle b, N\rangle, C')$ and move @ to the end of $\mathcal{M}$;
od
until $\chi_{new} = \chi_{old}$ and $\alpha = @$;
Return χ.

Figure 5.24: Computation of $\approx_{dbis}$

$\mathcal{M} := \emptyset$;

For all $a \in Act_\tau$ do $Add(\mathcal{M}, \langle a, M_a \rangle)$;

$Add(\mathcal{M}, @)$.

Figure 5.25: Initialization of $\mathcal{M}$

$\mathcal{M}_{new} := \emptyset$;

While $\mathcal{M} \neq \emptyset$ do

  $\beta := head(\mathcal{M})$;
  $\mathcal{M} := tail(\mathcal{M})$;
  If $\beta = \langle b, N \rangle$ is a step class then
    Compute the set $Split(N, C') = \{N'_1, \ldots, N'_r\}$;
    For all $i = 1, \ldots, r$ do $Add(\mathcal{M}_{new}, \langle b, N'_i \rangle)$;
  $Add(\mathcal{M}_{new}, @)$;

$\mathcal{M} := \mathcal{M}_{new}$.

Figure 5.26: Step (3.3.2): Splitting of the step partition

This procedure can be optimized using ordered trees, which makes the comparison of the distributions easier. We propose the following procedure for computing $Split(N, C')$. We construct a balanced ordered binary search tree (e.g. an AVL tree or Red/Black tree; see e.g. [CLR90]) as follows. The vertices v in this tree are the pairs $(v.prob, v.distr)$, where $v.prob \in [0, 1]$, $v.distr \subseteq N$ such that $v.distr \neq \emptyset$ and

$$v.distr = \{\nu \in N \mid \nu[C'] = v.prob\}$$

Note that there are no two different nodes v, w such that $v.prob = w.prob$. The nodes are ordered according to the value of $v.prob$, that is $v < w$ iff $w.prob < v.prob$. Let $N = \{\nu_1, \ldots, \nu_l\}$. We generate the tree by successively inserting the values $\nu_i[C']$, $i = 1, \ldots, l$. More precisely,

- We start with a tree consisting of its root v_0 where $v_0.distr = \{\nu_1\}$ and $v_0.prob = \nu_1[C']$.
- For $i = 2, \ldots, l$, we insert the value $\nu_i[B]$ into the tree (possibly creating a new node and performing the necessary rebalance operations) and the distribution ν_i into the set $v.distr$ where v is the node with $v.prob = \nu_i[C']$.

Now, the classes in $split(N, C')$ are exactly the sets $v.distr$, for v a node in the tree.

For fixed $\nu \in N$, the computation of $\nu[C']$ takes $\mathcal{O}(n)$ time. Thus, we get the values $\nu_i[C']$, $i = 1, \dots, l$, in time $\mathcal{O}(ln) = \mathcal{O}(|N|n)$. The construction of the tree takes $\mathcal{O}(|N| \log |N|))$ time. Thus, in step (3.3.2), any step class $\langle b, N \rangle$ causes the cost $\mathcal{O}(|N|n + |N| \log |N|))$. Ranging over all step classes $\langle b, N \rangle$ in $\mathcal{M}$, we obtain the time complexity $\mathcal{O}(mn + m \log m))$ for step (3.3.2).

It is now obvious how to compute $Split(\langle b, M \rangle, C)$ in the same complexity.

Complexity: We now turn to the proof of Theorem 5.7.7 and compute the worst case complexity of the algorithm of Figure 5.24. The space complexity $\mathcal{O}(nm)$ and $\mathcal{O}(nm^2)$ is clear. We now discuss the time complexity. Therefore, we first observe that, for any set sequence $\chi_1, \chi_2, \dots$ of partitions of some set S such that χ_{1+1} is finer than χ_i, we have that the number of blocks $C \in \chi_1 \cup \chi_2 \cup \dots$ is at most $2 \cdot |S| - 1$. This can be seen as follows. We can see the sequence $\chi_1, \chi_2, \dots$ as a tree, whose nodes are the blocks in the partitions and where we have and edge between $C \in \chi_k$ and $C' \in \chi_{k+1}$ iff $C \supseteq C'$. Now an easy combinatoric argument shows that the number of nodes in the tree is smaller than $2 \cdot |S| - 1$.

For this the finitary variants, we observe that the repeat–loop in Figure 5.24 is executed at most m^2 times. Thus, the computation of the predecessor predicates $Pre(a, M)$ in step (1) of the repeat–loop requires $\mathcal{O}(nm^3)$ time when we range over all iterations, because Section 5.7.3, we saw that the sets $Pre(a, M)$ can be obtained in time and space $\mathcal{O}(nm)$.

Step (3.3.2) is executed at most $\mathcal{O}(n)$ times. Any execution of step (3.3.2) calls the splitting procedure explained above which runs in time $\mathcal{O}(mn + m \log m)$. Thus, when we sum up over all executions of step (3.3.2) then we get the total cost $\mathcal{O}(n^2m + nm \log m)$. This yields the time complexity

$$\mathcal{O}(nm^3 + n^2m + nm \log m) \;=\; \mathcal{O}(nm(n + m^2))$$

for the whole algorithm. A similar argument yields the time complexity $\mathcal{O}(nm^2(n + m^2))$ for the ω–variants. For the weak variants $\approx_{dwbis}$ and $\approx^{\omega}_{dwbis}$, multiple computations of $Pre(a, M)$ (i.e. the set $Pre^{str}(a, M)$ or $Pre^{\omega}(a, M)$) in step (1) can be avoided since the weak transitions does not depend on the current state partition χ. For this, we shift the computation of $Pre(a, M)$ into step (3.3.2); i.e., whenever a new step class $\langle b, N' \rangle$ is created in step (3.3.2) then we may apply the corresponding $\mathcal{O}(nm)$ predecessor operator. As step (3.3.2) is executed at most $\mathcal{O}(n)$ times we get the time complexity $\mathcal{O}(n^2m)$ for all predecessor operations together. For any of the step classes $\langle b, N \rangle$ in $\mathcal{M}$, we have to keep a record of the sets $Pre(b, N)$; e.g. by adding the component $Pre(b, N)$ to the step classes $\langle b, N \rangle$ in $\mathcal{M}$. Since at any moment $\mathcal{M}$ contains at most m step classes this additional component for the step classes does not change the space complexity. We get:

Theorem 5.7.8 *The delay weak finitary bisimulation equivalence classes can be computed in time $\mathcal{O}(nm(n + m))$ and space $\mathcal{O}(nm)$; the delay weak ω–bisimulation in time $\mathcal{O}(nm(n + m^2))$ and space $\mathcal{O}(nm^2)$.*

Remark 5.7.9 These complexity results holds weak and branching delay bisimulations. For the weak variants $\approx_{dwbis}$ and $\approx^{\omega}_{dwbis}$ our method can be modified to get an algorithm with lower time but higher space complexities. See Theorem 5.7.8.

5.7.5 The Delay Simulation Preorder

This section presents polynomial time and space algorithms that compute the for the delay simulation preorders $\preceq_{del}^{fin}$ and $\preceq_{del}^{\omega}$. These algorithms are based on the general schema of Figure 5.19. We first formulate the main result of this section.

Theorem 5.7.10 *The delay finitary simulation preorder $\preceq_{del}^{fin}$ can be computed in time complexity $\mathcal{O}(n^5m^2/\log n)$ and space complexity $\mathcal{O}(nm + n^2 + m^2)$. The delay ω–simulation preorder $\preceq_{del}^{\omega}$ can be computed in time $\mathcal{O}(n^5m^3/\log n + n^3m^3)$ and space complexity $\mathcal{O}(nm^2 + n^2 + m^2)$.*

> $R := S \times S$;
>
> Repeat
>
> $R_{new} := R$;
>
>   For all $(s, s') \in R$ do
>
>   For all $(a, \mu) \in Steps(s)$ do
>
> $sim := true$;
>
>   If $(a \neq \tau \vee \mu \not\sqsubseteq_R \mu^1_{s'}) \wedge s' \notin Pre(a, \mu\uparrow_R)$ then $sim := false$;
>
>   If $\neg sim$ then $R_{new} := R_{new} \setminus \{(s, s')\}$;
>
> until $R = R_{new}$;
>
> Return R.

Figure 5.27: Basic idea for computing the delay simulation preorder

In the sequel, $\preceq_{del}$ denotes one of the delay simulation preorders $\preceq_{del}^{fin}$ or $\preceq_{del}^{\omega}$. As before, we briefly write $\Longrightarrow$ to denote one of the transitions $\Longrightarrow^{fin}$ and $\Longrightarrow^{\omega}$ and $Pre(a, M)$ for one of the predecessor predicates $Pre^{str}(a, M)$ and $Pre^{\omega}(a, M)$.

The remainder of this section is concerned with algorithms for computing the relation $\preceq_{del}$ with the time and space complexity given above. The basic idea of the probabilistic variant of the general schema in Figure 5.19 is sketched in Figure‘5.27.

A basic operation in the algorithm is the predicate $\sqsubseteq_R \subseteq S \times S$ defined by

$$s \sqsubseteq_R s' \text{ iff } s \xrightarrow{a} \mu \text{ implies } s' \stackrel{\widehat{a}}{\Longrightarrow} \mu\uparrow_R.$$

Then Proposition 5.6.3 implies that the relation $\preceq_{del}$ is the largest relation R such that $(s, s') \in R \implies s \sqsubseteq s'$. Hence, we can compute $\preceq_{del}$ as follows. We start with the trivial preorder $R = S \times S$. Then, we test whether $s \sqsubseteq_R s'$ for all pairs $(s, s') \in R$. If $s \sqsubseteq_R s'$ then we keep the pair (s, s') in R_{new}, otherwise, we remove it from R_{new}.

The test whether $s \sqsubseteq_R s'$ is simply done by checking for all outgoing transitions $s \xrightarrow{a} \mu$ of s whether there exists a matching delay transition $s' \stackrel{\widehat{a}}{\Longrightarrow} \mu\uparrow_R$; in other words, whether $s' \in Pre(a, \mu\uparrow_R)$. We treat the case $a = \tau$ and $\mu \sqsubseteq_R \{s' \mapsto 1\}$, in which case we do have a matching delay transition from s' separately, because this condition can be checked easily. [13] The problem with the set $Pre(a, \mu\uparrow_R)$ (for $a \neq \tau$ and $\mu \not\sqsubseteq_R \mu^1_{s'}$)is that the set $\mu\uparrow_R$ is infinite. As before, we use the fact that $Pre(a, \mu\uparrow_R) = Pre(a, \mu\uparrow_R \cap \cup_s Steps_a(s))$ to obtain a finite representation. The computation of $Pre(a, M)$ has been explained in Section 5.7.3.

The computation of $\mu\uparrow_R \cap \cup_s Steps_a(s)$ is done as follows. Let $Steps[a]$ denote the set $\cup_s Steps_a(s)$. Whenever we have a decreasing sequence $R_0 \supseteq R_1 \supseteq R_2 \supseteq \ldots$ of binary relations on the state space S then $\mu \not\sqsubseteq_{R_i} \mu'$ implies $\mu \not\sqsubseteq_{R_j} \mu'$ for all $j \geq i$. Thus, if we have detected that $\mu \not\sqsubseteq_R \mu'$ for a certain distribution $\mu' \in Steps[a]$ and the current relation R then we also know that $\mu \not\sqsubseteq_R \mu'$ in all further iterations. This observation motivates the idea to keep a record of the sets

$$Sim[a, \mu] \; = \; \{\mu' \in Steps[a] : \mu \sqsubseteq_R \mu'\}\,.$$

Then, to check whether $s' \in Pre(a, \mu\uparrow_R \cap [\xrightarrow{a}])$, we calculate the set $Pre(a, Sim[a, \mu])$ with the $\mathcal{O}(nm)$–method described in Section 5.7.4 and get:

$$s \sqsubseteq_R s' \text{ iff } s' \in Pre(s, Sim[a, \mu]) \text{ for all } s \xrightarrow{a} \mu \text{ where } a \neq \tau \text{ or } \mu \not\sqsubseteq_R \mu^1_{s'}.$$

Moreover, instead of the repeat–loop (where we distinguish between R_{new} and the relation R obtained by the previous iteration) we may do all modifications (that is, removements of certain pairs (s, s')) on the current relation R. This requires the recomputation of $Sim[a, \mu]$ whenever a pair (s, s') is removed from R; e.g. with the method sketched in Figure 5.28. Here, we make use of the observation that

$$\text{if } \mu \sqsubseteq_R \mu' \text{ and } s \notin supp(\mu) \text{ or } s' \notin supp(\mu') \text{ then } \mu \sqsubseteq_{R'} \mu' \text{ where } R' = R \setminus \{(s, s')\}.$$

For the test whether $\mu \sqsubseteq_R \mu'$ we may apply the network–based method mentioned in [BEM00].

For all $a \in Act_\tau$ and $\mu \in Steps[a]$ where $s \in supp(\mu)$ do

  $Sim_{new}[a, \mu] := \emptyset$;

  For all $\mu' \in Sim[a, \mu]$ where $s' \in supp(\mu')$ do

    If $\mu \sqsubseteq_R \mu'$ then $Sim_{new}[a, \mu] := Sim_{new}[a, \mu] \cup \{\mu'\}$;

  $Sim[a, \mu] := Sim_{new}[a, \mu]$;

Figure 5.28: Recomputation of $Sim[a, \mu]$ after the removement of (s, s') from R

These ideas lead to the algorithm shown in Figure 5.30. We organize R as a queue where the ordering in the initial queue is arbitrary.[14] Moreover, we use a special variable *last* which

[13] If one adds the distributions μ^1_s to $Steps_\tau(s)$ at the beginning of the algorithm then the τ–labeled transitions $s \xrightarrow{\tau} \mu$ with $\mu \sqsubseteq_R \mu^1_{s'}$ do not require a special treatment. These additional transitions does not change the simulation preorder and do not affect the complexity of our method. However, the test whether $\mu \sqsubseteq_R \mu^1_{s'}$ is possible without the (quite complicate) network–based methods. We just have to check whether $(t, s') \in R$ for all $t \in supp(t)$. This can be done in time $\mathcal{O}(n)$ while the computation of the maximum flow in $\mathcal{N}(\mu, \mu^1_{s'}, R)$ takes $\mathcal{O}(n^3/\log n)$ time.

[14] Of course, in the initialization of R the "trivial" pairs (s, s) can be ignored (since we always have $s \preceq_{dsim} s$). However, including the pairs (s, s) does not affect the asymptotic time or space complexity.

is either undefined (i.e. $last = \bot$) or an element (t_0, t_0') of R. Initially, $last$ is the last element of R. In the case where $last = (t_0, t_0')$ then none pair (t, t') in R that have been investigated after the last investigation of (t_0, t_0') was removed from R. (Here, by an *investigation* of an element (s, s') of R, we mean the execution of the steps (2)–(4) where (s, s') is the chosen element in step (2).) I.e. we have $t \sqsubseteq_R t'$ for all pairs (t, t') that are behind (on the right of) of (t_0, t_0') in R (see Figure 5.29). In step (3), we test whether $s \sqsubseteq_R s'$. The meaning of the

$$R: \quad \underbrace{(s, s')}_{head(R)} \quad \ldots \quad \underbrace{\underbrace{(t_0, t_0')}_{=last} \quad \ldots (t, t') \ldots}_{t \sqsubseteq_R t'}$$

Figure 5.29: R organized as a queue

boolean variable sim in step (4) is as before, i.e. it records whether or not $s \sqsubseteq_R s'$. Thus, when we get $s \sqsubseteq_R s'$ for the current pair (s, s') (i.e. the boolean variable sim is true in line (4)) then we reinsert (s, s') into R and distinguish two cases:

1. If $last = (s, s')$ then $t \sqsubseteq_R t'$ for simulation preorder $\preceq_{dsim}$ and our algorithms terminates (see the "then"–branch of step (4.2)).

2. If $(s, s') \neq last$ then we are in the situation of Figure 5.29 where $last = (t_0, t_0') \neq (s, s')$. We just move (s, s') to the end of R (via the command $Add(R, (s, s'))$ in step (4.1)) and investigate the next pair in R (see step (4.2.2) where we jump back to step (2)) but still deal with $last = (t_0, t_0')$.

If $s \not\sqsubseteq_R s'$ then we put $last = \bot$ (in step (4.3)) and "remove"(i.e. do not reinsert) the current pair (s, s') from R. In this case, the induced relations $\sqsubseteq_R$ on $Distr(S)$ and S change and the condition $t \sqsubseteq_R t'$ might be violated for some pair in R. Thus, we recompute $Sim[a, \mu]$ (in step (4.4)) and "wait" for the next pair (t_0, t_0') that will be not removed from R (see step (4.2.1)).

Whenever we start an investigation of the first pair (s, s') of R in step (2) then the set $Sim[a, \mu]$ contains exactly those distributions $\mu' \in Steps[a]$ where $\mu \sqsubseteq_R \mu'$. I.e. $Sim[a, \mu] = \mu \uparrow_R \cap Steps[a]$. This can be seen as follows. Initially, we deal with $Sim[a, \mu] = Steps[a]$ and $R = S \times S$. Thus, $\mu \sqsubseteq_R \mu'$ for all distributions μ, μ' on S. Whenever we remove a pair (s, s') from R (more precisely, we take (s, s') in step (2), obtain $s \not\sqsubseteq_R s'$ in step (3) but do not add (s, s') to the end of R) then we recalculate the sets $Sim[a, \mu]$ in step (4.4) (where we call a procedure as sketched in Figure 5.28).

Complexity: We first consider the space complexity. $\mathcal{O}(nm)$ space is needed for the representation of the probabilistic system itself. The representation of R and the networks $\mathcal{N}(\mu, \mu', R)$ require $\mathcal{O}(n^2)$ space while we need $\mathcal{O}(m^2)$ space for the sets $Sim[a, \mu]$. This yields the space complexity $\mathcal{O}(nm + n^2 + m^2)$ for the whole algorithm.

Next we discuss the time complexity. For the finitary case, we observe that steps (2)–(4) are executed at most n^2 times. Each execution of step (3) takes $\mathcal{O}(nm^2)$ time.[15] Ranging

[15] Here, we use the observation that the condition $\mu \not\sqsubseteq_R \mu_{s'}^1$ is equivalent to $(t, s') \notin R$ for some $t \in supp(\mu)$ which can be checked in linear time. To test whether $s' \in Pre(a, Sim[a, \mu])$ requires $\mathcal{O}(nm)$ time (see Proposition 5.7.6). Thus, given (s, s'), any transition $s \xrightarrow{a} \mu$ causes the cost $\mathcal{O}(nm)$. Ranging over all outgoing transitions from s, we get the time complexity $\mathcal{O}(nm^2)$ for step (3).

over all executions, step (3) requires $\mathcal{O}(n^3m^2)$ time. In step (4), only the time complexity for the recomputation of $Sim[a,\mu]$ in step (4.4) is of interest. In each execution of step (4.4), we check whether $\mu \sqsubseteq_R \mu'$ for at most $\mathcal{O}(m^2)$ pairs (μ,μ'). Using the maximum flow algorithm proposed by [CHM90], we get the time complexity $\mathcal{O}(m^2n^3/\log n)$ for each execution of step (4.4). Summing up over all pairs (s,s') that are removed from R we get the total cost $\mathcal{O}(m^2n^5/\log n)$ for step (4.4). This yields the worst case time complexity $\mathcal{O}(n^5m^2/\log n)$ for the whole computation of finitary delay bisimulation. For the ω variants we obtain $\mathcal{O}(n^5m^2/\log n + nm^2)$.

Computing the delay simulation preorder

Input: a probabilistic system (S, Steps)

Output: the delay simulation preorder $\preceq_{dsim}$

Method:

(1) [Initialization]

(1.1) Create a queue R whose elements are the pairs $(s, s') \in S \times S$;

(1.2) Let last be the last element of R;

(1.3) For any $a \in \mathit{Act}_\tau$ let $\mathit{Steps}[a] := \bigcup_{t \in S} \mathit{Steps}_a(t)$;

(1.4) For any $a \in \mathit{Act}_\tau$ and $\mu \in \mathit{Steps}[a]$ let $\mathit{Sim}[a, \mu] := \mathit{Steps}[a]$;

(2) $(s, s') := \mathit{head}(R)$;

$R := \mathit{tail}(R)$;

(3) $\mathit{sim} := \mathit{true}$; For all $(a, \mu) \in \mathit{Steps}(s)$ do

If $(a \neq \tau) \vee (\mu \not\sqsubseteq_R \mu^1_{s'}) \ \wedge \ s' \notin \mathit{Pre}(a, \mathit{Sim}[a, \mu])$ then $\mathit{sim} := \mathit{false}$;

(4) If sim then

(4.1) $\mathit{Add}(R, (s, s'))$;

(4.2) If $\mathit{last} = (s, s')$

then go to (5)

else $(*\ s \sqsubseteq_R s'$ and $\mathit{last} \neq (s, s')\ *)$

(4.2.1) If $\mathit{last} = \bot$ then $\mathit{last} := (s, s')$ fi;

(4.2.2) Go to (2)

fi

else $(*\ \neg \mathit{sim}$, i.e. $s \not\sqsubseteq_R s'\ *)$

(4.3) $\mathit{last} := \bot$

(4.4) For all $a \in \mathit{Act}_\tau$ and $\mu \in \mathit{Steps}[a]$ put

$$\mathit{Sim}[a, \mu] := \{\mu' \in \mathit{Sim}[a, \mu] : \mu \sqsubseteq_R \mu'\};$$

(4.5) go to (2);

fi

(5) Return R.

Figure 5.30: Algorithm for computing the delay simulation preorder

5.8 Conclusions

We have studied various notions of simulation and bisimulation for probabilistic systems. First we have recalled several variants of strong and weak (bi-)simulation, most of which are known from literature, and studied their properties. An important conclusion is that the weak bisimulations based on deterministic adversaries are not useful in verification, since they fail the desired soundness and compositionality results. The randomized variants, however, are sound for trace distribution inclusion and compositional w.r.t. parallel composition.

Moreover, we introduced two novel notions of (bi-)simulation equivalence that abstract from internal computations. We presented polynomial-time algorithms that compute the quotient spaces and proved the desired soundness and compositionality results. Thus, our notion of bisimulation equivalence could yield an alternative to the weak bisimulations of [SL95]. Although the equivalences à la [SL95] are the natural probabilistic counterpart to weak bisimulation equivalence in LTSs [Mil80, Par81, GW89], their definitions are rather complicated and the decidability is still an open problem. We argue that the definitions of our equivalences – which rely on the rather intuitive concept of norm functions à la [GV98] – are comparatively simple. Moreover, the use of norm functions in the definition of our equivalences allows for a characterization of the equivalence classes by means of graph-theoretical criteria which served as basis for our algorithm that computes the equivalence classes. In particular, the characterization of the delay predecessor predicates from 5.7.1 can easily be rewritten as terms of the relational mu-calculus.

Of course, a lot of work still has to be done: the algorithms have to be implemented and case studies have to be carried out in order to estimate their usefulness. Furthermore, we believe that the complexity bounds we derived are rather coarse and serious gains can be obtained by adapting the algorithms.

Furthermore, norm functions and the derived notions of bisimulations can also be defined for infinite systems.[16] We believe that, as in the nonprobabilistic case, in many applications, it is quite simple to "guess" a norm function and then to check (e.g. by hand or by theorem proving) whether it fulfills the necessary conditions.

Acknowledgements The authors would like to thank Frits Vaandrager and Holger Hermanns for many helpful comments.

[16] For our purposes, it was sufficient to consider the natural numbers as range of the norm functions. The framework of [GV98] also covers infinite, possibly uncountable, state spaces and allows for arbitrary well–founded sets as range of norm functions. We argue that these ideas can also be used to handle PLTSs of arbitrary size.

CHAPTER 6

Linear Parametric Model Checking of Timed Automata

Time present and time past
Are both perhaps present in time future
time future contained in time past.
If all time is eternally present
All time is unredeemable.
What might have been is an abstraction
Remaining a perpetual possibility
Only in a world of speculation.
What might have been and what has been
Point to one end, which is always present.
Footfalls echo in the memory
Down the passage which we did not take
Towards the door we never opened
Into the rose-garden. My words echo
Thus, in your mind.

T. S. Eliot, *Four Quartets*

Abstract We present an extension of the model checker Uppaal, capable of synthesizing linear parameter constraints for the correctness of parametric timed automata. A symbolic representation of the (parametric) state space in terms of parametric difference bound matrices is shown to be correct. A second contribution of this chapter is the identification of a subclass of parametric timed automata (L/U automata), for which the emptiness problem is decidable, contrary to the full class where it is known to be undecidable. Also, we present a number of lemmas that reduce the verification effort for L/U automata in certain cases. We illustrate our approach by deriving linear parameter constraints for a number of well-known case studies from the literature (exhibiting a flaw in a published paper).

6.1 Introduction

Model checking is emerging as a practical tool for automated debugging of complex reactive systems such as embedded controllers and network protocols. In model checking, a high-level description of a system is compared against a logical correctness requirement to discover inconsistencies. Traditional techniques for model checking do not admit an explicit modeling of time, and are thus unsuitable for analysis of real-time systems whose correctness depends on relative magnitudes of different delays. Consequently, Alur and Dill [AD94] proposed timed automata as a formal notion to model the behavior of real-time systems. Timed automata are state-transition diagrams annotated with timing constraints using finitely many real-valued clock variables. During the last decade, there has been enormous progress in the area of timed model checking. We refer to [Alu98, CGP99, LPY97, Yov98] for overviews of the underlying theory and references to applications. Timed automata tools such as Uppaal [LPY97], Kronos [BDM+98], and PMC [LTA98] are now routinely used for industrial case studies.

A disadvantage of the traditional approaches is, however, that they can only be used to verify concrete timing properties: one has to provide the values of all timing parameters that occur in the system. Typical examples of such parameters are upper and lower bounds on computation times, message delays and timeouts. For practical purposes, one is often interested in deriving the (symbolic) constraints on the parameters that ensure correctness. The process of manually finding and proving such results is very time consuming and error prone (we have discovered minor errors in the two examples we have been looking at). Therefore tool support for deriving the constraints *automatically* is very important.

In this chapter, we study a parametric extension of timed automata, as well as a corresponding extension of the (forward) state space exploration algorithm for timed automata. We show the theoretical correctness of our approach, and its feasibility by application to two non-trivial case studies. For this purpose, we have implemented a prototype extension of Uppaal, an efficient real-time model checking tool [LPY97]. The algorithm we propose and have implemented fundamentally relies on parametric difference bound matrices (PDBMs) and operations on these. PDBMs constitute a data type that extends the difference bound matrices (DBMs, [Dil98]) in a natural way. The latter are used for recording clock differences when model checking (non-parametric) timed automata. PDBMs are basically DBMs, where the matrix entries are parameter expressions rather than constants. Our algorithm is a semi-decision algorithm which will not terminate in all cases. In [AHV93], the problem of synthesizing values for parameters such that a property is satisfied was shown to be undecidable, so this is the best we can hope for.

A second contribution of this chapter is the identification of a subclass of parametric timed automata, called *lower bound/upper bound (L/U) automata*, which appears to be sufficiently expressive from a practical perspective, while it also has nice theoretical properties. Most importantly, we show that the emptiness problem, in [AHV93] shown to be undecidable for parametric timed automata, is decidable for L/U automata. We also establish a number of lemmas which allow one to reduce the number of parameters when tackling specific verification questions for L/U automata. The application of these lemmas has already reduced the verification effort drastically in several of our experiments.

Related work There are currently several other tools available that can do parametric model checking, namely LPMC, HyTech and TReX.

LPMC [LTA98] is a parametric extension of the timed model checker PMC [BST00]. The model checking algorithm implemented in LPMC differs from ours: it represents the state space of a system as an unstructured set of constraints, whereas we use PDBMs. Moreover, LPMC implements a partition refinement technique, whereas we use forward reachability. Other differences with our approach are that LPMC also allows for the comparison of non-clock variables to parameter constraints and for more general specification properties (full TCTL with fairness assumptions).

The model checker HyTech [HHW97] is a tool for linear hybrid automata. These are more general than parametric timed automata, since they allow the modeling of continuous behavior via linear differential equations. The HyTech implementation uses polyhedra as its basic data type. It can explore the state space by using either forward reachability, as we do, or partition refinement, as in LPMC. The tool has been applied successfully to relatively small examples such as a railway gate controller. Experience so far has shown that HyTech cannot cope with larger examples, such as the ones considered in this chapter, see [CS01].

The tool TReX [AAB00, ABS01] is currently the only one that can deal with non-linear parameter constraints. Moreover, TReX has a clever method for guessing the effect of control loops in a model, based on widening principles, which increases chances of termination. Independently, [AAB00] developed the same data structure as we did (PDBMs) and implemented some similar operations on these. However, the details of this were not worked out in the research literature. Hence, we believe that our contribution over [AAB00] consists of the following. Our work presents an extensive elaboration of the theory behind our implementation. In particular, we present a correctness proof of the model checking algorithm we implemented. That is, we prove that the symbolic semantics in terms of PDBMs is sound and complete for the concrete semantics in terms of single states and transitions. These proofs rely on a number of non-trivial generalizations of results for DBMs.

Each of the tools above has been applied to the IEEE 1394 Root Contention Protocol [CS01, BST00]. In Chapter 7, section 7.3, we present a comparison of the results of these case studies. An important outcome was that each of the verification approaches has it own merits, where our verification was the fastest. Recent experiments have shown that this also holds if we generate the constraints, rather than providing them to the tool and checking their correctness.

Organization of the chapter

The remainder of this chapter is organized as follows. Section 6.2 introduces the notion of parametric timed automata. Section 6.3 gives the symbolic semantics in terms of PDBMs and is the basis for the model checking algorithm presented in Section 6.3.5. In Section 6.4, we introduce the class of L/U automata. Section 6.5 reports on several experiments with our tool. Finally, Section 6.6 presents some conclusions.

Acknowledgements We thank the reviewers for their reports, in particular Reviewer 3 who gave many comments that helped us to improve our paper and pointed out the necessity of imposing nonnegative lower bounds on clocks in constrained PDBMs.

6.2 Parametric Timed Automata

Parametric timed automata were first defined in [AHV93]. They generalize the timed automata of [AD94]. The definition of parametric timed automata that we present in this section is very similar to the definition in [AHV93], except that progress is ensured via location invariants rather than via accepting states. This difference is not essential.

6.2.1 Parameters and Constraints

Throughout this chapter, we assume a fixed set of *parameters* $P = \{p_1, \ldots, p_n\}$.

Definition 6.2.1 (Constraints) A *linear expression* e is either an expression of the form $t_1 p_1 + \cdots + t_n p_n + t_0$, where $t_0, \ldots, t_n \in \mathbb{Z}$, or ∞. We write E to denote the set of all linear expressions. A *constraint* is an inequality of the form $e \sim e'$, with e, e' linear expressions and $\sim \in \{<, \leq, >, \geq\}$. The *negation* of constraint c, denoted $\neg c$, is obtained by replacing relation symbols $<, \leq, >, \geq$ by $\geq, >, \leq, <$, respectively. A *(parameter) valuation* is a function $v : P \rightarrow \mathbb{R}^{\geq 0}$ assigning a nonnegative real value to each parameter. There is a one-to-one correspondence between valuations and points in $(\mathbb{R}^{\geq 0})^n$. In fact we often identify a valuation v with the point $(v(p_1), \ldots, v(p_n)) \in (\mathbb{R}^{\geq 0})^n$.

If e is a linear expression and v is a valuation, then $e[v]$ denotes the expression obtained by replacing each parameter p in e with $v(p)$. Likewise, we define $c[v]$ for c a constraint. Valuation v *satisfies* constraint c, denoted $v \models c$, if $c[v]$ evaluates to true. The *semantics* of a constraint c, denoted $[\![c]\!]$, is the set of valuations that satisfy c. A finite set of constraints C is called a *constraint set*. A valuation *satisfies* a constraint set if it satisfies each constraint in the set. The *semantics* of a constraint set C is given by $[\![C]\!] := \bigcap_{c \in C} [\![c]\!]$. We say that C is *satisfiable* if $[\![C]\!]$ is nonempty.

Constraint c *covers* constraint set C, denoted $C \models c$, iff $[\![C]\!] \subseteq [\![c]\!]$. Constraint set C *is split by* constraint c iff neither $C \models c$ nor $C \models \neg c$.

During the analysis questions arise of the kind: given a constraint set C and a constraint c, does c hold, i.e., does constraint c cover C? There are three possible answers to this, *yes*, *no*, and *split*. A split occurs when c holds for some valuations in the semantics of C and $\neg c$ holds for some other valuations. Here will not discuss in detail methods for answering such questions: in the remainder of this paper we just assume the presence of the following "oracle" function.

Definition 6.2.2 (Oracle)

$$\mathcal{O}(c, C) = \begin{cases} \text{yes} & \textit{if } C \models c \\ \text{no} & \textit{if } C \models \neg c \\ \text{split} & \textit{otherwise} \end{cases}$$

The oracle function can be computed in polynomial time using linear programming (LP) solvers. Suppose we want to compute $\mathcal{O}(c, C)$, where c takes the form $e \leq e'$. Then we first maximize the linear function $e' - e$ subject to the set C of linear inequalities. This is a linear programming problem. If the outcome is negative then $\mathcal{O}(c, C) = \text{no}$. Otherwise we maximize $e - e'$ subject to C. If the outcome is less than or equal to 0 then $\mathcal{O}(c, C) = \text{yes}$. Otherwise $\mathcal{O}(c, C) = \text{split}$. In our implementation we use an LP solver that was kindly provided to us by the authors of [BST00], who built it for their LPMC model checking tool. This LP solver is geared to perform well on small, simple sets of constraints rather than large, complicated ones.

Observe that using the oracle, we can easily decide semantic inclusion between constraint sets: $[\![C]\!] \subseteq [\![C']\!]$ iff $\forall c' \in C' : \mathcal{O}(c', C) = \text{yes}$.

6.2.2 Parametric Timed Automata

Throughout this chapter, we assume a fixed set of clocks $X = \{x_0, \ldots, x_m\}$ and a fixed set of actions $A = \{a_1, \ldots, a_k\}$. The special clock x_0, which is called the *zero clock*, always has the value 0 (and hence does not increase with time).

A *simple guard* is an expression f of the form $x_i - x_j \prec e$, where x_i, x_j are clocks, $\prec \in \{<, \leq\}$, and e is a linear expression. We say that f is *proper* if $i \neq j$. We define a *guard* to be a (finite) conjunction of simple guards. We let g range over guards and write G to denote the set of guards. A *clock valuation* is a function $w : X \to \mathbb{R}^{\geq 0}$ assigning a nonnegative real value to each clock, such that $w(x_0) = 0$. We will identify a clock valuation w with the point $(w(x_0), \ldots, w(x_m)) \in (\mathbb{R}^{\geq 0})^{m+1}$. Let g be a guard, v a parameter valuation, and w a clock valuation. Then $g[v, w]$ denotes the expression obtained by replacing each parameter p with $v(p)$, and each clock x with $w(x)$. A pair (v, w) of a parameter valuation and a clock valuation *satisfies* a guard g, denoted $(v, w) \models g$, if $g[v, w]$ evaluates to true. The *semantics* of a guard g, denoted $[\![g]\!]$, is the set of pairs (v, w) such that $(v, w) \models g$. Given a parameter valuation v, we write $[\![g]\!]_v$ for the set of clock valuations $\{w \mid (v, w) \models g\}$.

A *reset* is an expression of the form, $x_i := b$ where $i \neq 0$ and $b \in \mathsf{N}$. A *reset set* is a set of resets containing at most one reset for each clock. The set of reset sets is denoted by R.

We now define an extension of timed automata [AD94, Yov98] called parametric timed automata. Similar models have been presented in [AHV93, AAB00, BST00].

Definition 6.2.3 (PTA) A *parametric timed automaton (PTA)* over set of clocks X, set of actions A, and set of parameters P, is a quadruple $\mathcal{A} = (Q, q_0, \to, I)$, where Q is a finite set of *locations*, $q_0 \in Q$ is the *initial location*, $\to \subseteq Q \times A \times G \times R \times Q$ is a finite *transition relation*, and function $I : Q \to G$ assigns an *invariant* to each location. We abbreviate a $(q, a, g, r, q') \in \to$ consisting of a source location q, an action a, a guard g, a reset set r, and a target location q' as $q \xrightarrow{a,g,r} q'$. For a simple guard $x_i - x_j \prec e$ to be used in an invariant it must be the case that $j = 0$, that is, the simple guard represents an upper bound on a clock.[1]

Example 6.2.4 A parametric timed automaton with clocks x, y and parameters p, q can be seen in Fig. 6.1. The initial location is $S0$ and has invariant $x \leq p$. There is a transition from

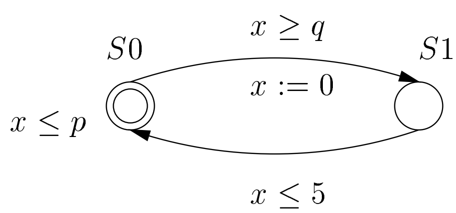


Figure 6.1: A parametric timed automaton

the initial location to $S1$, which has guard $y \geq q$ and reset set $\{x := 0\}$. There are no actions on the transitions. The transition from $S0$ to $S1$ can only become enabled if $p \leq q$, otherwise the system will end up in a deadlock.

To define the semantics of PTAs, we require two auxiliary operations on clock valuations. For clock valuation w and nonnegative real number d, $w + d$ is the clock valuation that adds to each clock (except x_0) a delay d. For clock valuation w and reset set r, $w[r]$ is the clock

[1] There is no fundamental reason to impose this restriction on invariants; our whole theory can be developed without it. However, technically the restriction makes our lives a bit easier, see for instance Proposition 6.3.17. In practice the condition is (as far as we are aware) always met.

valuation that resets clocks according to r.

$$(w+d)(x) = \begin{cases} 0 & \text{if } x = x_0 \\ w(x)+d & \text{otherwise} \end{cases} \qquad (w[r])(x) = \begin{cases} b & \text{if } x := b \in r \\ w(x) & \text{otherwise.} \end{cases}$$

Definition 6.2.5 (LTS) A *labeled transition system (LTS)* over a set of symbols Σ is a triple $Ł = (S, S_0, \rightarrow)$, with S a set of *states*, $S_0 \subseteq S$ a set of *initial states*, and $\rightarrow \subseteq S \times \Sigma \times S$ a *transition relation*. We write $s \xrightarrow{a} s'$ for $(s, a, s') \in \rightarrow$. A *run* of Ł is a finite alternating sequence $s_0 a_1 s_1 a_2 \cdots s_n$ of states $s_i \in S$ and symbols $a_i \in \Sigma$ such that $s_0 \in S_0$ and, for all $i < n$, $s_i \xrightarrow{a_{i+1}} s_{i+1}$. A state is *reachable* if it is the last state of some run.

Definition 6.2.6 (Concrete semantics) Let $\mathcal{A} = (Q, q_0, \rightarrow, I)$ be a PTA and v be a parameter valuation. The *concrete semantics of $\mathcal{A}$ under v*, denoted $[\![\mathcal{A}]\!]_v$, is the labeled transition system (LTS) $(S, S_0, \rightarrow)$ over $A \cup \mathbb{R}^{\geq 0}$ where

$$\begin{aligned} S &= \{(q, w) \in Q \times (X \rightarrow \mathbb{R}^{\geq 0}) \mid w(x_0) = 0 \wedge (v, w) \models I(q)\}, \\ S_0 &= \{(q, w) \in S \mid q = q_0 \wedge w = \lambda x.0\}, \end{aligned}$$

and transition predicate $\rightarrow$ is specified by the following two rules, for all (q, w), $(q', w') \in S$, $d \geq 0$ and $a \in A$,

- $(q, w) \xrightarrow{d} (q', w')$ if $q = q'$ and $w' = w + d$.
- $(q, w) \xrightarrow{a} (q', w')$ if $\exists g, r : q \xrightarrow{a,g,r} q'$ and $(v, w) \models g$ and $w' = w[r]$.

Note that the LTS $[\![\mathcal{A}]\!]_v$ has at most one initial state. It has no initial state if the invariant which the initial location must satisfy is unsatisfiable.

6.2.3 The Parametric Model Checking Problem

In its current version, Uppaal is able to check for reachability properties, in particular whether certain combinations of locations and constraints on clock variables are reachable from the initial configuration. Our parametric extension of Uppaal handles exactly the same properties. However, rather than just telling whether a property holds or not, our tool looks for constraints on the parameters which ensure that the property holds.

Definition 6.2.7 (Properties) Let $\mathcal{A} = (Q, q_0, \rightarrow, I)$ be a PTA. The sets of *system properties* and *state formulas* for $\mathcal{A}$ are defined by, respectively,

$$\psi ::= \forall\Box\phi \mid \exists\Diamond\phi \qquad\qquad \phi ::= x - y \prec b \mid q \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi$$

where $x, y \in X$, $b \in \mathbb{N}$ and $q \in Q$. Let $\mathcal{A}$ be a PTA, v a parameter valuation, s a state of $[\![\mathcal{A}]\!]_v$, and ϕ a state formula. We write $s \models_v \phi$ if ϕ holds in state s of $[\![\mathcal{A}]\!]_v$, we write $[\![\mathcal{A}]\!]_v \models \forall\Box\phi$ if ϕ holds in all reachable states of $[\![\mathcal{A}]\!]_v$, and we write $[\![\mathcal{A}]\!]_v \models \exists\Diamond\phi$ if ϕ holds for some reachable state of $[\![\mathcal{A}]\!]_v$.

The problem that we address in this chapter can now be stated as follows:

> *Given a parametric timed automaton $\mathcal{A}$ and a system property ψ, compute the set of parameter valuations v for which $[\![\mathcal{A}]\!]_v \models \psi$.*

Remark 6.2.8 Timed automata [AD94, Yov98] arise as a special case of PTAs for which the set P of parameters is empty. If $\mathcal{A}$ is a PTA and v is a parameter valuation, then the structure $\mathcal{A}[v]$ that is obtained by replacing all linear expressions e that occur in $\mathcal{A}$ by $e[v]$ is a timed automaton.[2] It is easy to see that in general $[\![\mathcal{A}]\!]_v = [\![\mathcal{A}[v]]\!]$. Since the reachability problem for timed automata is decidable [AD94], this implies that, for any $\mathcal{A}$, integer valued v and ψ, $[\![\mathcal{A}]\!]_v \models \psi$ is decidable.

6.2.4 Example: Fischer's Mutual Exclusion Algorithm

Figure 6.3 shows a PTA model of Fischer's mutual exclusion algorithm [Lam87]. The purpose of this algorithm is to guarantee mutually exclusive access to a critical section among competing processes P_1, $P_2, \ldots P_n$. In the algorithm, a shared variable lock is used for communication between the processes, with each process P_i running the code of Figure 6.2.

FISCHER (P_i)
$\mathsf{lock} := 0$
repeat
  while $\mathsf{lock} \neq 0$ **do skip od**
  $\mathsf{lock} := i$
  delay
until $\mathsf{lock} = i$
critical section
$\mathsf{lock} := 0$

Figure 6.2: Fischer's mutual exclusion algorithm

The correctness of this algorithm crucially depends on the timing of the operations. The key idea for the correctness is that any process P_i that sets $\mathsf{lock} := i$ is made to wait long enough before checking $\mathsf{lock} = i$ to ensure that any other process P_j that tested $\mathsf{lock} = 0$, before P_i set lock to its index, has already set lock to its index j, when P_i finally checks $\mathsf{lock} = i$.

Assume that read/write access to the global variable (in the operations $\mathsf{lock} = i$ and $\mathsf{lock} := 0$) takes between min_rw and max_rw time units and assume that the delay operation (including the time needed for the assignment $\mathsf{lock} := i$) takes between min_delay and max_delay time units. If we assume the basic parameter constraint $0 \leq min\_rw < max\_rw \wedge 0 \leq min\_delay < max\_delay$, then mutual exclusion is guaranteed if and only if $max\_rw \leq min\_delay$.

Now consider the PTA in Fig. 6.3. (Several different models of this algorithm exist [AL92, AHV93, Lyn96, KLL+97]; our model is closest to the one in [Lyn96].) It consists of four locations $start$ (which is initial), set, try_enter and cs; four parameters, min_rw, max_rw, min_delay and max_delay; one clock x and a shared variable $lock$. By convention, x and $lock$ are initially 0. Note that the process can remain in the locations $start$ and set for at least min_rw and strictly less than max_rw time units. Similarly, the process can remain in try_enter for a time in the interval $[min\_delay, max\_delay)$.

The shared variable, which is not a part of the definition of PTAs, is syntactic sugar which allows for an efficient encoding of the algorithm as a PTA. Also the notion of parallel

[2] Strictly speaking, $\mathcal{A}[v]$ is only a timed automaton if v assigns an integer to each parameter.

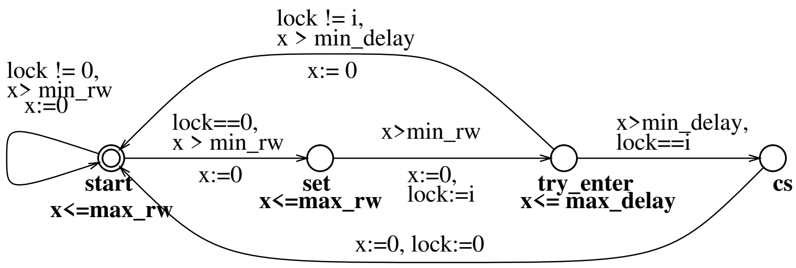


Figure 6.3: A PTA model of Fischer's mutual exclusion algorithm

composition for PTAs is standard, see for instance [LPY97] for their definitions.

6.3 Symbolic State Space Exploration

Our aim is to use basically the same algorithm for parametric timed model checking as for timed model checking. We represent sets of states symbolically in a similar way and support the same operations used for timed model checking. In the nonparametric case, sets of states can be efficiently represented using matrices [Dil98]. Similarly, in this chapter we represent sets of states symbolically as *(constrained) parametric difference bound matrices*.

A very similar approach was followed in [AAB00], although not worked out in detail. New in our presentation is the systematic use of structural operational semantics to deal with the nondeterministic computation that takes place in the parametric case.

6.3.1 Parametric difference bound matrices

In the nonparametric case, a difference bound matrix is a $(m + 1) \times (m + 1)$ matrix whose entries are elements from $(\mathsf{Z} \cup \{\infty\}) \times \{0, 1\}$. An entry $(c, 1)$ for D_{ij} denotes a nonstrict bound $x_i - x_j \leq c$, whereas an entry $(c, 0)$ denotes a strict bound $x_i - x_j < c$. In the parametric case, instead of using integers in the entries, we will use linear expressions over the parameters. Also, we find it convenient to view the matrix slightly more abstractly as a set of guards.

Definition 6.3.1 (PDBM) A *parametric difference bound matrix (PDBM)* is a set D which contains, for all $0 \leq i, j \leq m$, a simple guard D_{ij} of the form $x_i - x_j \prec_{ij} e_{ij}$. We require that, for all i, D_{ii} is of the form $x_i - x_i \leq 0$. Given a parameter valuation v, the *semantics* of D is given by $[\![D]\!]_v = [\![\bigwedge_{i,j} D_{ij}]\!]_v$. PDBM D is *satisfiable* for v if $[\![D]\!]_v$ is nonempty. If f is a guard of the form $x_i - x_j \prec e$ with $i \neq j$ (i.e., a proper guard), then $D[f]$ denotes the PDBM obtained from D by replacing D_{ij} by f. If i, j are indices then D^{ij} denotes the pair $(e_{ij}, \prec_{ij})$; we call D^{ij} a *bound* of D. Clearly, a PDBM is fully determined by its bounds.

Definition 6.3.2 (Constrained PDBM) A *constrained PDBM* is a pair (C, D) where C is a constraint set and D is a PDBM. We require that $C \models p \geq 0$, for each p, and $C \models e_{0i} \leq 0$, for each i. The *semantics* of (C, D) is given by $[\![C, D]\!] = \{(v, w) \mid v \in [\![C]\!] \wedge w \in [\![D]\!]_v\}$. We call (C, D) *satisfiable* if $[\![C, D]\!]$ is nonempty.

Condition $C \models p \geq 0$ expresses that parameter p may only take nonnegative values. The condition $C \models e_{0i} \leq 0$ ensures a nonnegative lower bound on the value of clock x_i. Such a condition is required since clocks in a PTA only take nonnegative values. A similar condition occurs in [Yov98]. In the setting of [Dil98] the condition of nonnegative lower bounds is not needed since in this paper clocks (called timers) may take values in $\mathbb{R}$. In [LLPY97, Alu98, CGP99, AAB00] the condition (or something similar) is needed but not mentioned.[3]

PDBMs with the tightest possible bounds are called *canonical*. To formalize this notion, we define an addition operation on linear expressions by

$$\begin{aligned}(t_1p_1 + \cdots + t_np_n + t_0) \quad &+ \quad (t'_1p_1 + \cdots + t'_np_n + t'_0)\\ &\triangleq \quad (t_1 + t'_1)p_1 + \cdots + (t_n + t'_n)p_n + (t_0 + t'_0).\end{aligned}$$

Also, we view Boolean connectives as operations on relation symbols $\leq$ and $<$ by identifying $\leq$ with 1 and $<$ with 0. Thus we have, for instance, $(\leq \wedge \leq) = \leq$, $(\leq \wedge <) = <$, $\neg \leq = <$, and $(\leq \implies <) = <$.

Our definition of a canonical form of a constrained PDBM is essentially equivalent to the one for standard DBMs.

Definition 6.3.3 (Canonical Form) A constrained PDBM (C, D) is *in canonical form* iff for all i, j, k, $C \models e_{ij}\ (\prec_{ij} \implies \prec_{ik} \wedge \prec_{kj})\ e_{ik} + e_{kj}$.

The proof of the following technical lemma is immediate from the definitions.

Lemma 6.3.4

1. *If* $v \models e \prec e'$ *and* $v \models e' \prec' e''$ *then* $v \models e\ (\prec \wedge \prec')\ e''$.
2. *If* $(v, w) \models x - y \prec e$ *and* $v \models e \prec' e'$ *then* $(v, w) \models x - y\ (\prec \wedge \prec')\ e'$.
3. *If* $v \models e\ (\prec \wedge \prec')\ e'$ *then* $v \models e \prec e'$.
4. *If* $(v, w) \models x - y\ (\prec \wedge \prec')\ e$ *then* $(v, w) \models x - y \prec e$.
5. *If* $(v, w) \models x - y \prec e$ *and* $(v, w) \models y - z \prec' e'$ *then* $(v, w) \models x - z\ (\prec \wedge \prec')\ e + e'$.
6. $v \models \neg(e \prec e')$ *iff* $v \models e'\ (\neg \prec)\ e$.

The next lemma states that canonicity of a constrained PDBM guarantees satisfiability

Lemma 6.3.5 *Suppose* (C, D) *is a constrained PDBM in canonical form and* $v \in [\![C]\!]$. *Then* D *is satisfiable for* v.

PROOF: By induction on i, with $0 \leq i \leq m$, we construct a valuation $(t_0, \ldots, t_i)$ for clock variables $(x_0, \ldots, x_i)$ such that all constraints D_{jk} for $0 \leq j, k \leq i$ are met.

To begin with, we set $t_0 = 0$. Then, trivially, $(v, x_0 \mapsto t_0) \models D_{00}$.

For the induction step, suppose that for some $i < m$ we have a valuation $(t_0, \ldots, t_i)$ for variables $(x_0, \ldots, x_i)$ such that all constraints D_{jk} for $0 \leq j, k \leq i$ are met. In order to

[3] For instance, in [CGP99] it is claimed on page 289: "If the clock zone is empty or unsatisfiable, there will be at least one negative entry in the main diagonal." This claim is incorrect. A counterexample is the canonical form of a DBM that contains as the only nontrivial guard $x_1 - x_0 \leq -1$.

extend this valuation to x_{i+1}, we have to find a value t_{i+1} such that the following simple guards hold for valuation $(v, x_0 \mapsto t_0, \ldots, x_{i+1} \mapsto t_{i+1})$:

$$D_{i+1,0} \quad \cdots \quad D_{i+1,i} \;\; D_{0,i+1} \quad \cdots \quad D_{i,i+1} \;\; D_{i+1,i+1} \tag{6.1}$$

Here the first $i + 1$ simple guards give upper bounds for t_{i+1}, the second $i + 1$ simple guards give lower bounds for t_{i+1}, and the last simple guard is trivially met by any choice for t_{i+1}. We claim that each of the upper bounds is larger than each of the lower bounds. In particular, the minimum of the upper bounds is larger than the maximum of the lower bounds. This gives us a nonempty interval of possible values for t_{i+1} to choose from. Formally, we claim that, for all $0 \leq j, k < i+1$, the following formula holds for valuation $(v, x_0 \mapsto t_0, \ldots, x_i \mapsto t_i)$:

$$x_j - e_{j,i+1} \quad \prec_{j,i+1} \wedge \prec_{i+1,k} \quad x_k + e_{i+1,k} \tag{6.2}$$

To see why (6.2) holds, observe that by induction hypothesis $(v, x_0 \mapsto t_0, \ldots, x_i \mapsto t_i) \models$

$$x_j - x_k \quad \prec_{jk} \quad e_{jk} \tag{6.3}$$

Furthermore, since (C, D) is canonical and $v \in [\![C]\!]$, $v \models$

$$e_{jk} \quad (\prec_{jk} \Longrightarrow \prec_{j,i+1} \wedge \prec_{i+1,k}) \quad e_{j,i+1} + e_{i+1,k} \tag{6.4}$$

Combination of (6.3) and (6.4), using Lemma 6.3.4(2), gives $(v, x_0 \mapsto t_0, \ldots, x_i \mapsto t_i) \models$

$$x_j - x_k \quad \prec_{j,i+1} \wedge \prec_{i+1,k} \quad e_{j,i+1} + e_{i+1,k}$$

which is equivalent to (6.2). This means that we can choose t_{i+1} in accordance with all the guards of (6.1). In particular, guard $D_{0,i+1}$ holds, which by the assumption that lower bounds on clocks are nonnegative implies that t_{i+1} is nonnegative. This completes the proof of the induction step and thereby of the lemma. □

The following lemma essentially carries over from the nonparametric case too, see for instance [Dil98]. As a direct consequence, semantic inclusion of constrained PDBMs is decidable for canonical PDBMs (using the oracle function).

Lemma 6.3.6 *Suppose $(C, D), (C', D')$ are constrained PDBMs and (C, D) is canonical. Then $[\![C, D]\!] \subseteq [\![C', D']\!] \Leftrightarrow ([\![C]\!] \subseteq [\![C']\!] \wedge \forall i, j : C \models e_{ij}(\prec_{ij} \Longrightarrow \prec'_{ij})e'_{ij})$.*

6.3.2 Operations on PDBMs

Our algorithm requires basically four operations to be implemented on constrained PDBMs: adding guards, canonicalization, resetting clocks and computing time successors.

Adding Guards

In the case of DBMs, adding a guard is a simple operation. It is implemented by taking the conjunction of a DBM and the guard (which is also viewed as a DBM). The conjunction operation just takes the pointwise minimum of the entries in both matrices. In the parametric case, adding a guard to a constrained PDBM may result in a set of constrained PDBMs. We define a relation $\Lleftarrow$ which relates a constrained PDBM and a guard to a collection of constrained PDBMs that satisfy this guard. For this we need an operation $\mathcal{C}$ that takes a PDBM and a simple guard, and produces a constraint stating that the bound imposed by the guard is weaker than the corresponding bound in the PDBM. Let $D^{ij} = (e_{ij}, \prec_{ij})$. Then

$$\mathcal{C}(D, x_i - x_j \prec e) \quad = \quad e_{ij}\ (\prec_{ij} \Longrightarrow \prec)\ e.$$

Relation $\Lleftarrow$ is defined as the smallest relation that satisfies the following rules:

$$(R1)\ \frac{\mathcal{O}(\mathcal{C}(D,f),C) = \text{yes}}{(C,D) \overset{f}{\Lleftarrow} (C,D)} \qquad (R2)\ \frac{\mathcal{O}(\mathcal{C}(D,f),C) = \text{no},\ f \text{ proper}}{(C,D) \overset{f}{\Lleftarrow} (C,D[f])} \tag{6.5}$$

$$(R3)\ \frac{\mathcal{O}(\mathcal{C}(D,f),C) = \text{split}}{(C,D) \overset{f}{\Lleftarrow} (C \cup \{\mathcal{C}(D,f)\}, D)} \qquad (R4)\ \frac{\mathcal{O}(\mathcal{C}(D,f),C) = \text{split},\ f \text{ proper}}{(C,D) \overset{f}{\Lleftarrow} (C \cup \{\neg\mathcal{C}(D,f)\}, D[f])}$$

$$(R5)\ \frac{(C,D) \overset{g}{\Lleftarrow} (C',D')\ ,\ (C'D') \overset{g'}{\Lleftarrow} (C'',D'')}{(C,D)\ \overset{g\wedge g'}{\Lleftarrow}\ (C'',D'')}$$

If the oracle replies "yes" then adding a simple guard will not change the constrained PDBM. If the answer is "no" then we tighten the bound in the PDBM. With the answer "split" there are two possibilities and two PDBMs with updated constraint systems are returned. Thus the result of the operation of adding a guard is a *set* of constrained PDBMs. The side condition "f proper" in $R2$ and $R4$ rules out guards of the form $x_i - x_i \prec e$ and thereby ensures that the diagonal bounds in the PDBM always remain equal to $(0, \leq)$. It is routine to check, using Lemma 6.3.4, that relation $\Lleftarrow$ is well-defined in the sense that $(C,D) \overset{g}{\Lleftarrow} (C',D')$ implies that (C',D') is a constrained PDBM. In particular the condition that clocks have nonnegative lower bounds is met. Note that if we update a bound in D the semantics of the PDBM may become empty: a typical situation occurs when D contains a constraint $x \geq 5$ and we add a guard $x \leq 3$. Note however that $(C,D) \overset{g}{\Lleftarrow} (C',D')$ and $[\![C]\!] \neq \emptyset$ implies $[\![C']\!] \neq \emptyset$. The following lemma characterizes $\Lleftarrow$ semantically.

Lemma 6.3.7 $[\![C,D]\!] \cap [\![g]\!] \ = \ \bigcup\{[\![C',D']\!] \mid (C,D) \overset{g}{\Lleftarrow} (C',D')\}$.

PROOF: "$\subseteq$". Assume $(v,w) \in [\![C,D]\!] \wedge (v,w) \models g$. By structural induction on g we prove that there exists a constrained PDBM (C',D') such that $(C,D) \overset{g}{\Lleftarrow} (C',D')$ and $(v,w) \in [\![C',D']\!]$.

For the induction basis, suppose g is of the form $x_i - x_j \prec e$. We consider four cases:

- $\mathcal{O}(\mathcal{C}(D,g),C) = \text{yes}$. Let $C' = C$ and $D' = D$. Then trivially $(v,w) \in [\![C',D']\!]$ and, by rule $R1$, $(C,D) \overset{g}{\Lleftarrow} (C',D')$.

- $\mathcal{O}(\mathcal{C}(D,g),C) = \text{no}$. By contradiction we prove that g is proper. Suppose g is not proper. Then, since $i = j$ and $v \models \neg e_{ij}(\prec_{ij} \implies \prec)e$, $v \models \neg(0 \prec e)$. By Lemma 6.3.4(6), $v \models e\neg \prec 0$. But $(v,w) \models g$ implies $v \models 0 \prec e$. Hence, by Lemma 6.3.4(1), $v \models 0 < 0$, a contradiction. Let $C' = C$ and $D' = D[g]$. Then, by rule $R2$, $(C,D) \overset{g}{\Lleftarrow} (C',D')$. Since $(v,w) \in [\![C,D]\!]$ and $(v,w) \models g$, it follows that $(v,w) \in [\![C',D']\!]$.

- $\mathcal{O}(\mathcal{C}(D,g),C) = \text{split}$ and $v \models \mathcal{C}(D,g)$. Let $C' = C \cup \{\mathcal{C}(D,g)\}$ and $D' = D$. Then, by rule $R3$, $(C,D) \overset{g}{\Leftarrow\!\!\dashv} (C',D')$. Moreover, by the assumptions, $(v,w) \in [\![C',D']\!]$.

- $\mathcal{O}(\mathcal{C}(D,g),C) = \text{split}$ and $v \models \neg\mathcal{C}(D,g)$. By contradiction we prove that g is proper. Suppose g is not proper. Then, since $v \models \neg\mathcal{C}(D,g)$, $v \models \neg(0 \prec e)$. By Lemma 6.3.4(6), $v \models e\neg \prec 0$. But $(v,w) \models g$ implies $v \models 0 \prec e$. Hence, by Lemma 6.3.4(1), $v \models 0 < 0$, a contradiction. Let $C' = C \cup \{\neg\mathcal{C}(D,g)\}$ and $D' = D[g]$. Then, by rule $R4$, $(C,D) \overset{g}{\Leftarrow\!\!\dashv} (C',D')$. By the assumptions $(v,w) \in [\![C',D']\!]$.

For the induction step, suppose that g is of the form $g' \wedge g''$. Then $(v,w) \models g'$. By induction hypothesis, there exist C'', D'' such that $(C,D) \overset{g'}{\Leftarrow\!\!\dashv} (C'',D'')$ and $(v,w) \in [\![C'',D'']\!]$. Since $(v,w) \models g''$, we can use the induction hypothesis once more to infer that there exist C', D' such that $(C'',D'') \overset{g''}{\Leftarrow\!\!\dashv} (C',D')$ and $(v,w) \in [\![C',D']\!]$. Moreover, by rule $R5$, $(C,D) \overset{g}{\Leftarrow\!\!\dashv} (C',D')$.

"$\supseteq$" Assume $(C,D) \overset{g}{\Leftarrow\!\!\dashv} (C',D')$ and $(v,w) \in [\![C',D']\!]$. By induction on size of the derivation of $(C,D) \overset{g}{\Leftarrow\!\!\dashv} (C',D')$, we establish $(v,w) \in [\![C,D]\!]$ and $(v,w) \models g$. There are five cases, depending on the last rule r used in the derivation of $(C,D) \overset{g}{\Leftarrow\!\!\dashv} (C',D')$.

1. $r = R1$. Then $C = C'$, $D = D'$ and $C \models \mathcal{C}(D,g)$. Let g be of the form $x_i - x_j \prec e$. Hence $(v,w) \in [\![C,D]\!]$ and $v \models \mathcal{C}(D,g)$. By the first statement $(v,w) \models x_i - x_j \prec_{ij}^D e_{ij}^D$, and by the second statement $v \models e_{ij}^D \; (\prec_{ij}^D \implies \prec) \; e$. Combination of these two observations, using parts (2) and (4) of Lemma 6.3.4 yields $(v,w) \models g$.

2. $r = R2$. Then $C = C'$, $D' = D[g]$ and $C \models \neg\mathcal{C}(D,g)$. Hence $(v,w) \models g$ and $v \models \neg\mathcal{C}(D,g)$. Let g be of the form $x_i - x_j \prec e$. By Lemma 6.3.4(6), $v \models e \neg(\prec_{ij}^D \implies \prec) \; e_{ij}^D$. Using parts (2) and (4) of Lemma 6.3.4, combination of these two observations yields $(v,w) \models x_i - x_j \prec_{ij}^D e_{ij}^D$. Since trivially (v,w) is a model for all the other guards in D, $(v,w) \in [\![C,D]\!]$.

3. $r = R3$. Then $C' = C \cup \{\mathcal{C}(D,g)\}$ and $D' = D$. Let g be of the form $x_i - x_j \prec e$. We have $(v,w) \in [\![C,D]\!]$. This implies $(v,w) \models x_i - x_j \prec_{ij}^D e_{ij}^D$. We also have $v \models e_{ij}^D \; (\prec_{ij}^D \implies \prec) \; e$. Combination of these two observations, using parts (2) and (4) of Lemma 6.3.4 yields $(v,w) \models g$.

4. $r = R4$. Then $C' = C \cup \{\neg\mathcal{C}(D,g)\}$ and $D' = D[g]$. We have $v \models \neg\mathcal{C}(D,g)$ and $(v,w) \models g$. Let g be of the form $x_i - x_j \prec e$. By Lemma 6.3.4(6), $v \models e \neg(\prec_{ij}^D \implies \prec) \; e_{ij}^D$. Using parts (2) and (4) of Lemma 6.3.4 yields $(v,w) \models x_i - x_j \prec_{ij}^D e_{ij}^D$. Since trivially (v,w) is a model for all other guards in D, $(v,w) \in [\![C,D]\!]$.

5. $r = R5$. Then g is of the form $g' \wedge g''$ and there are C'', D'' such that $(C,D) \overset{g'}{\Leftarrow\!\!\dashv} (C'',D'')$ and $(C'',D'') \overset{g''}{\Leftarrow\!\!\dashv} (C',D')$. By induction hypothesis, $(v,w) \in [\![C'',D'']\!]$ and $(v,w) \models g''$. Again by induction hypothesis, $(v,w) \in [\![C,D]\!]$ and $(v,w) \models g'$. It follows that $(v,w) \models g$.

□

FLOYD-WARSHALL (C_0, D_0)
$(C, D) := (C_0, D_0)$
for $k = 0$ **to** m
  do for $i = 0$ **to** m
    do for $j = 0$ **to** m
      $(C, D) :=$ **choose** (C', D') **such that**
$$(C, D) \overset{x_i - x_j \prec_{ik} \wedge \prec_{kj} e_{ik} + e_{kj}}{\Lleftarrow} (C', D')$$
return (C, D)

Figure 6.4: The Floyd-Warshall algorithm

Canonicalization

Each DBM can be brought into canonical form using classical algorithms for computing all-pairs shortest paths, for instance the Floyd-Warshall (FW) algorithm [CLR90]. In the parametric case, we also apply this approach except that now we run FW *symbolically*, see Figure 6.4. The algorithm repeatedly compares the difference between two clocks to the difference obtained by taking an intermediate clock into account (cf. the inequality in Definition 6.3.3). The symbolic FW algorithm contains a nondeterministic assignment, in which (C, D) nondeterministically gets a value from a set. This set may be empty, in which case the algorithm terminates unsuccessfully. We are interested in the (possibly empty, finite) set of all possible constrained PDBMs that may result when running the algorithm.

For the purpose of proving things we find it convenient to describe the computation steps of the symbolic FW algorithm in SOS style. In the SOS description, we use configurations of the form (k, i, j, C, D), where (C, D) is a constrained PDBM and $k, i, j \in [0, m+1]$ record the values of indices. In the rules below, k, i, j range over $[0, m]$.

$$\frac{(C, D) \overset{x_i - x_j \prec_{ik} \wedge \prec_{kj} e_{ik} + e_{kj}}{\Lleftarrow} (C', D')}{(k, i, j, C, D) \rightarrow_{FW} (k, i, j+1, C', D')}$$

$$(k, i, m+1, C, D) \rightarrow_{FW} (k, i+1, 0, C, D)$$

$$(k, m+1, 0, C, D) \rightarrow_{FW} (k+1, 0, 0, C, D)$$

We write $(C, D) \rightarrow_c (C', D')$ if there exists a sequence of $\rightarrow_{FW}$ steps leading from configuration $(0, 0, 0, C, D)$ to configuration $(m+1, 0, 0, C', D')$. In this case, we say that (C', D') is an *outcome* of the symbolic Floyd-Warshall algorithm on (C, D). It is easy to see that the set of all outcomes is finite and can be effectively computed. If the semantics of (C, D) is empty, then the set of outcomes is also empty. We write $(C, D) \overset{g}{\Lleftarrow}_c (C', D')$ iff $(C, D) \overset{g}{\Lleftarrow} (C'', D'') \rightarrow_c (C', D')$, for some C'', D''.

The following lemma says that if we run the symbolic Floyd-Warshall algorithm, the union of the semantics of the outcomes equals the semantics of the original constrained PDBM.

Lemma 6.3.8 $[\![C, D]\!] = \bigcup\{[\![C', D']\!] \mid (C, D) \rightarrow_c (C', D')\}$.

PROOF: By an inductive argument, using Lemma 6.3.7 and the fact that, for any valuation (v, w) in the semantics of (C, D),

$$\begin{array}{lll}(v,w) & \models & x_i - x_k \prec_{ik} e_{ik} \text{ and} \\ (v,w) & \models & x_k - x_j \prec_{kj} e_{kj}, \text{ and therefore by Lemma 6.3.4(5)} \\ (v,w) & \models & x_i - x_j \prec_{ik} \wedge \prec_{kj} e_{ik} + e_{kj}.\end{array}$$

□

Lemma 6.3.9 *Each outcome of the symbolic Floyd-Warshall algorithm is a constrained PDBM in canonical form.*

PROOF: As in [CLR90]. □

Remark 6.3.10 Non-parametric DBMs can be canonicalized in $\mathcal{O}(n^3)$, where n is the number of clocks. In the parametric case, however, each operation of comparing the bound $D(x, x')$ to the weight of another path from x to x' may give rise to two new PDBMs if this comparison leads to a split. Then the two PDBMs must both be canonicalized to obtain all possible PDBMs with tightest bounds. Still, that part of these two PDBMs which was already canonical, does not need to be investigated again. So in the worst case, the cost of the algorithm becomes $\mathcal{O}(2^{n^3})$. In practice, it turns out that this is hardly ever the case.

Resetting Clocks

A third operation on PDBMs that we need is resetting clocks. Since we do not allow parameters in reset sets, the reset operation on PDBMs is essentially the same as for DBMs, see [Yov98]. If D is a PDBM and r is a singleton reset set $\{x_i := b\}$, then $D[r]$ is the PDBM obtained by (1) replacing each bound D^{ij}, for $j \neq i$, by $(e_{0j} + b, \prec_{0j})$; (2) replacing each bound D^{ji}, for $j \neq i$, by $(e_{j0} - b, \prec_{j0})$. We generalize this definition to arbitrary reset sets by

$$D[x_{i_1} := b_1, \ldots, x_{i_h} := b_h] \quad = \quad D[x_{i_1} := b_1] \ldots [x_{i_h} := b_h].$$

It easily follows from the definitions that resets preserve canonicity. Note also that the reset operation is well-defined on constrained PDBMs: if (C, D) is a constrained PDBM then $(C, D[r])$ is a constrained PDBM as well: since clocks can only be reset to natural numbers, lower bounds on clocks remain nonnegative.

Lemma 6.3.11 *If (C, D) is canonical then $(C, D[r])$ is canonical as well.*

The following lemma characterizes the reset operation semantically.

Lemma 6.3.12 *Let (C, D) be a constrained PDBM in canonical form, $v \in [\![C]\!]$, and w a clock valuation. Then $w \in [\![D[r]]\!]_v$ iff $\exists w' \in [\![D]\!]_v : w = w'[r]$.*

PROOF: We only prove the lemma for singleton resets. Using Lemma 6.3.11, the generalization to arbitrary resets is straightforward. Let $r = \{x_i := b\}$ and $D' = D[r]$.
"$\Leftarrow$" Suppose $w' \in [\![D]\!]_v$ and $w = w'[r]$. In order to prove $w \in [\![D']\!]_v$, we must show that $(v, w) \models D'_{kj}$, for all k and j. There are four cases:

1. $k \neq i \neq j$. Then $D'_{kj} = D_{kj}$. Since $(v, w') \models D_{kj}$ and w and w' agree on all clocks occurring in D_{kj}, $(v, w) \models D'_{kj}$.
2. $k = i = j$. Then $D'_{kj} = D_{kj}$. Since $(v, w') \models D_{ii}$, $0 \prec_{ii} e_{ii}[v]$. Hence $(v, w) \models D'_{kj}$.
3. $k \neq i = j$. Then $D'_{kj} = x_k - x_j \prec_{k0} e_{k0} - b$. Using that $(v, w') \models D_{k0}$, we derive $w(x_k) - w(x_j) = w'(x_k) - b \prec_{k0} e_{k0}[v] - b$. Hence $(v, w) \models D'_{kj}$.
4. $k = i \neq j$. Then $D'_{kj} = x_k - x_j \prec_{0j} e_{0j} + b$. Using that $(v, w') \models D_{0j}$, we derive $w(x_k) - w(x_j) = b - w'(x_j) \prec_{0j} e_{0j}[v] + b$. Hence $(v, w) \models D'_{kj}$.

"$\Rightarrow$" Suppose $w \in [\![D']\!]_v$. We have to prove that there exists a clock valuation $w' \in [\![D]\!]_v$ such that $w = w'[r]$. Clearly we need to choose w' in such a way that, for all $j \neq i$, $w'(x_j) = w(x_j)$. This means that, for any choice of $w'(x_i)$, for all $j \neq i \neq k, v, w' \models D_{jk}$. Using the same argument as in the proof of Lemma 6.3.5, we can find a value for $w'(x_i)$ such that also the remaining simple guards of D are satisfied. $\square$

Time Successors

Finally, we need to transform PDBMs for the passage of time, notation $D\!\uparrow$. As in the DBMs case [Dil98], this is done by setting the upper bounds $x_i - x_0$ to $(\infty, <)$, for each $i \neq 0$, and leaving all other bounds unchanged. We have the following lemma.

Lemma 6.3.13 *Suppose (C, D) is a constrained PDBM in canonical form, $v \in [\![C]\!]$, and w a clock valuation. Then $w \in [\![D\!\uparrow]\!]_v$ iff $\exists d \geq 0\ \exists w' \in [\![D]\!]_v : w' + d = w$.*

PROOF: "$\Leftarrow$" Suppose $d \geq 0$, $w' \in [\![D]\!]_v$ and $w' + d = w$. We claim that $w \in [\![D\!\uparrow]\!]_v$. For this we must show that for each guard f of $D\!\uparrow$, $(v, w) \models f$. Let f be of the form $x_i - x_j \prec e$. We distinguish between three cases:

- $i \neq 0 \wedge j = 0$. In this case, by definition of $D\!\uparrow$, f is of the form $x_i - x_0 < \infty$, and so $(v, w) \models f$ trivially holds.
- $i = 0$. In this case f is also a constraint of D. Since $w' \in [\![D]\!]_v$ we have $(v, w') \models f$, and thus $-w'(x_j) \prec e[v]$. But since $d \geq 0$, this means that $-w(x_j) = -w'(x_i) - d \prec e[v]$ and therefore $(v, w) \models f$.
- $i \neq 0 \wedge j \neq 0$. In this case f is again a constraint of D. Since $w' \in [\![D]\!]_v$ we have $(v, w') \models f$, and therefore $w'(x_i) - w'(x_j) \prec e[v]$. But this means that $w'(x_i) - w'(x_j) = (w(x_i) - d) - (w(x_j) - d) \prec e[v]$ and therefore $(v, w) \models f$.

"$\Rightarrow$" Suppose $w \in [\![D\!\uparrow]\!]_v$. If $m = 0$ (i.e., there are no clocks) then $D\!\uparrow = D$ and the rhs of the implication trivially holds (take $w' = w$ and $d = 0$). So assume $m > 0$. For all indices i, j with $i \neq 0$ and $j \neq 0$, $(v, w) \models D_{ij}$. Hence $w(x_i) - w(x_j) \prec_{ij} e_{ij}[v]$. Thus, for any real number t, $w(x_i) - t - (w(x_j) - t) \prec_{ij} e_{ij}[v]$. But this means $(v, w - t) \models D_{ij}$. It remains to be shown that there exists a value d such that in valuation $(v, w - d)$ also the remaining guards D_{0i} and D_{i0} hold. Let

$$\begin{aligned} t_0 &= \max(0, w(x_1) - e_{10}[v], \ldots, w(x_n) - e_{n0}[v]) \\ t_1 &= \min(w(x_1) + e_{01}[v], \ldots, w(x_n) + e_{0n}[v]) \\ d &= (t_0 + t_1)/2 \\ w' &= w - d \end{aligned}$$

Intuitively, t_0 gives the least amount of time one has to go backwards in time from w to meet all upper bounds of D (modulo strictness), whereas t_1 gives the largest amount of time one can go backwards in time from w without violating any of the lower bounds of D (again modulo strictness). Value d sits right in the middle of these two. We claim that d and w' satisfy the required properties. For any i, since $(v, w) \models D_{0i}$, trivially

$$0 \quad \prec_{0i} \quad w(x_i) + e_{0i}[v] \tag{6.6}$$

Using that D is canonical we have, for any i, j,

$$e_{ji}[v] \; (\prec_{ji} \Longrightarrow \prec_{j0} \wedge \prec_{0i}) \; e_{j0}[v] + e_{0i}[v]$$

and, since $v, w \models D_{ji}$,

$$w(x_j) - w(x_i) \prec_{ji} e_{ji}[v].$$

Using these two observations we infer

$$w(x_j) - e_{j0}[v] \; (\prec_{ji} \Longrightarrow \prec_{j0} \wedge \prec_{0i}) \; w(x_j) - e_{ji}[v] + e_{0i}[v] \prec_{ji} w(x_i) + e_{0i}[v].$$

Hence

$$w(x_j) - e_{j0}[v] \prec_{j0} \wedge \prec_{0i} w(x_i) + e_{0i}[v] \tag{6.7}$$

By inequalities (6.6) and (6.7), each subterm of $\max$ in the definition of t_0 is dominated by each subterm of $\min$ in the definition of t_1. This implies $0 \leq t_0 \leq t_1$.

Now either $t_0 < t_1$ or $t_0 = t_1$. In the first case it easy to prove that in valuation (v, w) the guards D_{0i} and D_{i0} hold, for any i:

$$w'(x_i) = w(x_i) - d < w(x_i) - t_0 \leq w(x_i) - (w(x_i) - e_{i0}[v]) = e_{i0}[v]$$

and thus $w'(x_i) < e_{i0}[v]$ and $v, w' \models D_{i0}$. Also

$$-w'(x_i) = -w(x_i) + d < -w(x_i) + t_1 \leq -w(x_i) + (w(x_i) + e_{0i}[v]) = e_{0i}[v]$$

and so $-w'(x_i) < e_{0i}[v]$ and $v, w' \models D_{0i}$.

In the second case, fix an i. If $w(x_i) - e_{i0}[v] < t_0$ then

$$w'(x_i) = w(x_i) - d = w(x_i) - t_0 < w(x_i) - (w(x_i) - e_{i0}[v]) = e_{i0}[v]$$

and thus $w'(x_i) < e_{i0}[v]$ and $v, w' \models D_{i0}$. Otherwise, if $w(x_i) - e_{i0}[v] = t_0$ observe that by $t_0 = t_1$, inequality (6.7) and the fact that, $t_1 = w(x_j) + e_{0j}[v]$, for some j, $\prec_{i0} = \leq$. Since

$$w'(x_i) = w(x_i) - d \leq w(x_i) - t_0 \leq w(x_i) - (w(x_i) - e_{i0}[v]) \leq e_{i0}[v]$$

and thus $w'(x_i) \leq e_{i0}[v]$ this implies $v, w' \models D_{i0}$.

The proof that $v, w' \models D_{0i}$, for all i, in the case where $t_0 = t_1$ proceeds similarly. □

6.3.3 Symbolic semantics

Having defined the four operations on PDBMs, we are now in a position to describe the semantics of a parametric timed automaton symbolically.

Definition 6.3.14 (Symbolic semantics) Let $\mathcal{A} = (Q, q_0, \to, I)$ be a PTA. The *symbolic semantics* of $\mathcal{A}$ is an LTS: the states are triples (q, C, D) with q a location from Q and (C, D) a constrained PDBM in canonical form such that $[\![C, D]\!] \subseteq [\![I(q)]\!]$; the set of initial states is

$$\{(q_0, C, D) \mid (C_0, \mathsf{E}\!\uparrow) \overset{I(q_0)}{\Leftarrow\!\!\!\!\dashv}_c (C, D)\},$$

where $C_0 = \{p \geq 0 \mid p \in P\}$, E is the PDBM with $\mathsf{E}^{ij} = (0, \leq)$, for all i, j; the transitions are defined by the following rule:

$$\frac{q \xrightarrow{a,g,r} q' \ , \ (C, D) \overset{g}{\Leftarrow\!\!\!\!\dashv}_c (C'', D'') \ , \ (C'', D''[r]\!\uparrow) \overset{I(q')}{\Leftarrow\!\!\!\!\dashv}_c (C', D')}{(q, C, D) \to (q', C', D')}.$$

Observe that if (q, C, D) is a state in the symbolic semantics and $(v, w) \in [\![C, D]\!]$, then (q, w) is a state of the concrete semantics $[\![\mathcal{A}]\!]_v$. It is also easy to see that the symbolic semantics of a PTA is a finitely branching LTS. It may have infinitely many reachable states though.

In order to establish that each run in the symbolic semantics can be simulated by a run in the concrete semantics, we require two lemmas.

Lemma 6.3.15 *Suppose that (q, C, D) is an initial state of the symbolic semantics of $\mathcal{A}$ with $(v, w) \in [\![C, D]\!]$. Then the concrete semantics $[\![\mathcal{A}]\!]_v$ has an initial state (q_0, w_0) from which state (q, w) can be reached.*

PROOF: Using the fact that $(v, w) \in [\![C, D]\!]$, the definition of initial states, Lemma 6.3.8 and Lemma 6.3.7, we know that $q = q_0$, $(v, w) \models I(q_0)$ and $(v, w) \in [\![C_0, \mathsf{E}\!\uparrow]\!]$. By Lemma 6.3.13, we get that there exists a $d \geq 0$ and $w_0 \in [\![\mathsf{E}]\!]_v$ such that $w_0 + d = w$. Since $(v, w) \models I(q_0)$ and invariants in a PTA only give upper bounds on clocks, also $(v, w_0) \models I(q_0)$. It follows that (q_0, w_0) is a state of the concrete semantics $[\![\mathcal{A}]\!]_v$ and $(q_0, w_0) \xrightarrow{d} (q, w)$. Since $w_0 \in [\![\mathsf{E}]\!]_v$, w_0 is of the form $\lambda x.0$. Hence (q_0, w_0) is an initial state of the concrete semantics. □

Lemma 6.3.16 *Suppose that $(q', C', D') \to (q, C, D)$ is a transition in the symbolic semantics of $\mathcal{A}$ and $(v, w) \in [\![C, D]\!]$. Then there exists a pair $(v, w') \in [\![C, D]\!]$ such that in the concrete semantics $[\![\mathcal{A}]\!]_v$ there is a path from (q', w') to (q, w).*

PROOF: By the definition of transitions in the symbolic semantics and Lemmas 6.3.8 and 6.3.7, we know that there is a transition $q' \xrightarrow{a,g,r} q$ in $\mathcal{A}$, and there are C'', D'' such that $(v, w) \models I(q)$, $(v, w) \in [\![C'', D''[r]\!\uparrow]\!]$ and $(C', D') \overset{g}{\Leftarrow\!\!\!\!\dashv}_c (C'', D'')$. By Lemma 6.3.13, we get that there exists a $d \geq 0$ and $w'' \in [\![D''[r]]\!]_v$ such that $w'' + d = w$. Since $(v, w) \models I(q)$ and invariants in a PTA only give upper bounds on clocks, also $(v, w'') \models I(q)$. It follows that (q, w'') is a state of the concrete semantics $[\![\mathcal{A}]\!]_v$ and $(q, w'') \xrightarrow{d} (q, w)$. Using Lemma 6.3.12 we get that there exists a $w' \in [\![D'']\!]_v$ such that $w'' = w'[r]$. Using Lemma 6.3.8 and Lemma 6.3.7 again, it follows that $(v, w') \models g$ and $(v, w') \in [\![C', D']\!]$. Since (q', C', D') is a state of the symbolic semantics, $(v, w') \models I(q')$. Hence, (q', w') is a state of the concrete semantics and $(q', w') \xrightarrow{a} (q, w'')$ is a transition in the concrete semantics. Combination of this transition with the transition $(q, w'') \xrightarrow{d} (q, w)$ gives the required path in the concrete semantics. □

Proposition 6.3.17 *For each parameter valuation v and clock valuation w, if there is a run in the symbolic semantics of $\mathcal{A}$ reaching state (q, C, D), with $(v, w) \in [\![C, D]\!]$, then this run can be simulated by a run in the concrete semantics $[\![\mathcal{A}]\!]_v$ reaching state (q, w).*

PROOF: By induction on the number of transitions in the run.

As basis we consider a run with 0 transitions, i.e., a run that consists of an initial state of the symbolic semantics. So this means that (q, C, D) is an initial state. The induction basis now directly follows using Lemma 6.3.15.

For the induction step, assume that we have a run in the symbolic semantics, ending with a transition $(q', C', D') \to (q, C, D)$. By $(v, w) \in [\![C, D]\!]$ and Lemma 6.3.16, there exists a pair $(v, w') \in [\![C, D]\!]$ such that in the concrete semantics $[\![\mathcal{A}]\!]_v$ there is a path from (q', w') to (q, w). By induction hypothesis, there is a path in the concrete semantics leading up to state (q', w'). Extension of this path with the path from (q', w') to (q, w) gives the required path in the concrete semantics. □

Conversely, for each path in the concrete semantics, we can find a path in the symbolic semantics such that the final state of the first path is semantically contained in the final state of the second path.

Proposition 6.3.18 *For each parameter valuation v and clock valuation w, if there is a run in the concrete semantics $[\![\mathcal{A}]\!]_v$ reaching a state (q, w), then this run can be simulated by a run in the symbolic semantics reaching a state (q, C, D) such that $(v, w) \in [\![C, D]\!]$.*

PROOF: In any execution in the concrete semantics, we can always insert zero delay transitions at any point. Also, two consecutive delay transitions $(q, w) \xrightarrow{d} (q, w + d)$ and $(q, w + d) \xrightarrow{d'} (q, w + d + d')$ can always be combined into a single delay transition $(q, w) \xrightarrow{d+d'} (q, w + d + d')$. Therefore, without loss of generality, we only consider concrete executions that start with a delay transition, and in which there is a strict alternation of action transitions and delay transitions. The proof is by induction on the number of action transitions.

As basis we consider a run $(q_0, w_0) \xrightarrow{d} (q_0, w_0 + d)$, where $w_0 = \lambda x.0$, consisting of a single time-passage transition. By definition of the concrete semantics, $(v, w_0 + d) \models I(q_0)$. Using Lemma 6.3.13, we have that $(v, w_0 + d) \in [\![C_0, \mathsf{E}{\uparrow}]\!]$ since $(v, w_0) \in [\![C_0, \mathsf{E}]\!]$. From $(v, w_0 + d) \in [\![C_0, \mathsf{E}{\uparrow}]\!]$ and $(v, w_0 + d) \models I(q_0)$, using Lemma 6.3.7 and Lemma 6.3.8 we get that there exists C, D such that $(C_0, \mathsf{E}\uparrow) \overset{I(q_0)}{\Lleftarrow} (C, D)$ and $(v, w_0 + d) \in [\![C, D]\!]$. By definition, (C, D) is an initial state of the symbolic semantics. This completes the proof of the induction basis.

For the induction step, assume that the run in the concrete semantics of $[\![\mathcal{A}]\!]_v$ ends with transitions $(q'', w'') \xrightarrow{a} (q', w') \xrightarrow{d} (q, w)$. By induction hypothesis, there exists a run in the symbolic semantics ending with a state (q'', C'', D'') such that $(v, w'') \in [\![C'', D'']\!]$.

By definition of the concrete semantics, there is a transition $q'' \xrightarrow{g,a,r} q'$ in $\mathcal{A}$ such that $(v, w'') \models g$ and $w' = w''[r]$. Moreover, we have $q' = q$, $w = w' + d$ and $(v, w) \models I(q)$. Using Lemma 6.3.7 and Lemma 6.3.8 gives that there exists C', D' such that $(C'', D'') \overset{g}{\Lleftarrow}_c (C', D')$ and $(v, w'') \in [\![C', D']\!]$. By Lemma 6.3.12, we have $w' \in [\![D'[r]]\!]_v$. Moreover, by Lemma 6.3.13, $w \in [\![D'[r]{\uparrow}]\!]_v$. Using $(v, w) \models I(q)$, Lemma 6.3.7 and Lemma 6.3.8, we infer that there exists C, D such that $(v, w) \in [\![C, D]\!]$ and $(C', D'[r]\uparrow) \overset{I(q)}{\Lleftarrow}_c (C, D)$.

Finally, using the definition of the symbolic semantics, we infer the existence of a transition $(q'', C'', D'') \to (q, C, D)$.
□

Example 6.3.19 Figure 6.5 shows the symbolic state-space of the automaton in Fig. 6.1 represented by constrained PDBMs. In the initial state the invariant $x \leq p$ limits the value of x, and since both clocks have the same value also the value of y. When taking the transition from $S0$ to $S1$ we have to compare the parameters p and q. This leads to a split where in the one case no state is reachable since the region is empty, and in the other (when $q \leq p$) $S1$ can be reached. From then on no more splits occur and only one new state is reachable.

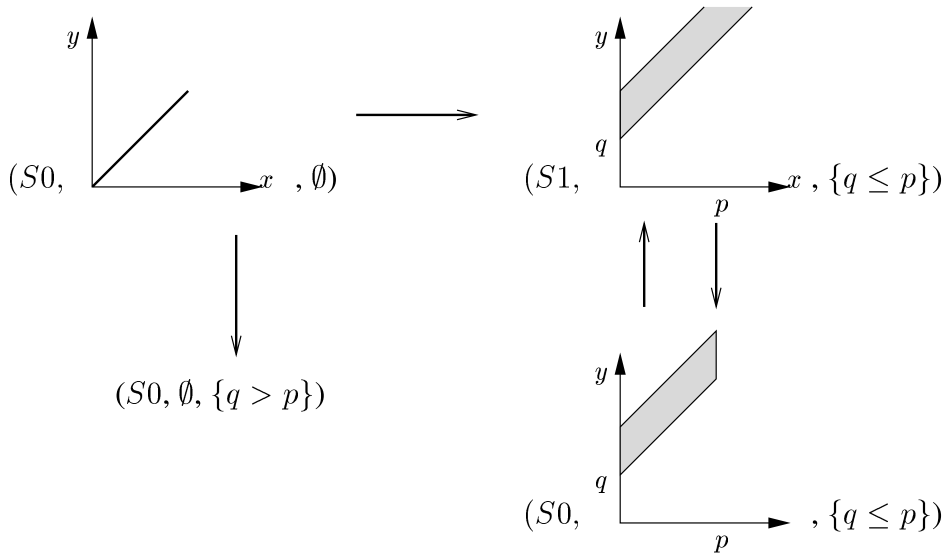


Figure 6.5: The symbolic state space of the PTA in Fig. 6.1.

6.3.4 Evaluating state formulas

We now define the predicate $\overset{\phi}{\Leftarrow}$ which relates a symbolic state and a state formula ϕ (as defined in Definition 6.2.7) to a collection of symbolic states that satisfy ϕ.

In order to check whether a state formula holds, we break it down into its atomic subformulas, namely checking locations and clock guards. Checking that a clock guard holds relies on the definition given earlier, of adding that clock guard to the constrained PDBM. We rely on a special normal form of the state formula, in which all $\neg$ signs have been pushed down to the basic formulas.

Definition 6.3.20 State formula ϕ is in *normal form* if all $\neg$ signs in ϕ appear only in subformulae of the form $\neg q$.

Since each simple guard with a $\neg$ sign in front can be rewritten to equivalent simple guard without, for each state formula there is an equivalent one in normal form.

In the following, let f be a simple guard, and ϕ be in normal form.

$$(Q_1)\frac{}{(q,C,D)\overset{q}{\Lleftarrow}(q,C,D)} \qquad (Q_2)\frac{q\neq q'}{(q,C,D)\overset{\neg q'}{\Lleftarrow}(q,C,D)}$$

$$(Q_3)\frac{(C,D)\overset{f}{\Lleftarrow}_c(C',D')}{(q,C,D)\overset{f}{\Lleftarrow}(q,C',D')}$$

$$(Q_4)\frac{(q,C,D)\overset{\phi_1}{\Lleftarrow}(q,C',D')\ ,\ (q,C',D')\overset{\phi_2}{\Lleftarrow}(q,C'',D'')}{(q,C,D)\overset{\phi_1\wedge\phi_2}{\Lleftarrow}(q,C'',D'')}$$

$$(Q_5)\frac{(q,C,D)\overset{\phi_1}{\Lleftarrow}(q,C',D')}{(q,C,D)\overset{\phi_1\vee\phi_2}{\Lleftarrow}(q,C',D')} \qquad (Q_6)\frac{(q,C,D)\overset{\phi_2}{\Lleftarrow}(q,C',D')}{(q,C,D)\overset{\phi_1\vee\phi_2}{\Lleftarrow}(q,C',D')}$$

The following lemma gives the soundness and completeness of relation $\overset{\phi}{\Lleftarrow}$.

Lemma 6.3.21 *Let ϕ be a state formula in normal form, q a location and (C,D) a constrained PDBMs. Let $[\![q,\phi]\!]$ denote the set $\{(v,w)\mid(q,w)\models_v\phi\}$. Then*

$$[\![C,D]\!]\cap[\![q,\phi]\!] \quad = \quad \bigcup\{[\![C',D']\!]\mid(q,C,D)\overset{\phi}{\Lleftarrow}(q,C',D')\}.$$

PROOF: "$\subseteq$": Assume that $(v,w)\in[\![C,D]\!]$ and $(q,w)\models_v\phi$. We prove that there are C', D' such that $(v,w)\in[\![C',D']\!]$ and $(q,C,D)\overset{\phi}{\Lleftarrow}(q,C',D')$. We proceed by induction on the structure of ϕ.

- Base cases.
  - Suppose $\phi=q'$. As $(q,w)\models_v q'$, clearly, $q=q'$. Since, by rule Q_1, $(q,C,D)\overset{q}{\Lleftarrow}(q,C,D)$, we can take $C=C'$ and $D=D'$ and the result follows.
  - Suppose $\phi=\neg q'$. Similar to the previous case, apply rule Q_2.
  - Suppose $\phi=f$ with f a simple guard. Then $(v,w)\in[\![C,D]\!]$ and $(v,w)\models f$. By Lemma 6.3.7 there exist C'',D'' such that $(C,D)\overset{f}{\Lleftarrow}(C'',D'')$ and $(v,w)\in[\![C'',D'']\!]$. Lemma 6.3.8 yields the existence of C',D' with $(C'',D'')\rightarrow_c(C',D')$ and $(v,w)\in[\![C',D']\!]$. Now, application of rule Q_3 yields $(q,C,D)\overset{f}{\Lleftarrow}(q,C',D')$.
- Induction step.
  - Suppose $\phi=\phi_1\wedge\phi_2$. Then $(q,w)\models_v\phi_1$ and $(q,w)\models_v\phi_2$. By applying the induction hypothesis on ϕ_1, we derive that there exist C'',D'' such that $(q,C,D)\overset{\phi_1}{\Lleftarrow}(q,C'',D'')$ and $(v,w)\in[\![C'',D'']\!]$. Applying the induction hypothesis on ϕ_2 yields the existence of C',D' such that $(q,C'',D'')\overset{\phi_2}{\Lleftarrow}(q,C',D')$ and $(v,w)\in[\![C',D']\!]$. Then by application of rule Q_4 also $(q,C,D)\overset{\phi_1\wedge\phi_2}{\Lleftarrow}(q,C',D')$.

- Suppose $\phi = \phi_1 \vee \phi_2$. Then $(q, w) \models_v \phi_1$ or $(q, w) \models_v \phi_2$. Suppose that $(q, w) \models_v \phi_1$. The induction hypothesis yields the existence of C', D' such that $(q, C, D) \overset{\phi_1}{\Lleftarrow} (q, C', D')$ and $(v, w) \in [\![C', D']\!]$. Then, by application of rule Q_5 $(q, C, D) \overset{\phi_1 \vee \phi_2}{\Lleftarrow} (q, C', D')$. The case $(q, w) \models \phi_2$ is similar (using rule Q_6).

"$\supseteq$": Assume $(q, C, D) \overset{\phi}{\Lleftarrow} (q, C', D')$ and $(v, w) \in [\![C', D']\!]$. By induction on the structure of the derivation of $\overset{\phi}{\Lleftarrow}$, we establish that $(v, w) \in [\![C, D]\!]$ and $(q, w) \models_v \phi$.

- Base cases. The derivation consists of a single step r.
  - $r = Q_1$. Then $\phi = q$, $C = C'$, $D = D'$. Trivially $(v, w) \in [\![C, D]\!]$ and $(q, w) \models_v q$.
  - $r = Q_2$. Similar to the previous case.
  - $r = Q_3$. Suppose $\phi = f$ with f a simple guard. Then $(C, D) \overset{f}{\Lleftarrow}_c (C', D')$. This means that there are C'', D'' with $(C, D) \overset{f}{\Lleftarrow} (C'', D'')$ and $(C'', D'') \rightarrow_c (C', D')$. By Lemma 6.3.8 we have $(v, w) \in [\![C'', D'']\!]$. Then we have by Lemma 6.3.7 that $(v, w) \models f$ and $(v, w) \in [\![C, D]\!]$.
- Induction step. Consider the last rule r in the derivation of $(q, C, D) \overset{\phi}{\Lleftarrow} (q, C', D')$.
  - $r = Q_4$. Then $\phi = \phi_1 \wedge \phi_2$ and $(q, C, D) \overset{\phi_1}{\Lleftarrow} (q, C'', D'')$ and $(q, C'', D'') \overset{\phi_2}{\Lleftarrow} (q, C', D')$ for some C'', D''. Applying the induction hypothesis to the second statement yields that $(q, w) \models_v \phi_2$ and $(v, w) \in [\![C'', D'']\!]$. Then applying the induction hypothesis to the first statement yields $(q, w) \models_v \phi_1$ and $(v, w) \in [\![C, D]\!]$. Then also $(q, w) \models_v \phi_1 \wedge \phi_2$.
  - $r = Q_5$. Then $\phi = \phi_1 \vee \phi_2$. Then $(q, C, D) \overset{\phi_1}{\Lleftarrow} (q, C', D')$. By induction hypothesis we have $(q, w) \models_v \phi_1$ and $(v, w) \in [\![C, D]\!]$.
  - $r = Q_6$. Similarly to the previous case.

□

6.3.5 Algorithm

We are now in a position to present our model checking algorithm for parametric timed automata. The algorithm displayed in Fig. 6.6 describes how our tool explores the symbolic state space and searches for constraints on the parameters for which a reachability property $\exists\Diamond\phi$ holds in a PTA $\mathcal{A}$.

In the algorithm, we use inclusion between symbolic states defined by

$$(q, C, D) \subseteq (q', C', D') \quad \triangleq \quad q = q' \wedge [\![C, D]\!] \subseteq [\![C, D']\!].$$

Note that whenever a triple (q, C, D) ends up in one of the lists maintained by the algorithm, (C, D) is a constrained PDBM in canonical form. This fact, in combination with Lemma 6.3.6, gives decidability of the inclusion operation. Our search algorithm explores the symbolic semantics in an "intelligent" manner, and stops whenever it reaches a state

REACHABLE ($\mathcal{A}$, ϕ)

RESULT := $\emptyset$,PASSED := $\emptyset$,WAITING := $\{(q_0, C, D) \mid (C_0, \mathsf{E}\uparrow) \overset{I(q_0)}{\Lleftarrow}_c (C, D)\}$
while WAITING $\neq \emptyset$ **do**
  select (q, C, D) from WAITING
  RESULT := RESULT $\cup \{(q', C', D') \mid (q, C, D) \overset{\phi}{\Lleftarrow} (q', C', D')\}$
  FALSE := $\{(q', C', D') \mid (q, C, D) \overset{\neg\phi}{\Lleftarrow} (q', C', D')\}$
  for each (q', C', D') in FALSE **do**
    if for all (q'', C'', D'') in PASSED: $(q', C', D') \not\subseteq (q'', C'', D'')$ **then**
      add (q', C', D') to PASSED
      for each (q'', C'', D'') such that $(q', C', D') \rightarrow (q'', C'', D'')$ **do**
        WAITING := WAITING $\cup \{(q'', C'', D'')\}$
return RESULT

Figure 6.6: The parametric model checking algorithm

whose semantics is contained in the semantics of a state that has been encountered before. Despite this, our algorithm need not terminate.

If it terminates, the result returned by the algorithm is a set of satisfiable symbolic states, all of which satisfy ϕ, for any valuation of the parameters and clocks in the state.

Theorem 6.3.22 *Suppose* (q, C, D) *is in the result set returned by* REACHABLE *($\mathcal{A}$, ϕ). Then* (C, D) *is satisfiable. Moreover, for all* $(v, w) \in [\![C, D]\!]$*,* (q, w) *is a reachable state of* $[\![\mathcal{A}]\!]_v$ *and* $(q, w) \models_v \phi$*.*

PROOF: It is easy to see that all the symbolic states returned by the algorithm are satisfiable: the only operation that may modify the constraint set is adding a guard, but this will never lead to unsatisfiable constraint sets. Since all constrained PDBMs returned by the algorithm are in canonical form, they are all satisfiable by Lemma 6.3.5.

Suppose that $(v, w) \in [\![C, D]\!]$. By a straightforward inductive argument, using Lemmas 6.3.15, 6.3.16 and 6.3.21, it follows that (q, w) is a reachable state of $[\![\mathcal{A}]\!]_v$ and $(q, w) \models_v \phi$. $\square$

For invariance properties $\forall\Box\phi$, our tool runs the algorithm on $\neg\phi$, and the result is then a set of symbolic states, none of which satisfies ϕ. The answer to the model checking problem, stated in Section 6.2.2, is obtained by taking the union of the constraint sets from all symbolic states in the result of the algorithm; in the case of an invariance property we take the complement of this set.

A difference between the above algorithm and the standard timed model checking algorithm is that we continue the exploration until either no more new states are found or all paths end in a state satisfying the property. This is because we want to find all the possible constraints on the parameters for which the property holds. Also, the operations on non-parametric DBMs only change the DBM they are applied to, whereas in our case, we may end up with a set of new PDBMs and not just one.

Some standard operations on symbolic states that help in exploring as little as possible, have also been implemented in our tool for parametric symbolic states. Before starting the state space exploration, our implementation determines the *maximal constant* for each clock. This is the maximal value to which the clock is compared in any guard or invariant in the PTA. When the clock value grows beyond this value, we can ignore its real value. This enables us to identify many more symbolic states, and helps termination. In fact, for unparameterized timed automata this trick *guarantees* termination [AD94, Alu98].

6.4 Lower Bound / Upper Bound Automata

This section introduces the class of *lower bound/upper bound (L/U) automata* and describes several (rather intuitive) observations that simplify the parametric model checking problem for PTAs in this class. Our results use the possibility to eliminate parameters in certain cases. This is a relevant issue, because the complexity of parametric model checking grows very fast in the number of parameters. Moreover, our observations yield some decidability results for L/U automata, where the corresponding problems are undecidable for general PTAs. The applicability of the results is illustrated by the verification of Fischer's algorithm.

6.4.1 Lower bound/Upper bound Automata

Informally, each parameter in an L/U automaton $\mathcal{A}$ occurs either as a lower bound in the invariants and guards of $\mathcal{A}$ or as an upper bound, but never as both. For instance, p is an upper bound parameter in $x - y < 2p$. Lower bound parameters are for instance q and q' in $y - x > q + 2q'$ ($\equiv x - y < -q - 2q'$) and in $x - y < 2p - q - 2q'$. A PTA containing both the guards $x - y \leq p - q$ and $z < q - p$ is not an L/U automaton.

Definition 6.4.1 A parameter $p_i \in P$ is said to *occur* in the linear expression $e = t_0 + t_1 \cdot p_1 + \cdots t_n \cdot p_n$ if $t_i \neq 0$; p_i *occurs positively* in e if $t_i > 0$ and p_i *occurs negatively* in e if $t_i < 0$. A *lower bound parameter* of a PTA $\mathcal{A}$ is a parameter that only occurs negatively in the expressions of $\mathcal{A}$ and an *upper bound parameter* of $\mathcal{A}$ is a parameter that only occurs positively in $\mathcal{A}$. We call $\mathcal{A}$ a *lower bound/upper bound (L/U) automaton* if every parameter occurring in $\mathcal{A}$ is either a lower bound parameter or an upper bound parameter.

Example 6.4.2 The PTA in Fig. 6.7 is an L/U automaton, where *min* is a lower bound parameter and *max* is an upper bound parameter. The model of Fischer's algorithm in Fig. 6.3 is also an L/U automaton. Here *min_rw* and *min_delay* are the lower bound parameters and *max_rw* and *max_delay* are the upper bound parameters.

From now on, we work with a fixed set $L = \{l_1, \dots l_K\}$ of lower bound parameters and a fixed set $U = \{u_1, \dots u_M\}$ of upper bound parameters with $L \cap U = \emptyset$ and $L \cup U = P$. Furthermore, we consider, apart from parameter valuations, also *extended parameter valuations*. Intuitively, an extended parameter valuation is a parameter valuation with values in $\mathbb{R}^{>0} \cup \{\infty\}$, rather than in $\mathbb{R}^{>0}$. Extended parameter valuations are useful in certain cases to solve the verification problem (over non-extended valuations) stated in Section 6.2.3. Working with extended parameter valuations may cause the evaluation of an expression to be undefined. For example, the expression $e[v]$ is not defined for $e = p - q$ and $v(p) = v(q) = \infty$. We therefore require that an extended parameter valuation does not assign the value ∞ to both a lower bound parameter and an upper bound parameter. Then we can easily extend notions $e[v]$, $(v, w) \models e$ and $\mathcal{A}[v]$ (defined in Section 6.2) to extended valuations. Here, we use the conventions that $0 \cdot \infty = 0$, that $x - y \prec \infty$ evaluates to true and $x - y \prec -\infty$ to false. In particular, we have $[\![\mathcal{A}]\!]_v = [\![\mathcal{A}[v]]\!]$ for extended valuations v and L/U automata $\mathcal{A}$. Moreover, we extend the orders $\sim$ to $\mathbb{R} \cup \{\infty\}$ in the usual way and we extend them to extended parameter valuations via point wise extension (i.e. $v \sim v'$ iff $v(p) \sim v'(p)$ for all $p \in P$). We denote an extended valuation of an L/U automaton by a pair (λ, μ), which equals the function λ on the lower bound parameters and μ on the upper bound parameters. We write 0 and ∞ for the functions assigning respectively 0 and ∞ to each parameter.

The following proposition is based on the fact that weakening the guards in $\mathcal{A}$ (i.e. decreasing the lower bounds and increasing the upper bounds) yields an LTS whose reachable states include those of $\mathcal{A}$. Dually, strengthening the guards in $\mathcal{A}$ (i.e. increasing the lower bounds and decreasing the upper bounds) yields an LTS whose reachable states are a subset of those of $\mathcal{A}$. The result crucially depends on the fact that state formulae (by definition) do not contain parameters. The usefulness of this result (and of several other results in this section) lies in the fact that the satisfaction of a property for infinitely many extended parameter valuations (λ', μ') is reduced to its satisfaction for a single valuation (λ, μ).

Proposition 6.4.3 *Let $\mathcal{A}$ be an L/U automaton and ϕ a state formula. Then*

1. $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond\phi \iff \forall\lambda' \leq \lambda, \mu \leq \mu' : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \exists\Diamond\phi.$
2. $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \forall\Box\phi \iff \forall\lambda \leq \lambda', \mu' \leq \mu : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi.$

PROOF: (sketch) The "$\Longleftarrow$" parts of both statements are trivial. The crucial observation for both "$\Longrightarrow$" parts is the following. For all linear expressions e in $\mathcal{A}$ and all extended parameter valuations (λ, μ), (λ', μ') with $\lambda' \leq \lambda$ and $\mu \leq \mu'$, we have that $e[\lambda, \mu] \leq e[\lambda', \mu']$. Therefore, if $((\lambda, \mu), w) \models x - y \prec e$, then $((\lambda', \mu'), w) \models x - y \prec e$.
□

The following example illustrates how Proposition 6.4.3 can be used to eliminate parameters in L/U automata.

Example 6.4.4 Reconsider the L/U automaton in Fig. 6.7. Location S_1 is reachable irrespective of the parameter values. By setting the parameter min to ∞ and max to 0, one checks with a non-parametric model checker that $\mathcal{A}[(\infty, 0)] \models \exists\Diamond S_1$. Then Proposition 6.4.3(1) (together with $[\![\mathcal{A}]\!]_v = [\![\mathcal{A}[v]]\!]$) yields that S_1 is reachable in $[\![\mathcal{A}]\!]_{(\lambda,\mu)}$ for all extended parameter valuations $0 \leq \lambda, \mu \leq \infty$.

Clearly, $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ iff $\lambda(min) \leq \mu(max) \wedge \lambda(min) < \infty$. We will see in this running example how we can verify this property completely by non-parametric model checking. Henceforth, we construct the automaton $\mathcal{A}'$ from $\mathcal{A}$ by substituting the parameter max by the parameter min yielding an (non L/U) automaton with one parameter, min. The next example shows that $[\![\mathcal{A}']\!]_v \models \exists\Diamond S_2$ for all valuations v, which essentially means that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ for all λ, μ such that $\mu(max) = \lambda(min) < \infty$. From this fact, Proposition 6.4.3(1) concludes that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ for all λ, μ with $\lambda(min) \leq \mu(max)$ and $\lambda(min) < \infty$.

The question whether there exists a (non-extended) parameter valuation such that a given location q is reachable, is known as the *emptiness problem* for PTAs. In [AHV93], it is shown that the emptiness problem is undecidable for PTAs with three clocks or more. The following proposition implies that we can solve the emptiness problem for an L/U automaton $\mathcal{A}$ by only considering $\mathcal{A}[(0, \infty)]$, which is a non-parametric timed automaton. Since reachability is decidable for timed automata ([AD94]), the emptiness problem is decidable for L/U automata. Then it follows that the dual problem is also decidable for L/U automata. This is the

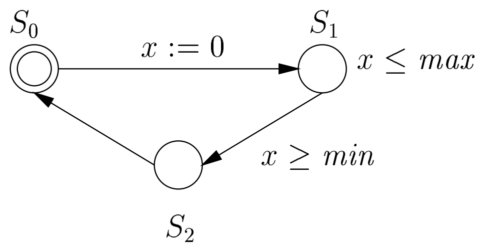


Figure 6.7: Reducing parametric to non-parametric model checking

universality problem for invariance properties, asking whether an invariance property holds for all parameter valuations.

Proposition 6.4.5 *Let $\mathcal{A}$ be an L/U automaton with location q. Then $\mathcal{A}[(0,\infty)] \models \exists\Diamond q$ if and only if there exists a (non-extended) parameter valuation (λ, μ) such that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond q$.*

PROOF: The "only if" part is an immediate consequence of Proposition 6.4.3(1) and the fact that $[\![\mathcal{A}[(0,\infty)]]\!] = [\![\mathcal{A}]\!]_{(0,\infty)}$. For the "if" part, assume that α is a run of $[\![\mathcal{A}[(0,\infty)]]\!]$ that reaches the location q. Let T' be the smallest constant occurring in $\mathcal{A}$ and let T be the maximum clock value occurring in α. (More precisely, if $\alpha = s_0 a_1 s_1 a_2 \ldots a_N s_N$ and $s_i = (q_i, w_i)$, then $T = \max_{i \leq N, x \in X} w_i(x)$; T' compensates for negative constants t_0 in expressions e of $\mathcal{A}$.) Now, take $\lambda(l_j) = 0$ and $\mu(u_j) = T + |T'| + 1$. Let $i \leq N$ and $g = x - y \prec e$ be the invariant associated with a state s_i occurring in α or the guard associated with the i^{th} transition taken by α. One easily shows that, since $w_i(x) - w_i(y) \prec e[0,\infty]$, also $w_i(x) - w_i(y) \prec e[\lambda, \mu]$, that is $((\lambda, \mu), w_i) \models g$. Hence, α is a run of $[\![\mathcal{A}]\!]_{(\lambda,\mu)}$, so $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond q$.
□

Corollary 6.4.6 *The emptiness problem is decidable for L/U automata.*

Definition 6.4.7 A PTA $\mathcal{A}$ is *fully parametric* if clocks are only reset to 0 and every linear expression in $\mathcal{A}$ of the form $t_1 \cdot p_1 + \cdots + t_n \cdot p_n$, where $t_i \in \mathbb{Z}$.

The following proposition is basically the observation in [AD94], that multiplication of each constant in a timed automaton and in a system property with the same positive factor preserves satisfaction.

Proposition 6.4.8 *Let $\mathcal{A}$ be fully parametric PTA. Then for all parameter valuations v and all system properties ψ*

$$[\![\mathcal{A}]\!]_v \models \psi \iff \forall t \in \mathbb{R}^{>0} : [\![\mathcal{A}]\!]_{t \cdot v} \models t \cdot \psi,$$

where $t \cdot v$ denotes the valuation $p \mapsto t \cdot v(p)$ and $t \cdot \psi$ the formula obtained from ψ by multiplying each number in ψ by t.

PROOF: It is easy to see that for all $t \in \mathbb{R}^{>0}$, $\alpha = s_0 a_1 s_1 a_2 \dots a_N s_N$ with $s_i = (q_i, w_i)$ is a run of $[\![\mathcal{A}]\!]_v$ if and only if $s'_0 a_1 s'_1 \dots a_N s'_N$ is a run of $[\![\mathcal{A}]\!]_{t \cdot v}$, where $s'_i = (q_i, t \cdot w_i)$ and $t \cdot w_i$ denotes $x \mapsto t \cdot w_i(x)$. $\square$

Then for fully parametric PTAs with one parameter and system properties ψ without constants (except for 0), we have $[\![\mathcal{A}]\!]_v \models \psi$ for all valuations v of P if and only if both $\mathcal{A}[0] \models \psi$ and $\mathcal{A}[1] \models \psi$. The need for a separate treatment of the value 0 is illustrated by the (fully parametric) automaton with a single transition equipped with the guard $x < p$. The target location of the transition is reachable for any value of p, except for $p = 0$.

Corollary 6.4.9 *For a fully parametric PTA $\mathcal{A}$ with one parameter, a constraint set C and a property ψ without constants (except 0), it is decidable whether $\forall v \in [\![C]\!] : [\![\mathcal{A}]\!]_v \models \psi$.*

Example 6.4.10 The PTA $\mathcal{A}'$ mentioned in Example 6.4.4 is a fully parametric timed automaton and the property $\exists\Diamond S_2$ is without constants. We establish that $\mathcal{A}'[0] \models \exists\Diamond S_2$ and $\mathcal{A}'[1] \models \exists\Diamond S_2$. Then Proposition 6.4.8 implies that $\mathcal{A}'[v] \models \exists\Diamond S_2$ for all v. As shown in Example 6.4.4, this implies that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ for all λ, μ with $\lambda(min) = \mu(max) < \infty$.

In the running example, we would like to use the same methods as above to verify that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \not\models \exists\Diamond S_2$ if $\lambda(min) > \mu(max)$. However, we can not take $min = max$ in this case, since the bound in the constraint is a strict one. The following definition and results allows us to move the strictness of a constraint into the PTA.

Definition 6.4.11 Let $P' \subseteq P$ be a set of parameters. Define $\mathcal{A}^{<}_{P'}$ as the PTA obtained from $\mathcal{A}$ by replacing every inequality $x - y \leq e$ in $\mathcal{A}$ by a strict inequality $x - y < e$, provided that e contains at least one parameter from P'. Similarly, define $\mathcal{A}^{\leq}_{P'}$ as the PTA obtained from $\mathcal{A}$ by replacing every inequality $x - y < e$ by a non–strict inequality $x - y \leq e$, provided that e contains at least one parameter from P'. For $\prec\, = <, \leq$, write $\mathcal{A}^{\prec}$ for $\mathcal{A}^{\prec}_{P}$. Moreover, define $v \prec_{P'} v'$ by $v(p) \prec v'(p)$ if $p \in P'$ and $v(p) = v'(p)$ otherwise.

Proposition 6.4.12 *Let $\mathcal{A}$ be an L/U automaton. Then*

1. $[\![\mathcal{A}^{\leq}]\!]_{(\lambda,\mu)} \models \exists\Diamond\phi \implies \forall \lambda' < \lambda, \mu < \mu' : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \exists\Diamond\phi$.
2. $[\![\mathcal{A}^{<}]\!]_{(\lambda,\mu)} \models \forall\Box\phi \iff \forall \lambda < \lambda', \mu' < \mu : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi$.

PROOF:

1 Let e be a linear expression occurring in $\mathcal{A}$. Then we can write $e = t_0 + e_1 + e_2$, where $t_0 \in \mathbb{Z}$, e_1 is an expression over the upper bound parameters and e_2 an expression over the lower bound parameters. Then we have

$$\begin{aligned}
&\mu \leq \mu' \implies e_1[\mu] \leq e_1[\mu'],\\
&\lambda' \leq \lambda \implies e_2[\lambda'] \leq e_2[\lambda],\\
&\lambda' \leq \lambda, \mu \leq \mu' \implies e[(\lambda,\mu)] \leq e[(\lambda',\mu')].
\end{aligned}$$

If there is at least one parameter occurring respectively in e_1 or e_2 then respectively

$$\begin{aligned}
&\mu < \mu' \implies e_1[\mu] < e_1[\mu']\\
&\lambda' < \lambda \implies e_2[\lambda] < e_2[\lambda'].
\end{aligned}$$

Thus if there is at least one parameter occurring in e, then

$$\lambda' < \lambda, \mu < \mu' \implies e[(\lambda,\mu)] < e[(\lambda',\mu')].$$

Now, let (λ,μ) be an extended valuation. Let $g \equiv x - y \prec e$ be a simple guard occurring in $\mathcal{A}^{\leq}$ and let $g' \equiv x - y \prec' e$ be the corresponding guard in $\mathcal{A}$. Assume that $(w,(\lambda,\mu)) \models g$, i.e. $w(x) - w(y) \prec e[(\lambda,\mu)]$. We show that $(w,(\lambda,\mu)) \models g'$. We distinguish two cases.

case 1: There exists a parameter occurring in e. Then $w(x) - w(y) \prec e[(\lambda,\mu)] < e[(\lambda',\mu')]$. Then certainly $(w,(\lambda,\mu)) \models g' \equiv x - y \prec' e$.

case 2: The expression e does not contain any parameter. Then $g' \equiv g$ and hence $(w,(\lambda,\mu)) \models g'$.

Now it easily follows that every run of $[\![\mathcal{A}^{\leq}]\!]_{(\lambda,\mu)}$ is also a run of $[\![\mathcal{A}]\!]_{(\lambda',\mu')}$. Thus, if a state satisfying ψ is reachable in $[\![\mathcal{A}^{\leq}]\!]_{(\lambda,\mu)}$ then it is also reachable in $[\![\mathcal{A}]\!]_{(\lambda',\mu')}$.

2, $\implies$: This follows from statement (1) of this proposition: assume that $[\![\mathcal{A}^{<}]\!]_{(\lambda,\mu)} \models \forall\Box\phi$ and let λ', μ' be such that $\lambda < \lambda'$, $\mu' < \mu$. Since $[\![\mathcal{A}^{<}]\!]_{(\lambda,\mu)} \not\models \exists\Diamond\neg\phi$, we have

$$\neg\forall\lambda'' < \lambda', \mu' < \mu'' : [\![\mathcal{A}^{<}]\!]_{(\lambda'',\mu'')} \models \exists\Diamond\neg\phi.$$

Then contraposition of statement (1) together with $(\mathcal{A}^{<})^{\leq} = \mathcal{A}^{\leq}$ yields $[\![\mathcal{A}^{\leq}]\!]_{(\lambda',\mu')} \not\models \exists\Diamond\neg\phi$. As $\mathcal{A}$ imposes stronger bounds than $\mathcal{A}^{\leq}$, also $[\![\mathcal{A}]\!]_{(\lambda',\mu')} \not\models \exists\Diamond\neg\phi$, which means $[\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi$.

2, $\Longleftarrow$: Let (λ,μ) be an extended valuation and assume that $[\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi$ for all $\lambda' > \lambda$, $\mu' < \mu$. Assume that α is a run of $[\![\mathcal{A}^{<}]\!]_{(\lambda,\mu)}$. We construct $\lambda' > \lambda$ and $\mu' < \mu$ such that α is also a run of $[\![\mathcal{A}]\!]_{(\lambda',\mu')}$. (Then we are done: since $[\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi$, the last state of α satisfies ϕ. Hence every reachable state of $[\![\mathcal{A}]\!]_{(\lambda,\mu)}$ satisfies ϕ, i.e. $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \forall\Box\phi$.)

We use the following notation. We write $v = (\lambda,\mu)$ and $v' = (\lambda',\mu')$. For a run α, we write $\alpha = s_0 a_1 s_1 a_2 \ldots a_N s_N$ with $s_i = (q_i, w_i)$, $I(q_i) = \wedge_j^{J'} I_{ij}$, $I_{ij} = x_{ij} \prec_{ij} E_{ij}$. As α is a run, either $a_{i+1} \in \mathbb{R}$ or there exists a transition $q_i \xrightarrow{g_{i-1}, a_{i+1}, r_{i+1}} q_{i+1}$ for each i, $1 \leq i \leq N$. We write the guard on this transition by $g_i = \wedge_j^{J} g_{ij}$, $g_{ij} = x_{ij} - y_{ij} \prec_{ij} e_{ij}$. Finally, if g is a guard or invariant in $\mathcal{A}$, then we denote the corresponding guard or invariant in $\mathcal{A}^{<}$ by $g^{<}$, i.e. the guard that is obtained as in Definition 6.4.11.

If neither the guards g_{ij} nor the invariants I_{ij} contains a parameter, then we can take v' arbitrarily and we have that α is a run of $[\![\mathcal{A}]\!]_{v'}$. Therefore, assume that at least one of the guards g_{ij} or invariants I_{ij} contains a parameter. Then, by definition of $\mathcal{A}^{<}$, this guard or invariant contains a strict bound. In this case, we construct $\lambda' > \lambda$ and $\mu' < \mu$ such that $w_i(x - y) < e[(\lambda',\mu')] < e[(\lambda,\mu)]$ if $g = x - y < e$ is an invariant I_{ij} or guard g_{ij} as above. Informally, we use the minimum "distance" $e[(\lambda,\mu)] - w_i(x - y)$ occurring in α to slightly increase the lower bounds and slightly decrease the upper bounds yielding $\lambda < \lambda'$ and $\mu < \mu'$.

Let

$$\begin{aligned} T_0 &= \min_{i \leq N, j \leq J'} \{E_{ij}[v] - w_i(x_{ij}) \quad | \quad \prec_{ij} = <\}, \\ T_1 &= \min_{i \leq N, j \leq < J} \{e_{ij}[v] - w_i(x_{ij} - y_{ij}) \quad | \quad \prec_{ij} = <\}, \\ 0 < &T < \min \{T_0, T_1\}, \end{aligned}$$

with the convention that $\min \emptyset = \infty$. At least one of the inequalities $\prec_i j$ is strict, since at least one of the guards contains a parameter. Hence $T_0 < \infty$ or $T_1 < \infty$. Since $(v, w_i) \models I_{ij} \wedge g_{ij}$, we have we have that $T_0 > 0$ and $T_1 > 0$. Hence $\infty > \min \{T_0, T_1\} > 0$ and the requested T exists. The crucial property is that if $g_{ij} \equiv x_{ij} - y_{ij} < e_{ij}$ or $g_{ij} \equiv x_{ij} - y_{ij} < E_{ij}$ we have respectively

$$\begin{aligned} T &< e_{ij}[v] - w_i(x_{ij} - y_{ij}) \\ T &< E_{ij}[v] - w_i(x_{ij} - y_{ij}). \end{aligned}$$

Now, let T' be the sum of the constants appearing in the guards and invariants that appear in the run α i.e.

$$T' = \sum_{i \leq N, j \leq J'} \mathrm{sum\_of\_const(E_{ij})} + \sum_{\mathrm{i \leq n, j \leq J}} \mathrm{sum\_of\_const(e_{ij})},$$

where $\mathrm{sum\_of\_const}(\mathrm{t_0 + t_1 \cdot p_1 + \cdots + t_n \cdot p_n}) = |\mathrm{t_1}| + \cdots + |\mathrm{t_n}|$. Since at least one of the guards or invariants contains a parameter, we have $T' > 0$.

Now, take $v' = (\lambda + \frac{T}{T'}, \mu - \frac{T}{T'})$ and consider $g_{ij} \equiv x_{ij} - y_{ij} \prec_{ij} e_{ij}$. We claim that $(v', w_i) \models g_{ij}$.

case 1: The expression g_{ij} does not contain any parameter. Then $g_{ij} = g_{ij}^{<}$ and $e_{ij}[v] = e_{ij}[v']$. Since $(w_i, v) \models g_{ij}$, also $(w_i, (v')) \models g_{ij}^{<}$.

case 2: There exists a parameter occurring in e. We can write $e = t_0 + t_1 \cdot u_1 + \cdots + t_M \cdot u_M - t'_1 \cdot l_1 - \cdots - t'_K \cdot l_K$, with $t_i \geq 0$, $t'_i \geq 0$ for $i > 0$. Then

$$\begin{aligned} e_{ij}[v'] &= t_0 + \sum_{k=1}^{M} t_k \cdot (\mu'_k - \frac{T}{T'}) - \sum_{k=1}^{K} t_k \cdot (\lambda'_k + \frac{T}{T'}) \\ &= (t_0 + \sum_{k=1}^{M} t_k \cdot \mu'_k - \sum_{k=1}^{K} t_k \cdot \lambda'_k) - \frac{T}{T'} \cdot (\sum_{k=1}^{M} t_k + \sum_{k=1}^{K} t'_k) \\ &\geq e_{ij}[v] - T \\ &> e_{ij}[v] - (e_{ij}[v] - w_i(x_{ij} - y_{ij})) \\ &= w_i(x_{ij} - y_{ij}). \end{aligned}$$

Therefore $(w_i, v') \models x_{ij} - y_{ij} < g_{ij}^{<}$ and then also $(w_i, v') \models x_{ij} - y_{ij} \prec_{ij} g_{ij}^{<}$.

Combining the cases (1) and (2) yields that for all i, j, $(w_i, v') \models x_{ij} - y_{ij} \prec_{ij} g_{ij}^{<}$. Similarly, one proves that $(w_i, v') \models x_{ij} - y_{ij} \prec_{ij} I_{ij}$. Therefore, α is a run of $[\![\mathcal{A}]\!]_{(\lambda', \mu')}$.

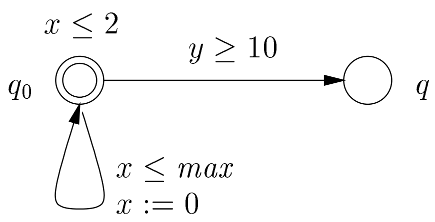


Figure 6.8: The converse of Proposition 6.4.12(1) does not hold.

□

The previous result concerns the automaton that is obtained when all the strict inequalities in the automaton are changed into nonstrict ones (or the other way around). Sometimes, we want to "toggle" only some of the inequalities. Then the following result can be applied.

Corollary 6.4.13 *Let $\mathcal{A}$ be an L/U automaton and $P' \subseteq P$.*

1. $[\![\mathcal{A}^{\leq}_{P'}]\!]_{(\lambda,\mu)} \models \exists\Diamond\phi \implies \forall\lambda' <_{P'} \lambda, \mu <_{P'} \mu' : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \exists\Diamond\phi.$
2. $[\![\mathcal{A}^{<}_{P'}]\!]_{(\lambda,\mu)} \models \forall\Box\phi \iff \forall\lambda <_{P'} \lambda', \mu' <_{P'} \mu : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi.$

PROOF: Let (λ, μ) be an extended valuation. Let $\mathcal{A}_0$ be the automaton obtained from $\mathcal{A}$ by substituting p by $(\lambda, \mu)(p)$ for every $p \notin P'$. Then $[\![\mathcal{A}^{<}_{P'}]\!]_{(\lambda,\mu)} = [\![\mathcal{A}^{<}_0]\!]_{(\lambda,\mu)}$ and $[\![\mathcal{A}^{\leq}_{P'}]\!]_{(\lambda,\mu)} = [\![\mathcal{A}^{\leq}_0]\!]_{(\lambda,\mu)}$. Now the result follows by applying Proposition 6.4.12 to $\mathcal{A}_0$. □

The following example shows that the converse of Proposition 6.4.12(1) does not hold.

Example 6.4.14 Consider the automaton $\mathcal{A}$ in Fig. 6.8. Recall that the clocks x and y are initially 0. Then $\mathcal{A} = \mathcal{A}^{\leq}$ and the location q is reachable if $max > 0$ but not if $max = 0$. This is so because if $max = 0$, then clock y is never augmented. Thus $\forall\lambda' < 0, 0 < \mu'$: $[\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \exists\Diamond\phi$, but not $[\![\mathcal{A}^{\leq}]\!]_{(0,0)} \models \exists\Diamond\phi$.

We believe that the class of L/U automata can be very useful in practice. Several examples known from the literature fall into this class, or can be modeled slightly differently to achieve this. We mention the root contention protocol [IEE96], Fischer's mutual exclusion algorithm [Lam87], the (toy) rail road crossing example from [AHV93], the bounded retransmission protocol (when considering fixed values for the integer variables) and the biphase mark protocol (with minor adaptations). Moreover, the time constrained automata models of [MMT91, Lyn96] can be encoded straightforwardly into L/U automata.

We expect that quite a few other distributed algorithms and protocols can be modeled with L/U automata, since it is natural that the duration of an event (such as the communication delay in a channel, the computation time needed to produce a result, the time required to open the gate in a rail road crossing) lies between a lower bound and an upper bound. These bounds are often parameters of the system.

The next section and Section 6.5 show that the techniques discussed in this section to eliminate parameters in L/U models reduce the verification effort significantly and possibly lead to a completely non-parametric model. Finally, we remark that similar techniques can be applied to lower bound and upper bound parameters, when they are present in a general PTA, which may thus also have parameters which are neither lower bound nor upper bound parameters and to which the techniques cannot be applied.

6.4.2 Verification of Fischer's mutual exclusion protocol

In this section, we apply the results from the previous section to verify the Fischer protocol described in Section 6.2.4. We establish the sufficiency of the protocol constraints by non-parametric model checking and the necessity of the constraints by eliminating three of the four parameters.

We also tried to use the prototype to verify the protocol model without any substitutions or changing of bounds, but this did not terminate within 20 hours. Since we observed that the constraint lists of the states explored kept on growing, we suspected that this experiment would not terminate at all. (Recall that parametric verification is undecidable.) Verification of the reduced models took only 2 seconds.

Now, consider the Fischer protocol model from Section 6.2.4 again. In this section, we analyze a system $\mathcal{A}$ consisting of two parallel processes P_1 and P_2. It is clear that $\mathcal{A}$ is a fully parametric L/U automaton: min_rw and min_delay are lower bound parameters and max_rw and max_delay upper bound parameters.

The mutual exclusion property is expressed by the formula $\Phi_{ME} \equiv \forall\Box\neg(P_1.cs \wedge P_2.cs)$. Recall that, when assuming the basic constraints $B_{ME} \equiv 0 \leq min\_rw < max\_rw \wedge 0 \leq min\_delay < max\_delay$, mutual exclusion is guaranteed for the parameter values satisfying $C_{ME} \equiv max\_rw \leq min\_delay$. Thus, we will prove that $v \models B_{ME} \implies (\llbracket\mathcal{A}\rrbracket_v \models \Phi_{ME} \iff v \models C_{ME})$, for all valuations v.

Sufficiency of the Constraints

We show that the constraints assure mutual exclusion, that is

$$\text{if } v \models C_{ME} \wedge B_{ME}, \text{ then } \mathcal{A}[v] \models \Phi_{ME}.$$

We perform the substitution

$$min\_rw \mapsto 0,\, max\_delay \mapsto \infty,\, min\_delay \mapsto max\_rw$$

to obtain a fully parametric automaton $\mathcal{A}'$ with one parameter, max_rw. We have established by non-parametric model checking that $\mathcal{A}'[0] \models \Phi_{ME}$ and $\mathcal{A}'[1] \models \Phi_{ME}$. Now Proposition 6.4.8 yields that $\llbracket\mathcal{A}'\rrbracket_v \models \Phi_{ME}$ for all valuations v (where only the value of max_rw matters). This means that $\llbracket\mathcal{A}\rrbracket_v \models \Phi_{ME}$ if $v(min\_rw) = 0$, $v(max\_rw) = v(min\_delay)$ and $v(max\_delay) = \infty$. Then Proposition 6.4.3(2) yields that the invariance property Φ_{ME} also holds if we increase the lower bound parameters min_rw and min_delay and if we decrease the upper bound parameter max_rw. More precisely, Proposition 6.4.3(2) implies that $\llbracket\mathcal{A}\rrbracket_v \models \Phi_{ME}$ for all v with $0 \leq v(min\_rw)$, $v(max\_rw) \leq v(min\_delay)$ and $v(max\_delay) \leq \infty$. Then, in particular, $\llbracket\mathcal{A}\rrbracket_v \models \Phi_{ME}$ if $v \models C_{ME} \wedge B_{ME}$.

Necessity of the Constraints:

We show that

$$v \models B_{ME} \wedge \neg C_{ME} \implies \mathcal{A}[v] \models \neg\Phi_{ME},$$

i.e. that if $v \models min\_rw < max\_rw \ \wedge \ min\_delay < max\_delay \ \wedge \ min\_delay < max\_rw$, then $\mathcal{A}[v] \models \neg\Phi_{ME} \equiv \exists\Diamond(P_1.cs \wedge P_2.cs)$. We construct the automaton $\mathcal{A}^{\leq}$ and proceed in two steps.

Step 1 Let v_0 be the valuation $v_0(min\_delay) = v_0(max\_delay) = 0$ and $v_0(min\_rw) = v_0(max\_delay) = 1$. By non-parametric model checking we have established that

$$\mathcal{A}^{\leq}[0] \models \neg\Phi_{ME} \tag{6.8}$$

$$\mathcal{A}^{\leq}[v_0] \models \neg\Phi_{ME}. \tag{6.9}$$

We show that it follows that for all v

$$v \models 0 = min\_delay = max\_delay \leq min\_rw = max\_rw \implies \mathcal{A}^{\leq}[v] \models \neg\Phi_{ME}. \tag{6.10}$$

Assume $v \models 0 = min\_delay = max\_delay \leq min\_rw = max\_rw$. Consider $t = v(min\_rw)$. If $v(min\_rw) = 0$, then (6.8) shows that $[\![\mathcal{A}^{\leq}]\!]_v \models \neg\Phi_{ME}$. Therefore, assume $v(min\_rw) > 0$ and consider $\frac{v}{t} \equiv \lambda x.\frac{v(x)}{t}$. It is not difficult to see that

$$\frac{v}{t} \models 0 = min\_delay = max\_delay \leq min\_rw = max\_rw = 1.$$

Therefore, (6.9) yields $[\![\mathcal{A}^{\leq}]\!]_{\frac{v}{t}} \models \neg\Phi_{ME}$. Since $\mathcal{A}^{\leq}$ is a fully parametric PTA, Proposition 6.4.8 yields that $[\![\mathcal{A}^{\leq}]\!]_v \models \neg\Phi_{ME}$.

Step 2 Let $\mathcal{A}'$ be the automaton that is constructed from $\mathcal{A}^{\leq}$ by performing the following substitution $min\_delay \mapsto 1$, $max\_delay \mapsto 1$, $min\_rw \mapsto max\_rw$. By parametric model checking we have established

$$v \models 1 \leq max\_rw \implies [\![\mathcal{A}']\!]_v \models \neg\Phi_{ME}. \tag{6.11}$$

This means that if

$$v \models min\_delay = max\_delay = 1 \leq min\_rw = max\_rw \implies [\![\mathcal{A}^{\leq}]\!]_v \models \neg\Phi_{ME}.$$

By an argument similar to the one we used to prove (6.10), we can use Proposition 6.4.8 to show that

$$v \models min\_delay = max\_delay \leq min\_rw = max\_rw \implies [\![\mathcal{A}^{\leq}]\!]_v \models \neg\Phi_{ME},$$

where now the case $v(min\_delay) = 0$ is covered by Equation (6.10) in Step 1. Then Proposition 6.4.3(1) yields that the reachability property $\neg\Phi_{ME}$ also holds if the values for the lower bounds are decreased and the values for the upper bounds are increased. Note that we may increase max_delay as much as we want; $v(max\_delay)$ may be larger than $v(min\_rw)$. Thus we have

$$\begin{aligned} v \models\ & min\_rw \leq max\_rw \ \wedge \ min\_delay \leq max\_delay \ \wedge \ min\_delay \leq max\_rw \\ & \implies [\![\mathcal{A}^{\leq}]\!]_v \models \neg\Phi_{ME} \end{aligned}$$

and then Proposition 6.4.12 yields that

$$\begin{aligned} v \models\ & min\_rw < max\_rw \ \wedge \ min\_delay < max\_delay \ \wedge \ min\_delay < max\_rw \\ & \implies [\![\mathcal{A}]\!]_v \models \neg\Phi_{ME}. \end{aligned}$$

model from	*initial constraints*	*property*	*Uppaal*	*time*	*memory*
[DKRT97a]	yes	safety1	param	1.3 m	34 Mb
[DKRT97a]	no	safety2	param	11 m	180 Mb
[DKRT97a]	yes	safety2	param	3.5 m	64 Mb

Figure 6.9: Experimental results for the bounded retransmission protocol

We have checked the result formulated in Equation (6.11) with our prototype implementation. The experiment was performed on a SPARC Ultra in 2 seconds CPU time and 7.7 Mb of memory.

The substitutions and techniques used in this verification to eliminate parameters are ad hoc. We believe, however, that more general strategies can be applied. Especially in this case, where the constraints are L/U–like (i.e. can be written in the form $e \prec 0$ such that every p occurring negatively in e is a lower bound parameter and every p occurring positively in e is an upper bound parameter), it should be possible to come up with smarter strategies for parameter elimination.

6.5 Experiments

6.5.1 A Prototype Extension of Uppaal

Based on the theory described in Section 6.3, we have built a prototype extension of Uppaal. In this section, we report on the results of experimenting with this tool.

Our prototype allows the user to give some initial constraints on the parameters. This is particularly useful when explorations cannot be finished due to lack of memory or time resources, or because a non-converging series of constraint sets is being generated. Often, partial results can be derived by observing the constraint sets that are generated during the exploration. Based on partial results, the actual solution constraints can be established in many cases. These partial results can then be checked by using an initial set of constraints.

6.5.2 The Bounded Retransmission Protocol

Description This protocol was designed by Philips for communication between remote controls and audio/video/TV equipment. It is a slight alteration of the well-known alternating bit protocol, to which timing requirements and a bound on the retry mechanism have been added. In [DKRT97a] constraints for the correctness of the protocol are derived by hand, and some instances are checked using Uppaal. Based on the models in [DKRT97a], an automatic parametric analysis is performed in [AAB00], however, no further results are given.

Parametric approach For our analysis, we use the timed automata models from the paper [DKRT97a]. These models typically consist of 7 communicating processes, varying from 2 locations with 4 transitions to 6 locations with 54 transitions, and has 5 clocks and 9 non-clock variables in total. In [DKRT97a] three different constraints are presented based on three properties which are needed to satisfy the safety specification of the protocol. We are only able to check two of these since one of the properties contains a parameter which our prototype version of Uppaal is not able to handle yet.

One of the constraints derived in [DKRT97a] is that $\mathrm{TR} \geq 2 \cdot \mathrm{MAX} \cdot \mathrm{T}_1 + 3 \cdot \mathrm{TD}$, where TR is the timeout of the receiver, T_1 is the timeout of the sender, MAX is the number of resends made by the sender, and TD is the delay of the channel. This constraint is needed to ensure that the receiver does not time out prematurely before the sender has decided to abort transmission. The sender has a parameter SYNC which decides for how long the sender waits until it expects that the receiver has realized a send error and reacted to it. In our parametric analysis we used TR and SYNC as parameters and instantiated the others to fixed values. Using our prototype we did derive the expected constraint $\mathrm{TR} \geq 2 \cdot \mathrm{MAX} \cdot \mathrm{T}_1 + 3 \cdot \mathrm{TD}$. However, we also derived the additional constraint $\mathrm{TR} - 2 \leq \mathrm{SYNC}$ which was not stated in [DKRT97a] for this property. The necessity of this constraint was verified by trying models with different fixed values for the parameters. The full set of constraints derived in [DKRT97a] includes a constraint $\mathrm{TR} \geq \mathrm{SYNC}$ which is based on the property we cannot check. Therefore the error we have encountered is only present in an intermediate result, the complete set of constraints derived is correct. The authors of [DKRT97a] have acknowledged the error and provided an adjusted model of the protocol, for which the additional constraint is not necessary.

The last constraint derived in [DKRT97a] arises from checking that the sender and receiver are not sending messages too fast for the channel to handle. In this model we treat T_1 as a parameter and derive the constraint $\mathrm{T}_1 > 2 \cdot \mathrm{TD}$ which is the same as is derived in [DKRT97a].

6.5.3 Other Experiments

We have experimented with parametric versions of several models from the standard Uppaal distribution, namely Fischer's mutual exclusion protocol, a train gate controller, and a car gear box controller.

In the case of Fischer's protocol (which is the version of the standard Uppaal distribution, and not the one discussed in the rest of this chapter), we parameterized a model with two processes, by turning the bound on the period the processes wait, before entering the critical section, into a parameter. We were able to generate the constraints that ensure the mutual exclusion within 2 seconds of CPU time on a 266 MHz Pentium MMX. Using these constraints as initial constraints and checking that now indeed the mutual exclusion is guaranteed, is done even faster. Fischer's protocol with two processes was also checked in [AAB00], which took about 3 minutes.

6.5.4 Discussion

Our prototype handles parametric versions of bench-mark timed automata rather well. In some cases, the prototype will not generate a converging series of constraints, but in all cases we were able to get successful termination when applying (conjectures of) solution constraints as initial constraints in the exploration. The amount of time and memory used is then in many cases quite reasonable.

From our results it is not easy to draw clear-cut conclusions about the type of parametric model, for which our prototype can successfully generate constraints. It seems obvious from the case studies that the more complicated the model, the larger the effort in memory and time consumption. So it is worthwhile to have small, simple models. However, the danger of non-termination is most present in models which have a lot of behavioral freedom. The most promising direction, therefore, will be to experiment with conjectured solution constraints, and to combine this with the techniques for L/U automata.

6.6 Conclusions

This chapter reports on a parametric extension to the model checker Uppaal. This tool is capable of generating parameter constraints that are necessary and sufficient for a reachability or invariant property to hold for a linear parametric timed automaton. The semantics of the algorithms underlying the tool is given in clean SOS–style rules. Although the work [AHV93] shows that parameter synthesis is undecidable in general, our prototype implementation terminates on many practical verification questions and the run time of the tool is acceptable. Significant reductions are obtained by parameter elimination in L/U automata.

There are several relevant and interesting topics for future research. First of all, serious improvements in the applicability of the tool can be obtained by improving the user interface. Currently, the tool generates many parameter equations whose disjunction is the desired constraint. Since the number of equations are can be quite large, it would be more convenient if the tool could simplify these set of equations. This could for instance be done with reduction techniques for BDDs.

Another relevant issue for parameter analysis is the theoretical investigation of the class of L/U automata. It would for instance be interesting to get more insight which types of problems are decidable for L/U automata and which are not. Furthermore, it would be interesting to investigate the use of L/U automata for synthesizing the constraints, rather than for analyzing given constraints as we did in this chapter. On the practical side, the reduction techniques for L/U automata could be implemented.

CHAPTER 7
A Case Study: the IEEE 1394 Root Contention Protocol

1394's for the fun stuff...
Phil Roden, 1394 Digital Design Manager, Texas Instruments

Then he is to take the two goats and present them before the Lord at the entrance to the Tent of Meeting. He is to cast lots for the two goats — one lot for the Lord and the other for the scapegoat. Aaron shall bring the goat whose lot falls to the Lord and sacrifice it for a sin offering.
Leviticus 16:7-9

"Teach me, and I will be quiet; show me where I have been wrong. How painful are honest words! But what do your arguments prove? Do you mean to correct what I say, and treat the words of a despairing man as wind? You would even cast lots for the fatherless and barter away your friend. "
Job 6:24-27

Abstract This chapter presents a formal specification and analysis of the Root Contention Protocol from the physical layer of the IEEE 1394 ("FireWire", "iLink") standard. In our protocol models, randomization, real-time and timing constraints play an essential role. We analyze the protocol with three different methods: We first give a manual verification in the probabilistic I/O automata model of Segala and Lynch, where the emphasis is on the probabilistic and real–time behavior. Then, we provide a mechanical verification with the model checker Uppaal, where we are particularly interested in the timing constraints. And, finally, we present a fully parametric analysis using parametric model checking.

7.1 Introduction

The Root Contention Protocol (RCP) is an industrial leader election protocol, which uses randomization and timing delays as essential ingredients to determine a leader among two processes. Moreover, constraints on the timing parameters are crucial for correct protocol operation. All this makes RCP a perfect case study to investigate the applicability of the theory and analysis techniques presented in this thesis.

RCP is part of the IEEE 1394 standard. This standard defines a serial bus that allows several multimedia devices, such as TVs, PCs and VCRs, to be connected in a network and to communicate with each other at high speed. IEEE 1394 is currently one of the standard protocols for interconnecting multimedia equipment and, today, one can buy PCs, laptops and digital cameras with an IEEE 1394 serial bus port.

Various parts of IEEE have been specified and/or verified formally, see for instance [DGRV00, KHR97, Lut97]. Root contention has become a popular case study in Formal Methods. It has been analyzed in [Sha99, SV99a, BLdRT00, BST00, CS01, D'A99, KNS01, FS01, Sto99b, SV99b, SS01, HRSV01]. Section 7.3 of this chapter presents a comparative study of the verification methods applied in these works. After this section, we shift to the verification work carried out by ourselves and present three different approaches to the verification of RCP.

First, we provide a manual verification of RCP in the probabilistic automaton model of Segala & Lynch. This verification is based on [SV99b]. The emphasis is on probabilistic and real–time behavior, but we also derive constraints on the timing parameters which guarantee protocol correctness. To make the verification easier to understand, we introduce several intermediate automata in between the implementation and the specification automaton. After publication of [SV99b], we discovered that the model presented here does not fully comply to the IEEE 1394 standard. More precisely, the communication between the processes was not modeled completely appropriately. However, we present the verification as it is in [SV99b], because the very same techniques would apply to a correct protocol model and redoing the proofs is tedious and scientifically not interesting.

Moreover, the second analysis in this chapter improves the protocol model and carries out a mechanical verification. We follow the same approach as in the first verification and base the analysis on similar intermediate automata. However, we disregard the probabilistic aspects of the protocol in this verification, since they are the same as before. Instead, we concentrate on the functional and timing behavior, which is where both models differ. We use the timed model checker Uppaal to analyze the protocol's timing constraints. Since the standard version Uppaal is not able to do parameter analysis and the parametric version was not available when we carried out this research, we establish the results experimentally. By checking a large number of protocol instances, we derive two constraints that are necessary and sufficient for correct protocol operation. Although not completely formal, this is fast and easy and we believe that the results are trustworthy. Due to the differences in the communication models, the constraints derived here differ from the ones found in the first analysis.

Thirdly, we briefly report on a fully parametric verification of RCP. For this, we use the prototype parametric model checker described in Chapter 6. We formulate the protocol correctness in Uppaal's logic and analyze several automaton models from the previous verifications. Here, we profit from the fact that the parameter constraints have already been found. So, rather than having the tool generating the constraints, which is often a time and memory consuming task, we provide the constraints found before as initial constraints to the tool. The results here coincide with the ones from the previous two verifications.

Organization of the chapter

We start with an informal description of the protocol in Section 7.2. Then Section 7.3 presents an overview of several papers in the literature analyzing RCP. Sections 7.4, 7.5 and 7.6 present respectively a manual, a mechanical and a parametric verification of RCP. Each of these three sections comes with their own concluding remarks, whereas Section 7.7 presents some overall conclusions.

7.2 Root Contention within IEEE 1394

This section explains the IEEE 1394 root contention protocol informally. We start with some background information on the IEEE 1394 standard. In particular, we zoom in on the physical layer and the tree identify phase within this layer, which is where RCP is located. Finally, we describe RCP itself and its timing constraints.

7.2.1 The IEEE 1394 standard

The IEEE 1394 standard [IEE00a], which is also known under the popular names of FireWire and iLink, specifies a high performance serial bus. It has been designed for interconnecting computer and consumer equipment, such as VCRs, PCs and digital cameras. The bus supports fast and cheap, peer–to–peer data transfer among up to 64 devices, both asynchronous and isochronous. The bus is hot plug–and–play, which means that devices can be added or removed at any time.

Although originally developed by Apple (FireWire), the version documented in [IEE96] has been accepted as a standard by IEEE in 1996. More than seventy companies — including Sun, Microsoft, Lucent Technologies, Philips, IBM, and Adaptec — have joined in the development of the IEEE 1394 bus and related consumer electronics and software. Currently, IEEE 1394 is one of the standard protocols for connecting digital multimedia equipment. The IEEE 1394a supplement [IEE00a] to the standard is the latest approved standard that concerns root contention and includes several clarifications, extensions, and performance improvements over earlier standards. Ongoing standardization developments are taking place in several IEEE working groups, resulting in several standard proposals. For instance, P1394b ("gigabyte 1394") [IEE01b] develops a faster version of 1394. The proposal P1394.1 [IEE01a] concerns so–called bus bridges for interconnecting several IEEE 1394 networks and P1394.3 [IEE00b] proposes a peer–to–peer data transport protocol. This chapter concerns the IEEE 1394a standard, unless clearly stated otherwise.

The IEEE 1394 standard (and its successors) provides a layered, OSI–style description of the protocol. The presentation is rather informal. It uses text, diagrams and program code to describe the protocol operation.

The standard refers to the devices in a 1394 network as *nodes*, each having one or more *ports*. A port may be connected to one other node's port, via a bi–directional cable. Nodes should be connected in a tree–like network topology, that is, without any cycles. There are four protocol layers: the *serial bus management layer*, the *transaction layer*, the *link layer* and the *physical layer*. RCP is part of the so–called *tree identify phase*, present in the lowest, physical layer. This layer provides the electrical and mechanical interface for data transmission across the bus. Furthermore, it handles bus configuration, arbitration and data transmission.

The physical layer starts with bus configuration. This is done automatically upon a bus reset: after power up and after device addition or removal. Bus configuration proceeds in three phases. First, bus initialization is performed. This is followed by the tree identify phase. The purpose of this phase is to identify a leader (root) node and the topology of all attached nodes. The root will act as bus master in subsequent phases of the protocol. Finally, in the self identify phase, each node selects a unique physical ID and identifies itself to the other nodes. When bus configuration has been completed, nodes can arbitrate for access to the bus and transfer data to any other node.

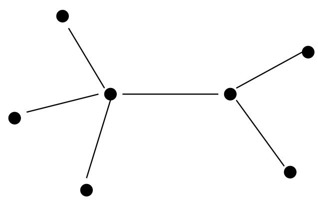


Figure 7.1: Initial network topology

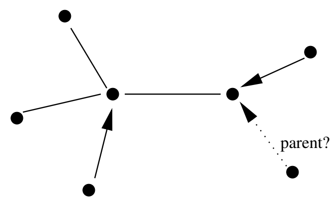


Figure 7.2: Intermediate configuration: leaf nodes send parent requests

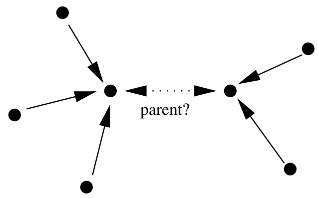


Figure 7.3: Two contending nodes

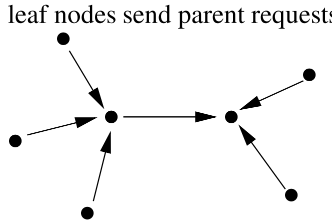


Figure 7.4: Final spanning tree: root contention has been resolved

The tree identify phase operates in the following way. First, it is checked whether the network topology is indeed a tree. If so, a spanning tree is constructed over the network and the root of this tree is elected as leader in the network.

The spanning tree is built as follows. As a basic operation, each node can drive a PARENT_NOTIFY (PN) or a CHILD_NOTIFY (CN) signal to a neighboring node. The node can also leave the line undriven (IDLE). The meaning of a PN signal is to ask the receiving node to become parent (connecting closer to the root) of the sending node. The sending node will then become its child and connects further away from the root. This signal is therefore also called a *parent request (req)*. A PN is acknowledged by a CN signal, also called *acknowledgement (ack)*. The receipt of a CN signal on a port is, in its turn, acknowledged by removing the PN signal from the connecting cable. A node only sends a PN signal via a port after it has received a PN signal on all its other ports. Thus, initially, only the leaf nodes, having only one port, send out a PN signal. If a node has received PN signals on all of its ports, then it has only child ports and it knows that is has been elected as the root of the tree. In the final stage of tree identify phase, two neighboring nodes may each try to find their parent by sending a PN signal to each other. This situation is called *root contention* and when it arises, the Root Contention Protocol is initiated to elect one of the two nodes as the root of the tree. Figures 7.1 – 7.4 (which have been borrowed from [DGRV00]) show a few snapshots of the tree identify phase. The arrows connect child ports to parent ports.

Lynch [Lyn96, p501] describes an abstract version of the tree identify protocol and suggests to elect the node with the larger unique identifier (UID) as the root in case of root contention. Since no UIDs are available during the tree identify phase (these will be assigned during a later phase of the physical layer protocol), the IEEE standard has chosen a probabilistic algorithm that is fully symmetric and does not require the presence of UIDs. This algorithm is described in the following section.

Timing constant		Min	Max
ROOT_CONTEND_FAST	1394	240 ns	260 ns
	1394a	760 ns	850 ns
	1394b	1600 ns	1610 ns
ROOT_CONTEND_SLOW	1394	570 ns	600 ns
	1394a	1590 ns	1670 ns
	1394b	3220 ns	3332 ns

Figure 7.5: Root contend wait times from IEEE 1394, 1394a and 1394b

7.2.2 The Root Contention Protocol

If a node receives a PN signal on a port, while sending a PN signal on that port, it knows it is in root contention, see Figure 7.3. Note that root contention is detected by each of the two contending nodes individually. Upon detection of root contention, a node backs off by removing the PN signal from the connecting cable, leaving the line in the state IDLE. At the same time, it starts a timer and picks a random bit. If the random bit is one, then the node waits for a time ROOT_CONTEND_SLOW (abbreviated $\mathrm{RC\_SLOW}$). If the random bit is zero, it waits for a shorter time ROOT_CONTEND_FAST (abbreviated $\mathrm{RC\_FAST}$). Table 7.5 lists the wait times as specified in the IEEE 1394, 1394a standards and P1394b proposal [IEE00a]. It is quite surprising that these values differ in the various standards and standard proposals, since these values influence the maximal values for several other protocol constants, see the discussion in Section 7.2.3.

When its timer expires, a node samples its contention port once again. If it sees IDLE, then it sends a PN anew and waits for a CN signal as an acknowledgment. If, on the other hand, a node samples a PN on its port, it replies with a CN signal as an acknowledgement and becomes the root.

If both nodes pick different random bits, then the slowest (picking the bit one) is elected as leader. In the case that both nodes pick identical random bits, there are two possibilities. The root contention times allow one process to wait significantly longer than the other, even if both processes pick the same random bits. If this occurs, then the slower node becomes the root. Secondly, if the nodes proceed with about the same speed, then root contention reoccurs: when its timer expires, each node sees an IDLE signal and starts sending a PN signal, which will cause renewed root contention. In that case, the whole process is repeated until one of the nodes becomes root. Eventually (with probability one), both nodes will pick different random bits, in which case root contention certainly is resolved.

7.2.3 Protocol Timing Constraints and their Implications

The timing constraints

RCP contains five timing constants, which can all be treated as parameters. The minimum and maximum values of the RCP wait times, listed in Table 7.5, yield the parameters rc_fast_min, rc_fast_max, rc_slow_min and rc_slow_max. The communication delay $delay$ in the wires is the fifth parameter. It models the maximal total time from sending a signal by one node to receiving it by the other node. That is, it includes the cable propagation delay and the time to process the cable line states by the hardware and software layers at the ports of the two nodes.

We assume that all parameters take values in $\mathbb{R}^{\geq 0}$ and, moreover, we assume three basic parameter constraints

$$rc\_slow\_min \leq rc\_slow\_max, \qquad (B_1)$$
$$rc\_fast\_min \leq rc\_slow\_min \text{ and} \qquad (B_2)$$
$$rc\_fast\_min \leq rc\_fast\_max. \qquad (B_3)$$

For the protocol to work correctly, two additional constraints are essential.

$$2 * delay < rc\_fast\_min. \qquad (Eq_1)$$
$$2 * delay < rc\_slow\_min - rc\_fast\_max. \qquad (Eq_2)$$

Note that we do not assume $rc\_fast\_max \leq rc\_slow\_max$ beforehand, but it follows from Eq_2. The timing constraints do not appear in the IEEE 1394 specifications [IEE96, IEE00a], but — fortunately — the wait times from the various standards (Table 7.5) do all meet these constraints.

The origin of these equations is visualized in Figure 7.6 and explained below. The analysis is based on informal notes [LaF97, Nyu97] to the IEEE P1394a Working Group. The equations from [LaF97] match ours, whereas [Nyu97] incorrectly cites [LaF97] and contains some errors. Due to an imperfection in the communication model, the scenario described below to illustrate the need for Eq_2 cannot occur in the model presented in Section 7.4. Hence, that section finds weaker constraint equations.

In the explanation below, we refer to the contending notes as Node 1 and Node 2. Recall that root contention is detected by both nodes individually.

Ad Equation Eq_1: In case of Node 2 selecting the short waiting period, constraint Equation Eq_1 ensures that the IDLE signal from Node 1 arrives at Node 2 *before* the waiting period of Node 2 ends (See circle 1 in Figure 7.6). Otherwise, the following erroneous scenario might happen: Node 2 might still see the first PN signal from Node 1, and erroneously send a CN signal to acknowledge this parent request. Once the IDLE signal from Node 1 arrives (behind schedule), Node 2 removes its CN signal again and declares itself root. When Node 1 ends its waiting period, however, it will see the IDLE signal from Node 2, as if nothing happened, and send a PN to Node 2. Awaiting the response it will time out, which leads to a bus reset. Therefore, constraint Equation Eq_1 is required for correct protocol operation.

Ad Equation Eq_2: This constraint ensures that root contention is always resolved in case of one node (say Node 1) selecting the short waiting period and the other (Node 2) selecting the long waiting period. More precisely, Eq_2 ensures that the new PN signal from Node 1 arrives at Node 2 *before* the waiting period of Node 2 ends (see circle 2 in Figure 7.6). Otherwise, Node 2 might still see the IDLE signal from Node 1 and sends another PN signal. Together with the PN message coming from Node 1 (after schedule), this will again lead to root contention, although the two nodes picked different random bits. Thus, this equation ensures that renewed root contention can only occur for if both nodes pick equal random bits.

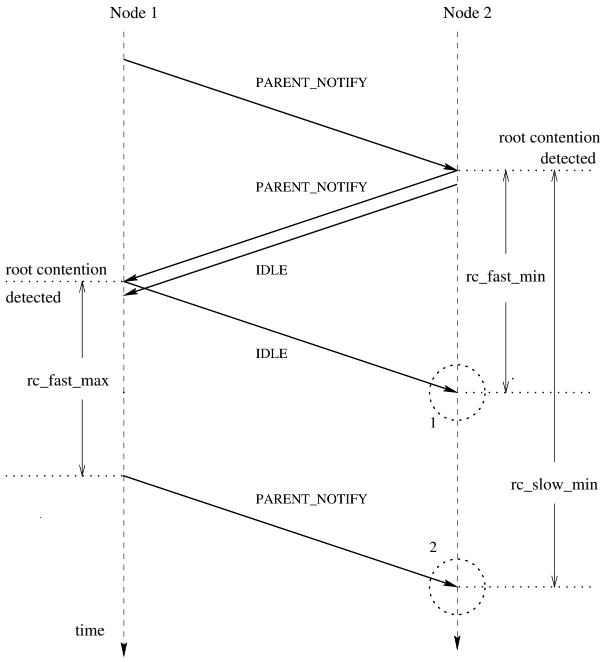


Figure 7.6: Visualization of the protocol timing constraints

Consequences of the timing constraints

The timing constraints have several consequences. Equations Eq_1 and Eq_2 can be rewritten as

$$delay < \frac{\min\left(rc\_fast\_min, rc\_slow\_min - rc\_fast\_max\right)}{2}.$$

Filling in the actual values of the wait times appearing in the various standards then yield an upper bound for the value of $delay$. We have $delay < 120$ ns for the IEEE 1394 standard, $delay < 370$ ns for IEEE 1394a and $delay < 800$ ns for P1394b. This influences the maximum cable and processing delays. Recall that the parameter $delay$ includes the propagation and the processing delay.

The cable propagation delay is specified to be less or equal than $5.05\ \mathrm{ns/m}$. Unfortunately, any additional processing delays are not explicitly specified in the standard. If, on the one hand, we disregard such extra delays, then the maximum cable length between two nodes is given by: $\frac{120\ \mathrm{ns}}{5.05\ \mathrm{ns/m}} \approx 24\ \mathrm{m}$ for 1394, approximately 73 m for 1394a and approximately 158 m is allowed for P1394b. However, for IEEE 1394 and 1394a, the cable length is already limited to $4.5\ \mathrm{m}$ by the worst case round trip propagation delay during bus arbitration.

If, on the other hand, processing delays cannot be ignored, then the current maximum cable length leaves room for a processing delay on each side of the wire of maximally $\frac{120-4.5\cdot 5.05}{2}$ ns $\approx$ 49 ns, $\frac{370-4.5\cdot 5.05}{2}$ ns $\approx$ 173 ns and $\frac{800-4.5\cdot 5.05}{2}$ ns $\approx$ 389 ns for IEEE 1394, 1394a and 1394b respectively.

If new applications require longer cable lengths — for instance, [IEE01b] expects $100\ \mathrm{m}$

cables to be needed for home networks — then the timing constraints above require either the root contention times to be increased, or the communication delay to be decreased. The latter is proposed in PIEEE 1394b. This proposal allows, besides copper wires, also glass optical fiber and plastic optical fiber to connect the devices. If glass optical fiber is used, then this standard proposal supports 100 meter cables, for plastic optical fibers this is 50 meter, while copper wires remain limited to 4.5 meter. Since glass optical fiber transmits signals nearly with the speed of light, the pure signal propagation delay cannot be a limiting factor anymore. Furthermore, the IEEE 1394 working groups also investigate alternative root contention protocols [LaF97] which allow for longer cable lengths.

7.3 Experiences with verifying the IEEE 1394 Root Contention protocol

This section compares several approaches to the verification of IEEE 1394 RCP and reports on the experiences and lessons to be learned when applying formal methods to industrial applications. Although RCP is small and easy to understand, the problems encountered in the verification of this protocol are in many aspects illustrative for the application of formal methods to other real–life applications.

7.3.1 Aspects of RCP

Several different features play a role in the modeling and analysis of RCP. Due to the use of random bits, the protocol is *probabilistic* in nature. *Real–time* is needed to model and analyze the root contend wait times and communication delays in the cables. Furthermore, *nondeterminism* is essential to model the fact that the root contend wait times and the communication delay are not fixed values, but lie within intervals. Within more abstract protocol descriptions, nondeterminism models the phenomenon that, if two nodes pick the same random bits, then either one of them is elected as leader or root contention reoccurs. Finally, *parametric models* treat the values of the timing delays as parameters rather than as fixed constants.

Moreover, several properties are of interest for the protocol's correctness. *Safety properties* are properties stating that "nothing bad ever happens" during execution of the protocol. A crucial safety property for RCP is that at most one leader is elected. *Liveness properties* state that "eventually, something good happens." An important liveness property is that at least one leader is elected (with probability one). Furthermore, *performance issues* concern the quantitative behavior of the protocol, for instance the probability that a leader is elected within a certain amount of time or the average number of rounds needed to elect a leader. Each of the properties mentioned can be tackled either parametrically or nonparametrically.

Obviously, it is impossible to consider all those features and properties at the same time. Therefore, each of the works described below focuses on one or more aspects, while abstracting from others.

Organization of the section This section is organized as follows. We first consider papers that study the functional and timing behavior of RCP in Section 7.3.2. We discuss several approaches to parametric verification in Section 7.3.3. Then, Section 7.3.4 presents several studies of the performance analysis of the protocol. Finally, Section 7.3.5 presents some conclusions. Within each subsection, the works are discussed in chronological order.

7.3.2 Functional and Timing behavior of RCP

Verification of RCP using stepwise refinement

The papers [Sto99b, SV99b, SS01] (where [SV99b] and [SS01] correspond to Sections 7.4 and 7.5 of this thesis.) all follow an automaton–based approach to verify RCP. The protocol and its specification are both described as automata, respectively $\mathtt{Impl}$ and $\mathtt{Spec}$, and correctness is expressed by $\mathtt{Impl} \sqsubseteq \mathtt{Spec}$. Here, $\sqsubseteq$ is a suitable notion of trace inclusion. The protocol correctness is established by stepwise abstraction: it is shown that $\mathtt{Impl} \sqsubseteq \mathtt{I}_1 \sqsubseteq \mathtt{I}_2 \sqsubseteq \mathtt{I}_3 \sqsubseteq \mathtt{Spec}$. Here the mentioned works consider different versions of $\mathtt{Impl}$ and $\mathtt{I}_1$, where $\mathtt{I}_1$ is obtained by abstracting from the communication in $\mathtt{Impl}$. Automaton $\mathtt{I}_2$ removes all timing information from $\mathtt{I}_1$ (in the discrete time case $\mathtt{I}_1 = \mathtt{I}_2$) and in $\mathtt{I}_3$ internal choices are further contracted. The main probabilistic analysis is carried out in the step $\mathtt{I}_2 \sqsubseteq \mathtt{I}_3$, which concerns very small automata.

Discrete time model ([Sto99b]) As a starting point for further verification, [Sto99b] describes a probabilistic, discrete time model of the protocol. It uses the probabilistic I/O automata framework developed by Segala [Seg95b]. The abstraction to discrete time is justified by the observation that RC_SLOW is about 2 times RC_FAST and that the communication delay is negligible compared to the root contention wait times.

In this model, the probabilistic behavior (in combination with fairness) has been studied. Most of the verification has been done manually, but several invariants and fairness properties have been checked with the model checker SMV [McM92]. It turned out that it is not so difficult to model the protocol in SMV and check the desired properties. However, the formal relationship between the I/O automaton model and the derived SMV model, needed to infer the correctness of the I/O automaton model model from the correctness of the SMV model, involves many technical details.

Real–time model ([SV99b] and Section 7.4 of this thesis.) In order to study the real–time behavior, timing has been modeled more precisely in [SV99b], using the probabilistic timed I/O automaton [Seg95b]. As in the discrete time model, the communication between the nodes is modeled as the transfer of packages (PN or CN). That is, single messages which are sent only once and, upon receipt, removed from the wire. The analysis of this model has been done manually, where the constants rc_fast_min, rc_fast_max, rc_slow_min, rc_slow_max and $delay$ are treated as timing parameters. Two constraints on these parameters are derived that ensure protocol correctness:

$$delay < rc\_fast\_min, \qquad (Eq_0)$$
$$2 * delay < rc\_slow\_min - rc\_fast\_max. \qquad (Eq_2)$$

These constraints differ from the constraints Eq_1 and Eq_2 in Section 7.2.3, which shows that the model in [SV99b] does not conform to the IEEE standard.

Detailed model ([SS01] and Section 7.5 of this thesis.) A close inspection of the IEEE documentation yielded that it is inappropriate to model the communication between the nodes by a packet mechanism as in [Sto99b, SV99b]. This is for two reasons. First, it is necessary to model the absence of a message (IDLE) explicitly. Secondly, signals, unlike packages, may remain unseen by the receiving node. This is the case if a second signal (possibly IDLE) arrives at the receiving node's port while the node has not sampled its port since the first signal has arrived.

This analysis yielded a more detailed model $\mathtt{Impl}^{\mathtt{B}}$, where the communication has been modeled by signals. More precisely, the model uses events to represent changes in the signals — the signals themselves are driven continuously across the wire. There is one (minor) point where the model in [SS01] does not conform to the IEEE standard: initially, both nodes in this model detect root contention simultaneously, whereas one can infer from the standard that any delay less than $delay$ is allowed between the detection of root contention by both nodes. When root contention reoccurs, the model in [SS01] does, however, allow for exactly this maximal delay between both detections.

Since the probabilistic analysis of this model is very similar to the real–time model, [SS01] only considers the timing aspects. By checking several safety and liveness properties for many different parameters values with the model checker Uppaal, [SS01] verified experimentally that the equations Eq_1 and Eq_2 from [LaF97] are necessary and sufficient for correct protocol operation. As is the case with SMV, it is not difficult to model the protocol in Uppaal, but the formal relationship between the I/O automaton model and the Uppaal model involves many technical details.

Modeling RCP with E–LOTOS

E–LOTOS Shankland et al. [Sha99, SV99a] present a formal description of RCP in E–LOTOS – an extension of LOTOS with time – of the entire tree identify phase in 1394, including RCP. An advantage of E–LOTOS is its similarity with programming languages, making it easy to read for engineers, see [MS00]. Since tools for this language have not been developed yet, no rigorous verification is carried out for the E–LOTOS models.

Although created independently, the models [Sha99, SV99a] (the RCP part) and [SV99b] are quite similar and both do not completely comply to the standard. Each of these works models the communication by a packet mechanism. Secondly, in [Sha99, SV99a], a CN is sent immediately after a PN has been detected, whereas the standard requires to wait at least the minimal root contention time. It is said in [SV99a, MS00] that this has been done because checking for a message after the waiting time has been expired is not expressible in E–LOTOS. If this is indeed the case, then this would plead for an extension of E-LOTOS with more expressive means.

Being integrated in the tree identify phase, the nodes in [Sha99, SV99a] automatically detect root contention asynchronously, in less than $delay$ time one after another, also in the initial state, which was not the case in [SS01].

7.3.3 Parametric Verification of RCP

Currently, several model checkers exist that can verify parametric timed systems, namely HyTech [HHW97], LPMC [gDUoT], TReX [ABS01] and a parametric extension to Uppaal [HRSV01]. All have been applied to RCP.

Given a system model $\mathcal{A}$, a property ϕ and (optionally) an initial parameter constraint C_0, the aim of parametric verification is to synthesize a parameter constraint C_1 which describes the exact conditions on the parameter values needed and sufficient for $\mathcal{A}$ to satisfy ϕ, where we may assume that the parameters meet C_0. Formally, we require that $C_0 \implies (\mathcal{A} \models \phi \iff C_1)$. Although this problem is undecidable ([AHV93]) for the input models used by the tools (timed automata), the tools mentioned about did manage to do parameter analysis for several practical examples, including RCP.

LPMC Toetenel and his team ([BLdRT00, BST00]) have used their parametric model checker LPMC to investigate the timing constraints of RCP. Here the values of $delay$ and (in some cases) $rc\_slow\_min - rc\_fast\_max$ are taken as parameters. The other values are taken as constants. Since the time and memory consumption in the verification was very modest, one might wonder why [BLdRT00, BST00] did not analyze a fully parametric model.

The entire verification is done with LPMC, which is unlike [SS01, HRSV01, CS01], where additional machinery is needed to deal with liveness properties and with probabilistic choice. The probabilistic choice has been replaced with a fairness property. Since only functional behavior is considered, this is appropriate, as the fairness property is implied by the probabilistic behavior of the protocol. The model in [BLdRT00] is similar to the model in [SV99b]; [BST00] is similar to [SS01]. The same timing constraints are found for the corresponding models.

However, by designating a different initial state, the model in [BST00] allows the nodes to detect root contention asynchronously also when the protocol starts. Thus, this model in [BST00] does completely conform to the standard.

Parametric Uppaal ([HRSV01] and Section 7.5 of this thesis.) The work [HRSV01] verified the models in [SV99b] and [SS01] with a parametric extension of the model checker Uppaal (see also Chapter 6), where all the five constants of RCP are treated as parameters. The sufficiency of the constraints in [SV99b] and [SS01] was checked by providing these as initial constraints to the tool and checking that no extra constraints are needed for the property under investigation to hold. For several cases, necessity is established similarly, that is, by checking that the property fails for all parameter values satisfying the negated constraints. This approach is different from [BLdRT00, BST00, CS01], which all synthesize the constraints.

The special format of the automata modeling RCP allowed certain parameters to be eliminated and still to obtain general results, which gained serious speed ups.

TReX Another parametric verification of RCP has been carried out by Collomb–Annichini & Sighireanu [CS01]. Using their TReX tool [ABS01] as well as HyTech, [CS01] analyzed liveness and safety properties of variations of `Impl`, including a model that allows for asynchronous detection of root contention in the initial state. All 5 timing constants are taken as parameters and the same constraints as in [SS01, HRSV01] are established.

TReX synthesizes the constraints for RCP automatically. Since TReX overapproximates the constraints for which the property does not hold, several runs of the tool with different initial constraints are needed to derive the constraints which are sufficient for the property under investigation to hold. In this way, [CS01] derives the constraints for several properties.

Moreover, [CS01] establishes that $\mathtt{Impl} \sqsubseteq \mathtt{I}_1$ if the parameters meet the constraints Eq_1 and Eq_2. Here, Eq_1 and Eq_2 are given as initial constraints, not synthesized. The notion of observation in TReX allows to smoothly check whether $\mathtt{Impl} \sqsubseteq \mathtt{I}_1$, whereas [HRSV01] needs rather complicated constructions on timed automata.

The performance of TReX is worse, both in time and in space, than the parametric extension to Uppaal, also in the cases where constraints were checked and not synthesized. This is explained by the fact that TReX is more general than Parametric Uppaal. In particular, it is able to deal with nonlinear constraints, which require more complex comparison algorithms. Compared with HyTech, TReX is slower, but HyTech was not able to synthesize the constraints for RCP and run out of memory for several crucial properties.

Comparison All tools are able to analyze the constraints for RCP and report the same results. Each of the approaches has its own weak and strong points – basically determined by the power of underlying tools. LPMC is the only tool that can handle fairness, but analyzes only 2 parameters; parametric Uppaal is fast, but did not synthesize the constraints; TReX can synthesize the constraints, but since it overapproximates the constraints, multiple runs of the tool are needed. Thus, it seems reasonable to expect that a combination of the techniques implemented in the various tools will yield further improvements.

7.3.4 Performance Analysis of RCP

Performance analysis aims at a quantitative analysis of a system. Interesting performance measures for RCP are for instance the minimal probability to elect a leader within a certain time or the maximal average number of rounds needed before a leader is elected. Traditionally, performance analysis techniques have been developed for purely probabilistic systems and cannot be applied directly to systems combining probability with nondeterminism, such as RCP. Moreover, most performance measures are no longer expressible as a single number, but, depending on how the nondeterminism is resolved, yield a number within an interval. Several performance analysis methods have been extended for systems with nondeterminism in [LSS94, BA95, Alf97, McI99, KNSS01]. Since the timing delays of RCP lie in intervals, rather than having fixed values, only [LSS94, KNSS01] are directly applicable here.

Thus, the approach taken by [D'A99] and [FS01] is to remove the nondeterminism and replace it respectively by a probabilistic and a deterministic choice. The work [KNS01] is one of the few case studies taking the challenge of doing performance analysis in the presence of nondeterminism.

Spades D'Argenio [D'A99] investigates the performance of RCP using a discrete event simulator for $\spadesuit$ (Spades). $\spadesuit$ is a stochastic process algebra which allows to specify delays by arbitrary probability distributions. Discrete event simulation is similar to testing in the sense that many runs from a system are taken, for which the performance measures are then computed. However, since all choices are probabilistic, one can exactly quantify the accuracy of the simulation — which is high if very many runs are taken.

The protocol model is based on [SV99b]. Although the standard specifies timing delays to be taken nondeterministically within their respective intervals, [D'A99] assumes a uniform distribution for the root contention times and β–distribution for the communication delay. Since techniques and tools for doing performance analysis in the presence of nondeterminism and real–time hardly exist, resolving the nondeterministic choices by probabilistic ones is currently the best one can do. The analysis shows that, in most of the cases, root contention is resolved in one round of the protocol. It also revealed that both the average time until root contention is resolved and its variance grow approximately linearly with the cable length.

Deadline properties Kwiatkowska, Norman and Sproston [KNS01] focus on a quantitative analysis of RCP and study *deadline properties* of the protocol. Given a deadline of d ns, they compute the minimal probability that RCP elects a leader within that deadline, for different values of d. Such deadline properties can be verified automatically by the tool Prism [dAKN$^+$00, KNP02] for systems modeled as finite Markov decision processes (MDPs).

The models analyzed are $\mathtt{Impl}$ and $\mathtt{I}_1$ from [SS01], augmented with probabilities $\frac{1}{2}$ for the outcomes of the coin flips. Since $\mathtt{Impl}$ and $\mathtt{I}_1$ include real–time, these do not fall into the class of MDPs. Three techniques are used to translate $\mathtt{I}_1$ into a finite MDP: the first uses the forward reachability method from [KNSS01] implemented in Hytech; the second partitions the state space of $\mathtt{I}_1$ according to the region equivalence as in [AD94]; and the last one interprets $\mathtt{I}_1$ in integer semantics, which yields in this case a model that is equivalent to the standard real semantics, but which is finite. The resulting MDPs were then given as input to Prism and all yield (upon termination) the same minimal probability for electing a leader within the choose deadlines. Since $\mathtt{Impl} \sqsubseteq \mathtt{I}_1$, we know that the minimal probability to elect a leader within deadline d for $\mathtt{I}_1$ is a *lower bound* for the minimal probability for $\mathtt{Impl}$ to do so. This needs not be the exact probability, since it is not known whether $\mathtt{I}_1 \sqsubseteq \mathtt{Impl}$ – probably so, but a firm result would be useful here.

The model $\mathtt{Impl}$ was analyzed using integer semantics only, which was the most efficient technique for the analysis of $\mathtt{I}_1$. Since the state space of the generated MDP grows with the deadline d, smaller deadlines can be analyzed for $\mathtt{Impl}$ than for $\mathtt{I}_1$. The minimal probabilities for $\mathtt{Impl}$ are the same as for $\mathtt{I}_1$, thus suggesting that $\mathtt{I}_1 \sqsubseteq \mathtt{Impl}$ indeed.

pGCL A third analysis of the performance of RCP has been carried out by Fidge & Shankland [FS01] using pGCL (probabilistic guarded command language). The language pGCL [MM99] is a probabilistic extension of Dijkstra's GCL where pre– and postcondition predicates no longer yield booleans, but values in $[0, 1]$ representing probabilities.

The model analyzed in [FS01] is a high–level description of the protocol in which no nondeterminism is present. Instead, root contention always reoccurs when two nodes pick the same random bits. When computing the worst case performance, this abstraction is appropriate. However, a formal proof to justify this would be valuable.

Using pGLC proof rules, [FS01] establishes that the probability to terminate in M rounds of the protocol equals $1 - (p^2 + (1-p)^2)^M = 1 - (1 - 2p + 2 \cdot p^2)^M$, where p is the probability to select fast timing. Since a round is bounded by $\mathit{rc\_slow\_max}$ (communication delays are neglected), the probability to terminate within a deadline $M \cdot \mathit{rc\_fast\_max}$ is at least $1 - (1 - 2p + 2 \cdot p^2)^M$. These lower bounds are strictly smaller than the exact ones derived by [KNS01]. This can be explained by the fact that, if both nodes select fast timing, then a round is bounded by the smaller figure $\mathit{rc\_fast\_min}$. However, [FS01] present an easy to compute symbolic expression, whereas the tool [KNS01] computes exact bounds for a fixed number of deadlines.

Besides RCP, this work also provides a pGCL analysis of the tree identify phase, of which RCP is a part.

Future work Several interesting performance aspects of RCP remain to be investigated, such as the minimum and maximum average number of rounds and the minimal and maximum average time before a leader is elected. The works [D'A99, KNS01, FS01] give a good starting point here.

A useful fact for efficiency reasons, which has been successfully exploited in [KNS01], is that the implementation relation $\sqsubseteq$ mentioned above preserves many relevant performance measures (namely those that can be expressed by traces). In other words, if $\mathcal{A} \sqsubseteq \mathcal{B}$ then $\mathcal{A}$

does not perform worse than $\mathcal{B}$ for those measures. If we also have $\mathcal{B} \sqsubseteq \mathcal{A}$, then $\mathcal{A}$ and $\mathcal{B}$ satisfy *exactly* the same performance measures.

Moreover, we remark that the minimal and maximal average *number of rounds* can be computed easily with the methods by [BA95, Alf97] on the automaton $\mathtt{I}_3$, because we can abstract from the exact timing delays. However, no techniques exist yet for calculation of the average *time* before a leader is elected; an extension of the results by [BA95, Alf97] to probabilistic timed automata would be useful here.

A funny property of RCP is that the performance can (theoretically) be improved by using a biased coin. More precisely, both the minimal and the maximal time before a leader is elected become smaller, if the probability to choose fast timing (random bit zero) slightly increases. This is the case because, although the minimal and maximal number of rounds are smallest if an unbiased coin is used, the time per round is lowest when both processes select fast timing. Therefore, favoring fast timing a little bit, yields faster termination of the protocol.

7.3.5 Concluding Remarks

From the papers [Sha99, SV99a, BLdRT00, BST00, CS01, D'A99, KNS01, FS01] and our own experiences with the formal verification of the IEEE 1394 Root Contention protocol, we conclude the following.

In order for the results of a formal verification to be reliable, the protocol models must – of course – be realistic. Constructing a realistic protocol model is, however, not easy. It is unavoidable to abstract from certain details in the standard, but it is hard to judge whether these abstractions are appropriate. This is hampered by the fact that industrial standards are often informal, incomplete and difficult to read for nonexperts.

Since it turned out to be inappropriate to model the communication delay between the nodes by a packet mechanism in RCP, it is worthwhile considering to what extent this is appropriate in the other parts of the Tree Identify Phase.

For a maximal profit from tool support, it is desirable to have more established translations between different formalisms and input languages of tools. Encoding a model specified in a certain formalism into the input language of a tool often involves a lot of technical details due to differences in synchronization principles, fairness assumptions, priorities, etc. Since such encodings can often be automated quite easily, a lot of time verification effort could be saved.

7.4 First Approach: A Manual Verification of the Root Contention Protocol

The rest of this chapter is devoted to the verification of RCP, using three different techniques. We start with a manual verification.

The verification in this section is carried out in the PA model [Seg95b, SL95], described in Chapter 2. More precisely, we use two extensions of this model, viz. probabilistic I/O automata and timed probabilistic I/O automata. These are also due to Segala [Seg95b] and present below. Following a well–tried method, we prove the correctness of the protocol by establishing a probabilistic simulation between the implementation and the specification, both modeled as (timed) probabilistic I/O automata.

The probabilistic simulation relations from [Seg95b, SL95] are rather complex. To simplify the simulation proofs, we use the simpler kind of simulation relations, namely probabilistic step refinements and probabilistic hyperstep refinements, which were introduced in Section 2.5.

The strategy followed in the simulation proof is the following. Given the protocol automaton $\mathtt{Impl}^{\mathtt{A}}$ and the abstract specification $\mathtt{Spec}$, we define three intermediate automata $\mathtt{I}_1^{\mathtt{A}}$, $\mathtt{I}_2^{\mathtt{A}}$, and $\mathtt{I}_3^{\mathtt{A}}$. First, $\mathtt{I}_1^{\mathtt{A}}$ abstracts from the message passing in $\mathtt{Impl}^{\mathtt{A}}$, but keeps the same probabilistic choices and most of the timing information. Next, $\mathtt{I}_2^{\mathtt{A}}$ abstracts from all the timing information in $\mathtt{I}_1^{\mathtt{A}}$, and $\mathtt{I}_3^{\mathtt{A}}$ abstracts from the probabilistic choice in $\mathtt{I}_2^{\mathtt{A}}$. The introduction of the intermediate automata allows us to separate our concerns. The simulation between $\mathtt{Impl}^{\mathtt{A}}$ and $\mathtt{I}_1^{\mathtt{A}}$ is easy from a probabilistic point of view and its proof mainly involves traditional, non–probabilistic techniques like proving invariants. The remaining simulations between automata $\mathtt{I}_2^{\mathtt{A}}$, $\mathtt{I}_3^{\mathtt{A}}$ and $\mathtt{Spec}$ deal with probabilistic choice, but since these automata are small this is not so difficult anymore.

Randomized algorithms and protocols have been the topic of various case studies. For instance, [LSS94] analyzes time bounds in the randomized dining philosopher's problem, in [Agg94] a number of algorithms for constructing a spanning tree in a network are designed and verified, [PSL97] verifies the randomized consensus protocol by Aspnes & Herlihy. All these case studies are carried out in the probabilistic I/O automata framework, but only the verification in the latter work is (partially) based on simulation relations; the others use different techniques.

This rest of this section is organized as follows. Section 7.4.1 introduces the probabilistic I/O automaton and the timed probabilistic I/O automaton model. Section 7.4.2 presents the automaton models describing the protocol implementation and specification. Then, Section 7.4.3 defines the intermediate automata and establishes the protocol correctness via the simulation relations. Finally, Section 7.4.4 presents some concluding remarks.

7.4.1 Probabilistic I/O Automata

The protocol analysis in this section is carried out in the probabilistic I/O automaton framework. This is a variation of the PA framework, where each PA is equipped with a distinction between input and output actions, and with a notion of fair behavior. The probabilistic I/O automaton model enables, even more than the PA model, unambiguous, concise and understandable system models, see [Lyn96] for supporting arguments. This section introduces the probabilistic I/O automaton model together with several results and operations we need in our verification.

Definition 7.4.1 A *probabilistic I/O automaton* $\mathcal{A}$ is a probabilistic automaton enriched with

1. a partition of $V_{\mathcal{A}}$ into *input actions* $in_{\mathcal{A}}$ and *output actions* $out_{\mathcal{A}}$, and
2. a *task partition* $tasks_{\mathcal{A}}$, which is an equivalence relation over $out_{\mathcal{A}} \cup int_{\mathcal{A}}$ with countably many equivalence classes.

We require that $\mathcal{A}$ is *input enabled*, which means that for all $s \in S_{\mathcal{A}}$ and all $a \in in_{\mathcal{A}}$, there is a μ such that $s \xrightarrow{a}_{\mathcal{A}} \mu$.

As probabilistic I/O automata are enriched PAs, we can use the notions of nonprobabilistic variant, reachable state, execution and trace also for probabilistic I/O automata. For their definitions, see Section 2.3.1 on page 40. The set of executions (execution fragments, traces) and finite executions (execution fragments, traces) of $\mathcal{A}$ are respectively denoted by $execs(\mathcal{A})$ $(frags(\mathcal{A}), traces(\mathcal{A}))$ and by $execs^*(\mathcal{A})$ $(frags^*(\mathcal{A}), traces^*(\mathcal{A}))$.

In the protocol analysis, there is one more operation we need, namely, hiding of actions. This operation turns external actions into internal ones, so that no synchronization takes place on those actions. We define this operation on the level of PAs, rather than on probabilistic I/O automatons.

Definition 7.4.2 If $\mathcal{A}$ is a PA and $X \subseteq ext_{\mathcal{A}}$, then $hide(\mathcal{A}, X)$ is the probabilistic automaton $(S_{\mathcal{A}}, S^0_{\mathcal{A}}, (ext_{\mathcal{A}} \setminus X, int_{\mathcal{A}} \cup X), \Delta_{\mathcal{A}})$.

Definition 7.4.3 Let $\mathcal{A}$ be a probabilistic I/O automaton. An execution α of $\mathcal{A}$ is *fair* if the following conditions hold for each class C of $tasks_{\mathcal{A}}$:

1. If α is finite, then C is not enabled in the final states of α.

2. If α is infinite, then α contains either infinitely many actions from C or infinitely many occurrences of states in which no action in C is enabled.

Similarly, a trace of $\mathcal{A}$ is *fair* in $\mathcal{A}$ if it is the trace of a fair execution of $\mathcal{A}$. The sets of fair executions and fair traces of $\mathcal{A}$ are denoted by $fexecs(\mathcal{A})$ and $ftraces(\mathcal{A})$ respectively.

Definition 7.4.4 Let $\mathcal{A}$ and $\mathcal{B}$ be probabilistic automata with $ext_{\mathcal{A}} = ext_{\mathcal{B}}$. Let r be a mapping from $S_{\mathcal{A}}$ to $S_{\mathcal{B}}$. Then r induces a relation $\tilde{r} \subseteq frags(\mathcal{A}) \times frags(\mathcal{B})$ as follows: if $\alpha = s_0 a_1 s_1 \cdots \in frags(\mathcal{A})$, $\mathcal{I}$ is the index set of α, $\beta = t_0 b_1 t_1 \cdots \in frags(\mathcal{B})$ and $\mathcal{J}$ is the index set of β, then $\alpha \tilde{r} \beta$ if and only if there is a surjective, nondecreasing index mapping $m : \mathcal{I} \to \mathcal{J}$, such that for all $i \in \mathcal{I}, j \in \mathcal{J}$,

1. $m(0) = 0$

2. $r(s_i) = t_{m(i)}$

3. if $i > 0$ then either of the following conditions holds

   (a) $a_i = b_{m(i)} \wedge m(i) = m(i-1) + 1$ or

   (b) $a_i \in I_{\mathcal{A}} \wedge b_{m(i)} \in I_{\mathcal{B}} \wedge m(i) = m(i-1) + 1$ or

   (c) $a_i \in I_{\mathcal{A}} \wedge m(i) = m(i-1)$.

In [PSL97], fair trace distribution inclusion, notation $\sqsubseteq_{\mathrm{FTD}}$, is proposed as an implementation relation between probabilistic I/O automata that preserves both safety and liveness properties.

Claim 7.4.5 ([SV02a]) *Let $\mathcal{A}$ and $\mathcal{B}$ be probabilistic I/O automata. Let r be a probabilistic step refinement from $\mathcal{A}$ to $\mathcal{B}$ that relates each fair execution of $\mathcal{A}$ only to fair executions of $\mathcal{B}$. Then $\mathcal{A} \sqsubseteq_{\mathrm{FTD}} \mathcal{B}$.*

Timed probabilistic I/O automata

Timed probabilistic I/O automata extends probabilistic I/O automata with time passage actions, c.f. Definition 2.2.10.

Definition 7.4.6 A *timed probabilistic I/O automaton* $\mathcal{A}$ is a probabilistic automaton enriched with a partition of $ext_{\mathcal{A}}$ into *input actions* $in_{\mathcal{A}}$, *output actions* $out_{\mathcal{A}}$, and the set $\mathbb{R}^{>0}$ of positive real numbers or *time-passage actions*. We require[1] that, for all $s, s', s'' \in S_{\mathcal{A}}$ and $d, d' \in \mathbb{R}^{>0}$ with $d' < d$,

1. $\mathcal{A}$ is input enabled,
2. each step labeled with a time-passage action leads to a Dirac distribution,
3. (Time determinism) if $s \xrightarrow{d}_{\mathcal{A}} s'$ and $s \xrightarrow{d}_{\mathcal{A}} s''$ then $s' = s''$.
4. (Wang's axiom) $s \xrightarrow{d}_{\mathcal{A}} s'$ iff $\exists s'' : s \xrightarrow{d'}_{\mathcal{A}} s''$ and $s'' \xrightarrow{d-d'}_{\mathcal{A}} s'$.

As timed probabilistic I/O automata are enriched probabilistic automata, we can use the notions of nonprobabilistic variant, reachable state, and execution (fragment), also for timed probabilistic I/O automata.

We say that an execution α of $\mathcal{A}$ is *diverging* if the sum of the time-passage actions in α is ∞.

Definition 7.4.7 Let $\mathcal{A}, \mathcal{B}$ be timed or untimed probabilistic I/O automata. A function r is a *probabilistic (hyper)step refinement* from $\mathcal{A}$ to $\mathcal{B}$ if r is a probabilistic (hyper)step refinement from the underlying probabilistic automaton of $\mathcal{A}$ to the underlying probabilistic automaton of $\mathcal{B}$.

In [SV02a], it is argued that, under certain assumptions (met by the automata studied in this section), $\sqsubseteq_{\mathrm{TD}}$ can be used as a safety and liveness preserving implementation relation between timed I/O automata. In addition, the relation $\sqsubseteq_{\mathrm{DFTD}}$ is proposed as a safety and liveness preserving implementation relation between timed probabilistic I/O automata and probabilistic I/O automata.

Claim 7.4.8 ([SV02a]) *Let $\mathcal{A}$ be a timed probabilistic I/O automaton and let $\mathcal{B}$ be a probabilistic I/O automaton. Let r be a probabilistic step refinement from $hide(\mathcal{A}, \mathbb{R}^{>0})$ to $\mathcal{B}$ that relates each divergent execution of $\mathcal{A}$ only to fair executions of $\mathcal{B}$. Then $\mathcal{A} \sqsubseteq_{\mathrm{DFTD}} \mathcal{B}$.*

7.4.2 The Protocol model

An informal description of the protocol has been given in Section 7.2. The timed probabilistic I/O automata describing the behavior of these nodes are given in Figure 7.7, using the IOA syntax of [GLV97] extended with a simple form of probabilistic choice. For simplicity, we refer to the two contending nodes as node 1 and node 2. Roughly, the idea behind the protocol model is as follows. When node `p` has detected root contention, it first flips a coin (i.e., performs the action `Flip(p)`). If head comes up then it waits a short time, somewhere in the interval $[rc\_fast\_min, rc\_fast\_max]$. If tail comes up then it waits a long time, somewhere in the interval $[rc\_slow\_min, rc\_slow\_max]$. After the waiting period has elapsed, either no message

[1] For simplicity the conditions here are slightly more restrictive than those in [LV96b].

type $\mathtt{P} = \mathbf{enumeration\ of}\ \mathtt{1}, \mathtt{2}$
type $\mathcal{M} = \mathbf{enumeration\ of}\ \bot, \mathit{req}, \mathit{ack}$
type $\mathsf{Status} = \mathbf{enumeration\ of}\ \mathit{unknown}, \mathit{root}, \mathit{child}$
type $\mathsf{Toss} = \mathbf{enumeration\ of}\ \mathit{head}, \mathit{tail}$
automaton `Node(p: P)`
  states
    $\mathtt{status} : \mathsf{Status} := \mathit{unknown}$,
    $\mathtt{coin} : \mathsf{Toss}$,
    $\mathtt{snt} : \mathcal{M} := \mathit{req}$,
    $\mathtt{rec} : \mathcal{M} := \mathit{req}$,
    $\mathtt{x} : \mathsf{Reals} := 0$
  signature
    input `Receive(`**const** `i, m: M)` **where** $\mathtt{m} \neq \bot$
    output `Send(`**const** `i, m: M)` **where** $\mathtt{m} \neq \bot$,
      `Root(`**const** `i)`
    internal `Flip(`**const** `i)`,
      `Child(`**const** `i)`
    delay `Time(d: Reals)` **where** $\mathtt{d} > 0$
  transitions
    internal `Flip(p)`
      pre $\mathtt{status} = \mathit{unknown} \wedge \mathtt{snt} = \mathit{req} \wedge \mathtt{rec} = \mathit{req}$
      eff $\mathtt{coin} := \begin{cases} \mathit{head} & \frac{1}{2} \\ \mathit{tail} & \frac{1}{2} \end{cases}$;
        $\mathtt{x} := 0$;
        $\mathtt{snt} := \bot$;
        $\mathtt{rec} := \bot$
    output `Send(p, m)`
      pre $\mathtt{status} = \mathit{unknown} \wedge \mathtt{snt} = \bot$
        $\wedge\ \mathtt{x} \geq \mathsf{rc\_min}(\mathtt{coin})$
        $\wedge\ \mathtt{m} = \mathbf{if}\ \mathtt{rec} = \bot\ \mathbf{then}\ \mathit{req}\ \mathbf{else}\ \mathit{ack}$
      eff $\mathtt{snt} := \mathtt{m}$
    input `Receive(p, m)`
      eff $\mathtt{rec} := \mathtt{m}$
    output `Root(p)`
      pre $\mathtt{status} = \mathit{unknown} \wedge \mathtt{snt} = \mathit{ack}$
      eff $\mathtt{status} := \mathit{root}$
    internal `Child(p)`
      pre $\mathtt{status} = \mathit{unknown} \wedge \mathtt{rec} = \mathit{ack}$
      eff $\mathtt{status} := \mathit{child}$
    delay `Time(d)`
      pre $\mathtt{status} = \mathit{unknown} \Rightarrow$
        $(\mathtt{snt} \neq \mathit{ack} \wedge \mathtt{rec} \neq \mathit{ack} \wedge \neg(\mathtt{snt} = \mathit{req} \wedge \mathtt{rec} = \mathit{req})$
        $\wedge\ \mathtt{snt} = \bot \Rightarrow \mathtt{x} + \mathtt{d} \leq \mathsf{rc\_max}(\mathtt{coin}))$
      eff $\mathtt{x} := \mathtt{x} + \mathtt{d}$

Figure 7.7: Node automaton.

from the contender has been received, or a parent request has arrived[2]. In the first case the node sends a request message to its contender (i.e., performs the action $\mathtt{Send}(\mathtt{p},\ req)$), in the second case it sends an acknowledgement message (i.e., performs the action $\mathtt{Send}(\mathtt{p},\ ack)$). As soon as a node has sent an acknowledgement, it declares itself to be the root (via the action $\mathtt{Root}(\mathtt{p})$), and whenever a node has received an acknowledgement it assumes that its contender will become root and it declares itself child (via the action $\mathtt{Child}(\mathtt{p})$). If a node that has sent a request subsequently receives a request, then it concludes that there is root contention again. In that case, the protocol starts all over again. The basic idea behind the protocol is that if the outcomes of the coin flips are different, the node with outcome tail (i.e., the slow one) will become root. And since with probability one the outcomes of the two coin flips will eventually be different, the root contention protocol will terminate (with probability one).

The timed probabilistic I/O automaton for node $\mathtt{p}$ ($\mathtt{p} = 1, 2$), displayed in Figure 7.7, has five state variables: variable $\mathtt{status}$ tells whether the node has become $root$, $child$, or whether its status is still $unknown$; variable $\mathtt{coin}$ records the outcome of the coin flip; variable $\mathtt{snt}$ records the last value (if any) that has been sent to the contender and may take values req, ack or $\perp$; similarly $\mathtt{rec}$ records the last value that has been received (if any); variable $\mathtt{x}$, finally, models the arbitration timer that records the time that has elapsed since root contention has been detected. We use two auxiliary functions $\mathsf{rc\_min}(\mathsf{c})$ and $\mathsf{rc\_max}(\mathsf{c})$ from Toss to Reals given by, for $\mathsf{c} \in \mathsf{Toss}$,

$$\begin{aligned} \mathsf{rc\_min}(\mathsf{c}) &\triangleq \textbf{if } \mathsf{c} = head \textbf{ then } \textit{rc\_fast\_min} \textbf{ else } \textit{rc\_slow\_min} \\ \mathsf{rc\_max}(\mathsf{c}) &\triangleq \textbf{if } \mathsf{c} = head \textbf{ then } \textit{rc\_fast\_max} \textbf{ else } \textit{rc\_slow\_max} \end{aligned}$$

Now it should not be difficult to understand the precondition/effect style definitions in Figure 7.7, except maybe for the definition of the $\mathtt{Time}(\mathtt{d})$ transitions. This part states that time will not progress if the status of the node is unknown and (1) an acknowledgement has been sent, or (2) an acknowledgement has been received, or (3) a parent request has both been sent and received. In the first case the automaton will instantaneously perform a $\mathtt{Root}(\mathtt{p})$ action, in the second case it will perform a $\mathtt{Child}(\mathtt{p})$ action, and in the third case there is contention and the automaton will flip a coin.[3] The last clause in the precondition of $\mathtt{Time}(\mathtt{d})$ enforces that a $\mathtt{Send}(\mathtt{p},\ \mathtt{m})$ action is performed within either *rc_fast_max* or *rc_slow_max* time after the coin flip (depending on the outcome). Once the status of the automaton has become $root$ or $child$ there are no more restrictions on time passage.

The two automata for node 1 and node 2 communicate via wires, which are modeled as the timed probabilistic automata $\mathtt{Wire}(1,\ 2)$ and $\mathtt{Wire}(2,\ 1)$ specified in Figure 7.8. We assume an upper bound $delay \geq 0$ on the communication delay.

The full system $\mathtt{Impl}^{\mathtt{A}}$ can now be described as the parallel composition of the two node automata and the two wire automata, with all synchronization actions hidden (see Figure 7.9).

Remark 7.4.9 As Segala [Seg95b] points out in his thesis, it would be useful to study the theory of *receptiveness* [SGSL98] in the context of randomization. As far as we know, nobody has taken up this challenge yet. Intuitively, an automaton is receptive if it does not

[2]The IDLE signal is not modeled explicitly here and this model is therefore not completely conform standard, but it does provide useful insight in the probabilistic and real–time behavior.

[3]Note that in each of these three cases we abstract in our model from the computation time required to perform these actions.

```
automaton Wire(i:P, j: P)
  states
    msg : 𝓜 :=⊥,
    x : Reals := 0
  signature
    input Send(const i, m: M) where m ≠⊥
    output Receive(const j, m: M) where m ≠⊥
    delay Time(d:Reals) where d > 0
  transitions
    input Send(i, m)
      eff msg := m;
          x := 0
    output Receive(j, m)
      pre m = msg
      eff msg :=⊥
    delay Time(d)
      pre msg ≠⊥⇒ x + d ≤ delay
      eff x := x + d
```

Figure 7.8: Wire automaton.

$$\mathtt{Impl}^{\mathtt{A}} \triangleq \textbf{hide } \mathtt{Send(i, m), Receive(i, m)} \textbf{ for } \mathtt{i} : \mathtt{P}, \mathtt{m} : \mathcal{M} \textbf{ in } \textbf{compose } \mathtt{Node(1); Wire(1, 2); Node(2); Wire(2, 1)}$$

Figure 7.9: The full system.

constrain its environment, for instance by not accepting certain inputs or by preventing time to pass beyond a certain point. Behavior inclusion is used as an implementation relation in the I/O automata framework and we exclude trivial implementations by requiring that an implementation is receptive.

If we replace all probabilistic choices by nondeterministic choices in the automata of this section, then the resulting timed I/O automata are receptive in the sense of [SGSL98]. Even with a more restrictive definition of receptivity, in which we allow the environment to resolve all probabilistic choices, the automata of this section remain receptive.

7.4.3 Verification and Analysis

Of course, the key correctness property of the root contention protocol which we would like to prove is that eventually exactly one node is designated as root. This correctness property is described by the two state probabilistic I/O automaton Spec of Figure 7.10. We will establish that $\mathtt{Impl}^{\mathtt{A}}$ implements Spec, provided the following two constraints on the parameters are

automaton `Spec`
  states
    `done` : Bool $:= \mathit{false}$
  signature
    output `Root(i: P)`
  transitions
    output `Root(i)`
      pre `done` $= \mathit{false}$
      eff `done` $= \mathit{true}$
  tasks
    One block

Figure 7.10: Specification.

met:[4]

$$\mathit{delay} < \mathit{rc\_fast\_min} \qquad (Eq_0)$$
$$2 \cdot \mathit{delay} < \mathit{rc\_slow\_min} - \mathit{rc\_fast\_max}. \qquad (Eq_2)$$

Moreover, we assume the basic constraint

$$\mathit{rc\_fast\_min} \leq \mathit{rc\_fast\_max} \leq \mathit{rc\_slow\_min} \leq \mathit{rc\_slow\_max}.$$

Note that this is slightly stronger than the basic constraints $B_1 - B_3$ from Section 7.2. Moreover, recall that the parameters range over $\mathbb{R}^{\geq 0}$.

Within our proof, we introduce three intermediate automata $\mathtt{I}_1^{\mathtt{A}}$, $\mathtt{I}_2^{\mathtt{A}}$ and $\mathtt{I}_3^{\mathtt{A}}$, and prove that

$$\mathtt{Impl}^{\mathtt{A}} \sqsubseteq_{\mathrm{TD}} \mathtt{I}_1^{\mathtt{A}} \sqsubseteq_{\mathrm{TD}} \mathtt{I}_2^{\mathtt{A}} \sqsubseteq_{\mathrm{TD}} \mathtt{I}_3^{\mathtt{A}} \sqsubseteq_{\mathrm{TD}} \mathsf{Spec}.$$

These results (or more precisely the refinements that are established in their proofs) are then used to obtain that

$$\mathtt{Impl}^{\mathtt{A}} \sqsubseteq_{\mathrm{TD}} \mathtt{I}_1^{\mathtt{A}} \sqsubseteq_{\mathrm{DFTD}} \mathtt{I}_2^{\mathtt{A}} \sqsubseteq_{\mathrm{FTD}} \mathtt{I}_3^{\mathtt{A}} \sqsubseteq_{\mathrm{FTD}} \mathsf{Spec}.$$

$\mathtt{I}_1^{\mathtt{A}}$ is a timed probabilistic I/O automaton, which abstracts from all the message passing in $\mathtt{Impl}^{\mathtt{A}}$, while preserving the probabilistic choices as well as most information about the timing of the `Root(i)` events. $\mathtt{I}_2^{\mathtt{A}}$ is a probabilistic I/O automaton which is identical to $\mathtt{I}_1^{\mathtt{A}}$, except that all real-time information has been omitted. In $\mathtt{I}_3^{\mathtt{A}}$ the two coin flips from each node of the protocol are combined into a single probabilistic transition.

Invariants

We will show that there exists a probabilistic step refinement from $\mathtt{Impl}^{\mathtt{A}}$ to an intermediate automaton $\mathtt{I}_1^{\mathtt{A}}$. To establish a refinement, we first need to introduce a number of invariants for automaton $\mathtt{Impl}^{\mathtt{A}}$.

We use subscripts 1 and 2 to refer to the state variables of `Node(1)` and `Node(2)`, respectively, and subscripts 12 and 21 to refer to the state variables of `Wire(1, 2)` and `Wire(2, 1)`,

[4] As mentioned before, the timing constraints here are different from those in Section 7.2, due to an imperfection in the communication model.

respectively. So, $\mathtt{x}_1$ denotes the clock variable of $\mathtt{Node}(1)$, $\mathtt{x}_{12}$ denotes the clock variable of $\mathtt{Wire}(1,\ 2)$, etc. Within formulas we further use the following abbreviations, for $\mathtt{i} \in \mathtt{P}$,

$$
\begin{aligned}
\mathsf{Cont}(\mathtt{i}) &\triangleq \mathtt{snt_i} = \mathit{req} \wedge (\mathtt{rec_i} = \mathit{req} \vee \mathtt{msg_{ji}} = \mathit{req}) \\
\mathsf{Wait}(\mathtt{i}) &\triangleq \mathtt{snt_i} = \mathtt{rec_i} = \perp \\
\mathsf{rc\_min}_\mathtt{i} &\triangleq \mathsf{rc\_min}(\mathtt{coin_i}) \\
\mathsf{rc\_max}_\mathtt{i} &\triangleq \mathsf{rc\_max}(\mathtt{coin_i})
\end{aligned}
$$

Predicate $\mathsf{Cont}(\mathtt{i})$ states that node $\mathtt{i}$ has either detected contention (a request has both been sent and received) or will do so in the near future (the node has sent a request and will receive one soon). Predicate $\mathsf{Wait}(\mathtt{i})$ states that node has flipped the coin and is waiting for the delay time to expire; no message has been received yet. State function $\mathsf{rc\_min}_\mathtt{i}$ gives the minimum delay time for node $\mathtt{i}$, and state function $\mathsf{rc\_max}_\mathtt{i}$ the maximum delay time (both state functions depend on the outcome of the coin flip).

We claim that assertions (7.1)-(7.17) below are invariants of automaton $\mathtt{Impl}^\mathtt{A}$.

$$
\begin{aligned}
&\mathtt{x_i} \geq 0 && (7.1)\\
&\mathtt{status_i} = \mathit{unknown} \wedge \mathtt{snt_i} \neq \mathit{req} \Rightarrow \mathtt{x_i} \leq \mathsf{rc\_max}_\mathtt{i} && (7.2)\\
&\mathtt{snt_i} = \mathit{ack} \Rightarrow \mathtt{x_i} \geq \mathsf{rc\_min}_\mathtt{i} && (7.3)\\
&\mathtt{status_i} = \mathit{root} \Rightarrow \mathtt{snt_i} = \mathit{ack} && (7.4)\\
&\mathtt{status_i} = \mathit{child} \Rightarrow \mathtt{rec_i} = \mathit{ack} && (7.5)\\
&\mathtt{x_{ij}} \geq 0 && (7.6)\\
&\mathtt{msg_{ij}} \neq \perp \Rightarrow \mathtt{x_{ij}} \leq \mathit{delay} && (7.7)\\
&\mathsf{Cont}(\mathtt{i}) \Leftrightarrow \mathsf{Cont}(\mathtt{j}) \Rightarrow \mid \mathtt{x_i} - \mathtt{x_j} \mid \ \leq \mathit{delay} && (7.8)\\
&\mathsf{Cont}(\mathtt{i}) \wedge \neg\mathsf{Cont}(\mathtt{j}) \Rightarrow \mathsf{Wait}(\mathtt{j}) \wedge \mathtt{msg_{ij}} = \perp \wedge \mathtt{x_j} \leq \mathit{delay} && (7.9)\\
&\mathtt{msg_{ij}} \neq \perp \Rightarrow \mathtt{rec_j} = \perp && (7.10)\\
&\mathtt{msg_{ij}} = \perp \Rightarrow \mathtt{snt_i} = \perp \vee \mathtt{rec_j} \neq \perp \vee \mathsf{Cont}(\mathtt{i}) && (7.11)
\end{aligned}
$$

$$
\begin{aligned}
&\mathtt{msg_{ij}} = \mathit{req} \wedge \neg\mathsf{Wait}(\mathtt{i}) \Rightarrow \mathtt{snt_i} = \mathit{req} \wedge \mathtt{snt_j} \neq \mathit{ack} \wedge \\
&\qquad\qquad \mathsf{rc\_min}_\mathtt{i} \leq \mathtt{x_i} - \mathtt{x_{ij}} \leq \mathsf{rc\_max}_\mathtt{i} && (7.12)\\
&\mathtt{msg_{ij}} = \mathit{req} \wedge \mathsf{Wait}(\mathtt{i}) \Rightarrow \mathtt{snt_j} = \mathit{req} \wedge \mathtt{x_i} \leq \mathtt{x_{ij}} && (7.13)\\
&\mathtt{snt_i} = \perp \wedge \mathtt{rec_i} = \mathit{req} \Rightarrow \mathtt{snt_j} = \mathit{req} \wedge \mathtt{rec_j} = \perp \wedge \mathtt{x_j} \geq \mathsf{rc\_min}_\mathtt{j} && (7.14)\\
&\mathtt{rec_i} = \mathit{ack} \Rightarrow \mathtt{snt_j} = \mathit{ack} && (7.15)\\
&\mathtt{msg_{ij}} = \mathit{ack} \Rightarrow \mathtt{snt_i} = \mathit{ack} && (7.16)\\
&\mathtt{snt_i} = \mathit{ack} \Rightarrow \mathtt{rec_i} = \mathtt{snt_j} = \mathit{req} \wedge \mathtt{rec_j} \neq \mathit{req} \wedge \mathtt{x_j} \geq \mathsf{rc\_min}_\mathtt{j} && (7.17)
\end{aligned}
$$

Assertions (7.1)-(7.7) are local invariants, which can be proven straightforwardly for automata $\mathtt{Node}(\mathtt{i})$ and $\mathtt{Wire}(\mathtt{i},\ \mathtt{j})$ in isolation. Most of the time nodes 1 and 2 are either both in contention or both not in contention. Assertion (7.8) states that in these cases the values of the clocks of the two nodes differ by at most delay. Assertion (7.9) expresses that the only case where node $\mathtt{i}$ is in contention but the other node $\mathtt{j}$ is not occurs when $\mathtt{j}$ has just flipped a coin but the request message that $\mathtt{j}$ sent to $\mathtt{i}$ has not yet arrived or been processed. If a channel contains a message then nothing has been received at the end of this channel

(7.10). If the channel from i to j is empty then either no message has been sent into the channel at i, or a message has been received at j, or we have a situation where i is in contention and j has just flipped a coin and moved to a new phase (7.11). If the channel from i to j contains a request message then there are two possible cases. Either i has sent the message and is waiting for a reply (7.12), or there is contention and i has just flipped a coin (7.13). If i has received a request message without having sent anything, then j has sent this message but has not received anything (7.14). The last three invariants deal with situations where there is an acknowledgement somewhere in the system (7.15)-(7.17). In these cases the global state is almost completely determined: if an acknowledgement is in a channel or has been received then it has been sent, and if a node has sent an acknowledgement then it has received a request, which in turn has been sent by the other node.

The automaton $\mathtt{Impl}^{\mathtt{A}^-}$ and the invariants were translated into Uppaal's input language. This allowed us to check the invariants mechanically. This was done for a large number of parameter values, just as in the approach taken in Section 7.5.

The proofs of the following two lemmas are tedious but completely standard since they only refer to the non-probabilistic automaton $\mathtt{Impl}^{\mathtt{A}^-}$. Detailed proofs can be obtained via URL www.cs.kun.nl/~fvaan/PAPERS/SVproofs.

Lemma 7.4.10 *Suppose state s satisfies assertions (7.1)-(7.17) and $s \xrightarrow{\mathtt{Send(i,\,m)}} s'$. Then $s \models \mathtt{msg_{ij}} = \mathtt{rec_j} = \perp$ and $s' \models \mathsf{Cont}(\mathtt{i}) \Leftrightarrow \mathsf{Cont}(\mathtt{j})$.*

Lemma 7.4.11 *Assertions (7.1)-(7.17) hold for all reachable states of* $\mathtt{Impl}^{\mathtt{A}}$.

Remark 7.4.12 The constraint Eq_0 on the timing parameters is used to prove Lemma 7.4.10 and ensures that there can never be two messages traveling in a wire at the same time. This property allows for a very simple model of the wires, in which a new message overwrites an old message. Since Lemma 7.4.10 is used to prove Lemma 7.4.11, the first constraint is also used in our proof of Lemma 7.4.11.

The parameter constraint Eq_2 is not used in either of the proofs above.

The first intermediate automaton

Intermediate automaton $\mathtt{I}_1^{\mathtt{A}}$ is displayed in Figure 7.11.

This probabilistic timed I/O automaton records the status for each of the two nodes to be either $init$, $head$, $tail$, or $done$. In addition $\mathtt{I}_1^{\mathtt{A}}$ maintains a clock x to impose timing constraints between events. Apart from the delay action there are three actions: Flip(i), which corresponds to node i flipping a coin, Root(i), which corresponds to node i declaring itself to be the root, and Retry(c), which models the restart of the protocol in the case where the outcome of both coin flips is c. Node i performs a (probabilistic) Flip(i) action in its initial state. A Root(i) transition may occur if both nodes have flipped a coin and it is *not* the case that the outcome for i is $head$ and for j $tail$. A Retry(c) transition may occur if both nodes have flipped c. Clock x is used to express that both nodes flip their coin within time $delay$ after the (re-)start of the protocol. In addition it ensures that subsequently (depending on the outcome of the coin flips) at least *rc_fast_min* – $delay$ or *rc_slow_min* – $delay$ time and at most *rc_fast_max* or *rc_slow_max* time will elapse before either a Root(i) or a Retry(c) action occurs.

```
automaton I_1^A
  type Phase = enumeration of init, head, tail, done
  states
    phase : Array[P, Phase] := constant(init),
    x : Reals := 0
  signature
    output Root(i: P)
    internal Flip(i: P),
             Retry(c: Toss)
    delay Time(d: Reals) where d > 0
  transitions
    internal Flip(i)
      pre phase[i] = init
      eff phase[i] := { head  1/2
                      { tail  1/2  ;
          if phase[next(i)] ≠ init then x := 0
    output Root(i)
      pre {phase[1], phase[2]} ⊆ {head, tail}
          ∧ ¬(phase[i] = head ∧ phase[next(i)] = tail)
          ∧ x ≥ rc_min(phase[i]) − delay
      eff phase := constant(done)
    internal Retry(c)
      pre phase = constant(c)
          ∧ x ≥ rc_min(c)
      eff phase := constant(init);
          x := 0
    delay Time(d)
      pre init ∈ {phase[1], phase[2]} ⇒ x + d ≤ delay
          ∧ {phase[1], phase[2]} ⊆ {head, tail} ⇒
              x + d ≤ max(rc_max(phase[1]), rc_max(phase[2]))
      eff x := x + d
```

Figure 7.11: Intermediate automaton $\mathtt{I}_1^{\mathtt{A}}$.

Proposition 7.4.13 $\mathtt{Impl}^{\mathtt{A}} \sqsubseteq_{\mathrm{TD}} \mathtt{I}_1^{\mathtt{A}}$. *More specifically the conjunction, for* $\mathtt{i} \in \mathtt{P}$*, of*

$$\begin{array}{rcl} \mathtt{phase}[\mathtt{i}] & = & \textbf{if } \mathtt{status}_1 = \mathit{root} \vee \mathtt{status}_2 = \mathit{root} \textbf{ then } \mathit{done} \textbf{ else} \\ & & \quad \textbf{if } \mathsf{Cont}(\mathtt{i}) \textbf{ then } \mathit{init} \textbf{ else } \mathtt{coin}_\mathtt{i} \textbf{ fi fi} \\ \mathtt{x} & = & \textbf{if } \mathsf{Cont}(1) \vee \mathsf{Cont}(2) \textbf{ then } \min(\mathtt{x}_{12}, \mathtt{x}_{21}) \textbf{ else } \min(\mathtt{x}_1, \mathtt{x}_2) \end{array}$$

determines a probabilistic step refinement from $\mathtt{Impl}^{\mathtt{A}}$ *to* $\mathtt{I}_1^{\mathtt{A}}$.

PROOF: Routine. See `http://www.cs.kun.nl/~fvaan/PAPERS/SVproofs`. □

Remark 7.4.14 The constraint Eq_2 on the timing parameters is used in the proof of Proposition 7.4.13 and ensures that contention may only occur if the outcomes of both coin flips are the same. This property is needed to prove termination of the algorithm (with probability 1).

The first constraint Eq_0 is also used in our proof, since our proof of Proposition 7.4.13 uses Lemma 7.4.11, which in its turn uses the first constraint. However, we conjecture that if we prove the inclusion between $\mathtt{Impl}^{\mathtt{A}}$ and $\mathtt{Spec}$ without using intermediate automata, we do not need the first constraint. In fact, this constraint is an implicit assumption in the model, needed to justify the modeling of the communication channels by a one–place buffer.

Remark 7.4.15 In [And98] it is claimed that if both nodes happen to select slow timing or if both nodes select fast timing, contention results again. This is incorrect. In automaton $\mathtt{I}_1^{\mathtt{A}}$ each of the two nodes may become root if both nodes happen to select the same timing delay. This may also occur within a real–world implementation of the protocol: if in the implementation the timing parameters of one node are close to their minimum values, in the other node close to their maximum values, and if the communication delay is small, then it may occur that a request message of node $\mathtt{i}$ arrives at node $\mathtt{j}$ before the timing delay of node $\mathtt{j}$ has expired. In fact, by instantiating the timing parameters differently in different devices (for instance via some random mechanism!) one may reduce the expected time to resolve contention. Unfortunately, a more detailed analysis of this phenomenon falls outside the scope of this section.

Remark 7.4.16 Another way in which the performance of the protocol could be improved is by repeatedly polling the input during the timing delay, rather than checking it only at the end. We suggest that, if the process receives a request when the timing delay has not yet expired, then it immediately sends an acknowledgement (and declares itself root). If the process has not received a request during the timing delay, then it sends a request and proceeds as the current implementation. In a situation where node i flips head and selects a timing delay of *rc_fast_min* and the other node j flips tail and selects a timing delay of *rc_slow_max*, our version elects a leader within at most *rc_fast_min* $+ 3\mathit{delay}$, whereas in the current version this upper bound is *rc_slow_max* $+ 3\mathit{delay}$.

The second intermediate automaton

In Figure 7.12 the second intermediate automaton $\mathtt{I}_2^{\mathtt{A}}$ is described. $\mathtt{I}_2^{\mathtt{A}}$ is a probabilistic I/O automaton that is identical to $\mathtt{I}_1^{\mathtt{A}}$ except that all real-time information has been abstracted away; instead a (trivial) task partition is included. The proof of the following Proposition 7.4.17 is easy: the projection function π from $\mathtt{I}_1^{\mathtt{A}}$ to $\mathtt{I}_2^{\mathtt{A}}$ trivially is a probabilistic step refinement (after hiding of the time delays).

```
automaton I2^A
  states
    phase : Array[P, Phase] := constant(init)
  signature
    output Root(i: P)
    internal Flip(i: P),
             Retry(c: Toss)
  transitions
    internal Flip(i)
      pre phase[i] = init
      eff phase[i] := { head 1/2
                      { tail 1/2
    output Root(i)
      pre {phase[1], phase[2]} ⊆ {head, tail}
        ∧ ¬(phase[p] = head ∧ phase[next(p)] = tail)
      eff phase := constant(done)
    internal Retry(c)
      pre phase = constant(c)
      eff phase := constant(init)
  tasks
    One block
```

Figure 7.12: Intermediate automaton $\mathtt{I}_2^{\mathtt{A}}$.

Proposition 7.4.17 $\mathtt{I}_1^{\mathtt{A}} \sqsubseteq_{\mathrm{TD}} \mathtt{I}_2^{\mathtt{A}}$.

Proposition 7.4.18 *If $\alpha \in execs(\mathtt{I}_1^{\mathtt{A}})$ is diverging and π relates α and β, then β is fair.*

The result formulated in the Proposition 7.4.18 above follows by the fact that a diverging execution of $\mathtt{I}_1^{\mathtt{A}}$ either contains infinitely many `Retry` actions, or contains an infinite suffix with a `Root(p)` transition followed by an infinite number of delay transitions. Now the Claim 7.4.8 implies $\mathtt{I}_1^{\mathtt{A}} \sqsubseteq_{\mathrm{DFTF}} \mathtt{I}_2^{\mathtt{A}}$.

The third intermediate automaton

Figure 7.13 gives the IOA code for the probabilistic I/O automaton $\mathtt{I}_3^{\mathtt{A}}$. This automaton abstracts from $\mathtt{I}_2^{\mathtt{A}}$ since it only has a single probabilistic transition. Within automaton $\mathtt{I}_3^{\mathtt{A}}$, $init$ is the initial state and $done$ is the final state in which a root has been elected. The remaining states win_1, win_2, $same$ correspond to situations in which both processes have flipped but no leader has been elected yet. The value win_{p} indicates that the results are different and the outcome of p equals tail. In state $same$ both coin flips have yielded the same result.

Proposition 7.4.19 $\mathtt{I}_2^{\mathtt{A}} \sqsubseteq_{\mathrm{TD}} \mathtt{I}_3^{\mathtt{A}}$. *More specifically, the following function r from (reachable) states of $\mathtt{I}_2^{\mathtt{A}}$ to discrete probability spaces over states of $\mathtt{I}_3^{\mathtt{A}}$ is a probabilistic hyperstep refine-*

automaton $\mathtt{I}_3^{\mathtt{A}}$
  type $\mathsf{Loc} = \textbf{enumeration of } \ init, win_1, win_2, same, done$
  states
    $\mathtt{loc} : \mathsf{Loc} := init$
  signature
    output `Root(p:  P)`
    internal `Flips`,
        `Retry`
  transitions
    internal `Flips`
      pre $\mathtt{loc} = init$
      eff $\mathtt{loc} := \begin{cases} win_1 & \frac{1}{4} \\ win_2 & \frac{1}{4} \\ same & \frac{1}{2} \end{cases}$
    output `Root(p)`
      pre $\mathtt{loc} \in \{win_{\mathtt{p}}, same\}$
      eff $\mathtt{loc} := done$
    internal `Retry`
      pre $\mathtt{loc} = same$
      eff $\mathtt{loc} := init$
  tasks
    One block

Figure 7.13: Intermediate automaton $\mathtt{I}_3^{\mathtt{A}}$.

ment from $\mathtt{I}_2^{\mathtt{A}}$ *to* $\mathtt{I}_3^{\mathtt{A}}$ *(we represent a state with a list containing the values of its variables):*

$$\begin{array}{rcl}
r(init, init) & = & \{init \mapsto 1\} \\
r(head, init) & = & \{win_2 \mapsto \frac{1}{2}, same \mapsto \frac{1}{2}\} \\
r(init, head) & = & \{win_1 \mapsto \frac{1}{2}, same \mapsto \frac{1}{2}\} \\
r(tail, init) & = & \{win_1 \mapsto \frac{1}{2}, same \mapsto \frac{1}{2}\} \\
r(init, tail) & = & \{win_2 \mapsto \frac{1}{2}, same \mapsto \frac{1}{2}\} \\
r(head, head) & = & \{same \mapsto 1\} \\
r(tail, tail) & = & \{same \mapsto 1\} \\
r(head, tail) & = & \{win_2 \mapsto 1\} \\
r(tail, head) & = & \{win_1 \mapsto 1\} \\
r(done, done) & = & \{done \mapsto 1\}
\end{array}$$

The proofs of the following Propositions 7.4.20 and 7.4.21 can be found in [Sto99b]. These proofs are the only places in our verification where nontrivial probabilistic reasoning takes place: establishing $\sqsubseteq_{\mathrm{FTD}}$ basically amounts to proving that the probabilistic mechanism in the protocol ensures termination with probability 1. Note that the automata involved are all very simple: $\mathtt{I}_2^{\mathtt{A}}$ has 10 states, $\mathtt{I}_3^{\mathtt{A}}$ has 5 states, and `Spec` has 2 states.

Proposition 7.4.20 $\mathtt{I}_2^{\mathtt{A}} \sqsubseteq_{\mathrm{FTD}} \mathtt{I}_3^{\mathtt{A}}$.

Proposition 7.4.21

1. $\mathtt{I}_3^{\mathtt{A}} \sqsubseteq_{\mathrm{TD}} \mathtt{Spec}$. *More specifically, the function determined by the predicate done* $\Leftrightarrow$ $\mathtt{loc} =$ *done is a probabilistic step refinement from* $\mathtt{I}_3^{\mathtt{A}}$ *to* $\mathtt{Spec}$.
2. $\mathtt{I}_3^{\mathtt{A}} \sqsubseteq_{\mathrm{FTD}} \mathtt{Spec}$.

7.4.4 Concluding Remarks

To make our verification easier to understand, we introduced three auxiliary automata in between the implementation and the specification automaton. We also used the simpler notion of probabilistic (hyper)step refinement (see Chapter 2) rather than the more general but also complex simulation relations (especially in the timed case!) which have been proposed by Segala and Lynch [Seg95b, SL95]. The complexity of the definitions in [Seg95b, SL95] is mainly due to the fact that a single step in one machine can in general be simulated by a sequence of steps in the other machine with the same external behavior. In the probabilistic case this means that a probabilistic transition in one machine can be simulated by a tree like structure in the other machine. In the simulations that we use in this section, a single transition in one machine is simulated by at most one transition in the other machine. In our case study we were able to carry out the correctness proof by using only probabilistic (hyper)step refinements.

The idea to introduce auxiliary automata in a simulation proof has been studied in many papers, see for instance [AL91]. The verification reported in this section indicates that the introduction of auxiliary automata can be very useful in the probabilistic case: it allowed us to first deal with the nonprobabilistic and real–time behavior of the protocol, basically without being bothered by the complications of randomization; nontrivial probabilistic analysis was only required for automata with 10 states or less.

7.5 Second Approach: Mechanical Verification of the Root Contention Protocol

In this section, we present a mechanical verification of RCP. We improve the protocol model from the previous section and then use the model checker Uppaal to investigate the timing constraints of the new protocol model. Since the probabilistic phenomena in the new protocol model are basically the same as before, we do not reconsider probabilistic aspects in this section.

As pointed out in Section 7.2.3, timing parameters play an essential role in the root contention protocol. There are currently three tools[5] available that (at least in some cases, see [AHV93]) can do parametric analysis of timed systems: HyTech [HHW97], PMC [gDUoT] and – very recently – the tool described in [AAB00]. Whereas HyTech and PMC can currently analyze and derive linear parameter constraints, [AAB00] describes a prototype implementation which can also deal with nonlinear constraints. Since the performance of HyTech is limited, and we expected the protocol to be too complex for it, and since only prototypes of the other two tools are currently available, we decided to use the Uppaal tool. Uppaal has already been used successfully in various verifications [HSLL97, DKRT97b]. It can model real–time systems with a finite control structure. A limited class of properties, viz. reachability properties, can be checked automatically and (relatively) efficiently. Our protocol models fit naturally into its input language.

[5] At the time we carried out this research, the prototype extension of Uppaal was in a very preliminary stage.

Using the enhanced model, we investigate the correct operation of the root contention protocol with Uppaal. By analyzing numerous instances of the protocol for different values of the parameters, Uppaal allowed us to do an approximate parameter analysis. The constraints that we deduce are exactly those from [LaF97].

Unlike most other Uppaal case studies, we carry out the verification using intermediate automata, as in Section 7.4. The intermediate automata used in this section are similar to the ones in the previous section, but have been adapted to match the new protocol model. To do so, we need several constructions on Uppaal models. The works [Jen99, JLS00] discuss an other approach to this problem. The difference with our work is that [Jen99, JLS00] consider timed automata without invariants but with shared variables and urgent channels and require the specification automaton to be deterministic. We use the notion of step refinement to deal with nondeterministic specifications in certain cases.

The rest of this section is organized as follows. In Section 7.5.1, we describe how Uppaal can be used in verifying system correctness via stepwise abstraction. Section 7.5.2 gives the semantics of a Uppaal model. Moreover, this section explains how Uppaal can be used to verify *trace inclusion*, i.e. that one model in Uppaal is a correct implementation of another one. Section 7.5.3 presents the new protocol model and Section 7.5.4 is concerned with the verification and analysis of these models. Finally, in Section 7.5.5, conclusions are given.

7.5.1 Verification of Trace Inclusion with Uppaal

Uppaal [Upp, LPY97] is a tool box for modeling, simulation and automatic verification of real–time systems, based on timed automata. In the present case study, we used the Uppaal2k version. Uppaal can simulate a model, i.e. it can provide a particular execution of the model step by step, and it can automatically check whether a given reachability property is satisfied and, if so, provide a trace showing how the property is satisfied.

Model checking with Uppaal

This section gives an informal introduction to Uppaal and is based on [LPY97]. A system in Uppaal is modeled as a network of nondeterministic processes (called automata) with a finite control structure and real–valued clocks. Communication between the processes takes place via channels (via binary synchronization on complementary actions). Within Uppaal it is possible to model automata via a graphical description. Furthermore, templates are available to facilitate the specification of multiple automata with the same control structure.

Basically, a process is a finite state machine (or labeled transition system) extended with clock variables. The nodes of an automaton describe the control locations. Each location can be decorated with an invariant: a number of clock bounds expressing the range of clock values that are allowed in that location. The edges of the transition system represent changes in control locations. Each edge can be labeled by a *guard* g, an *action (label)* a, and a collection r of *clock resets*. All three types of labels are optional. A guard is a boolean combination of inequalities over clock variables, expressing when the transition is enabled, i.e. when it can be taken. Upon taking a transition, the clock resets, if present, are executed. The action label, if present, enforces binary synchronization. This means that exactly one of the other processes has to take a complementary action (where a! and a? are complementary). If no other process is able to synchronize on the action, the transition is not enabled. A process can have a location from which more than one transition is enabled with the same action. In this way, nondeterministic choices can be specified within Uppaal. Note that time can only

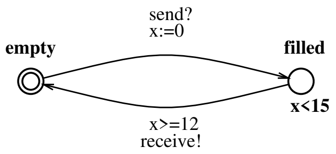


Figure 7.14: An example Uppaal process automaton (`Buffer`)

elapse at the locations (conform invariants). Transitions are taken instantaneously, i.e. no time elapses during transitions. Three special types of locations are available:

1. *Initial locations*, denoted by (O), define the initial state of the system (exactly one per process).

2. *Urgent locations*, denoted by (U), are locations in which no time can be spent, hence a shorthand notation for a location that satisfies the virtual invariant $x \leq 0$. The (fresh) clock x is reset on all transitions to the urgent location.

3. *Committed locations*, denoted by (C), are used to make the incoming and the outgoing transition atomic. Being in a committed location, the process execution cannot be interrupted and no time elapses. We used these locations to encode multi–way synchronization in Uppaal (see Section A.1.2).

Moreover, channels can declared as being urgent and shared integer variables can be used to communicate between the processes. Since we did not use these Uppaal features in our verification, we do not describe them here.

Consider the Buffer process automaton displayed in Figure 7.14. This automaton models a one–place buffer which delivers a message with a time delay between 12 and 15 time units.

Uppaal is able to analyze reachability properties automatically. These properties must be of the form $\mathrm{E}\Diamond p$ or $\mathrm{A}\Box p$, where p is Boolean expression over clock constraints and locations of the automata. For example, $\mathrm{E}\Diamond\mathtt{Buffer.filled} \wedge x > 13$ is a property over the automaton `Buffer`. Informally, $\mathrm{E}\Diamond p$ denotes that there must be some state (= location + clock values) which is reachable from the initial state and which satisfies p. This logic is sufficient to specify reachability properties, invariants, and bounded liveness properties. For the latter, see [ABK98]. However, general liveness and fairness properties, e.g.whether an event occurs infinitely often, cannot be expressed in Uppaal.

Notational conventions

Throughout this chapterwe use the following conventions for automata.

We assume that the automata do not have urgent channels, committed locations, and shared variables and we assume that any two components in a network use different clocks.

We denote the absence of a transition label by a special symbol τ. When convenient, we assume that all labels on a transition are present, interpreting the absence of a guard or invariant by true and the absence of a reset set by $\emptyset$. We denote the invariant of a location q by $\mathrm{Inv}(q)$.

Moreover, we assume a fixed set of action names, Names, such that $\tau \notin \mathrm{Names}$, and for any subset of $N \subseteq \mathrm{Names}$, we write $N! = \{a! \mid a \in N\}$, $N? = \{a? \mid a \in N\}$. Then the set of discrete actions is given by $act = \mathrm{Names}? \cup \mathrm{Names}! \cup \{\tau\}$. The action τ is called the *invisible* or *internal* action; the other actions are called *visible*. The set of *visible actions* occurring in automaton A is the denoted by Act_A. By abuse of notation, we let a, b range over act, but in the expressions $a?$ and $a!$, a ranges over action names.

7.5.2 The Semantics of Uppaal Models

Similarly to the verification in the previous section, we express the behavior of a system by the set of its traces and use trace inclusion to express that one system correctly implements another one. The Uppaal tool interprets networks of automata as *closed systems*, which are systems that cannot interact with their environment. In closed systems we cannot express many sets of traces. (The traces of closed systems consists of their time passage actions.) Therefore, we provide – in addition to the standard Uppaal semantics – an interpretation of Uppaal models as *open systems*, which still have the possibility to interact with the environment. Then we define two operators, parallel composition $\|$ and action restriction $\setminus$ to express the closed system semantics in terms of the open system semantics. Finally, we give a translation of each open model to a closed model with equivalent reachability properties. In this way, we are able to verify all reachability properties of open systems (and in particular trace inclusion) with Uppaal.

We use the same (Uppaal) syntax for open and for closed systems. We use the word *automaton* if we interpret the model as an open system and the term *network of automata* if we use the standard closed system interpretation.

Open system semantics

Now, we give the open system semantics of an automaton A by its underlying timed labeled transition system. We may assume that A consists of one component, because we give the semantics of the parallel composition later in this section. Our treatment here is rather informal. For more rigorous definitions, the reader is referred to [AD94].

A *timed labeled transition system* (TLTS) consists of a set of states, a set of actions comprising the set of positive real numbers and a transition relation which relates certain states and actions to other states. The TLTS associated to an automaton A is given as follows.

The states (q, v) of the TLTS consist of a location q of A and a *clock valuation* v. The latter is a function that assigns a value in $\mathbb{R}^{\geq 0}$ to each clock variable of the automaton. We require that each state (q, v) of the TLTS underling A meets the location invariant of q. The initial state of the TLTS, if any, is given by the initial location q_0 in the automaton and the clock valuation v_0 that assigns 0 to each clock variable. (Note that there is no initial state if v_0 does not meet the location invariant of q_0.)

The transitions $s \xrightarrow{\ell} s'$ of the TLTS, labeled with a discrete or time passage action, indicate the following: Being in state $s = (q, v)$, that is, the system is in location q and the clocks have the values as described by v, the system can move from s to a new state s', in which the location and clock variables have been updated according to the delay or discrete step taken in the automaton. The *time passage actions* $s \xrightarrow{d} s'$ of the TLTS, where $d \in \mathbb{R}^{>0}$, augment all clocks with d time units and leave the locations unchanged. Such a transition is enabled if augmenting the clocks with d time units is allowed by $\mathrm{Inv}(q)$. The *discrete actions* $s \xrightarrow{a} s'$ change the state as specified by the transition in the network of automata. This means that the guard of the transition involved is met in s, that the invariant of the destination of the transition involved is met in s' and that the clock variables are set according to the transition involved. We label the transition in the TLTS by a special symbol τ if the corresponding transition in the automaton is unlabeled.

Hence, the TLTS has an infinite set of states, actions and (for non–trivial automata) transitions.

Example 7.5.1 The TLTS underlying the system depicted in Figure 7.14 consists of the states (empty, u) and (filled, u') for $u \geq 0$ and $u' < 15$ (where u and u' denote the valuations that assign respectively the values u and u' to the clock x), the initial state $(\mathit{empty}, 0)$ and the transitions

$$\begin{aligned}
&(\mathit{empty}, u) \xrightarrow{\texttt{send?}} (\mathit{filled}, 0), \text{for } u \geq 0,\\
&(\mathit{filled}, u) \xrightarrow{\texttt{receive!}} (\mathit{empty}, u), \text{for } 12 \leq u \leq 15,\\
&(\mathit{empty}, u) \xrightarrow{d} (\mathit{empty}, u + d), \text{for } d > 0,\\
&(\mathit{filled}, u) \xrightarrow{d} (\mathit{filled}, u + d), \text{for } d > 0, u + d < 15.
\end{aligned}$$

When interpreted as a closed system, the TLTS underlying `Buffer` would not have any transitions, because the actions `receive` and `send` cannot synchronize.

Definition 7.5.2

1. A *timed execution* of a TLTS is a possibly infinite sequence s_0, a_1, s_1, $a_2, \ldots$ such that s_0 is the initial state and s_i is a state, a_i a (discrete or delay) action, and $s_i \xrightarrow{a_{i+1}} s_{i+1}$ a transition.
2. A *timed execution* of an automaton is a timed execution of its underlying TLTS – a sequence (q_0, v_0), a_1, (q_1, v_1), a_2, $(q_2, v_2), \ldots$ where q_i and v_i, are a location and clock valuation respectively.
3. A state is *reachable* if there exists an execution passing that state.
4. A *trace* (of either an automaton or a TLTS) arises from an execution by omitting the states and internal actions. The sets of traces of a TLTS or an automaton A are both denoted by $\mathit{traces}(A)$. We write $A \sqsubseteq_{\mathrm{TR}} B$ if $\mathit{traces}(A) \subseteq \mathit{traces}(B)$.

Example 7.5.3 The state $(\mathit{filled}, 13)$ is reachable in the automaton `Buffer`. An execution passing it by is $(\mathit{empty}, 0)$, 18, $(\mathit{empty}, 18)$, `send?`, $(\mathit{filled}, 0)$, 10, $(\mathit{filled}, 10)$, 3, $(\mathit{filled}, 13)$. The states $(\mathit{filled}, 15)$ and $(\mathit{filled}, 16)$ are not reachable in this automaton.

The following theorem expresses the main result by Alur and Dill [AD94], upon which Uppaal and other model checkers based on timed automata are built.

Theorem 7.5.4 *The set of reachable states of an automaton is decidable.*

An important class of automata and TLTS is formed by the deterministic systems. Intuitively, determinism means that, given the current state and the action to be taken, the next state (if any) is uniquely determined.

Definition 7.5.5

1. A TLTS is called *deterministic* if there are no τ labeled transitions and for every state s and every action $a \in \mathit{act}$, there is at most one state t such that $s \xrightarrow{a} t$.

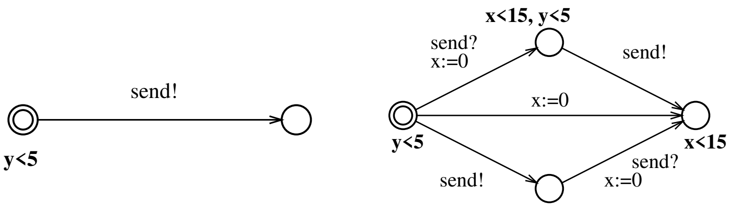


Figure 7.15: `Sender` and `Sender` $\|$ `Buffer` $\setminus$ $\{$`receive`$\}$

2. An automaton is called *deterministic* if there are no unlabeled transitions and $q \xrightarrow{a\ g\ r} q'$ and $q \xrightarrow{a\ g'\ r'} q''$ implies that $q' = q'' \wedge r = r'$ or that $g \wedge \mathrm{Inv}(q)$ and $g' \wedge \mathrm{Inv}(q)$ are disjoint, (i.e. $g \wedge g' \wedge \mathrm{Inv}(q)$ is unsatisfiable).

It is not difficult to prove that if an automaton is deterministic then its underlying TLTS is so.

Parallel composition of automata

The parallel composition operator $\|$ which we consider is basically the composition as in CCS [Mil80]. This means that the automaton $A_1 \parallel A_2$ contains the control locations of both automata. A transition in the parallel composition corresponds to either one of the components taking a transition and the other one remaining in the same location or to both components taking a transition simultaneously, while synchronizing on complementary actions $a?$ and $a!$. Synchronization yields an invisible action in the composition (and hence no other components can synchronism with the same action).

Definition 7.5.6 The *parallel composition* of two automata, A_1 and A_2 is the automaton $A_1 \parallel A_2$ such that

1. The locations of $A_1 \parallel A_2$ are the pairs whose first element is a location of A_1 and its second element one of A_2.
2. The invariant of the location (q_1, q_2) is $\mathrm{Inv}(q_1) \wedge \mathrm{Inv}(q_2)$.
3. The initial location is the pair with the initial location of A_1 and the initial location of A_2.
4. The step $(q_1, q_2) \xrightarrow{a,g,r} (q_1', q_2')$ is a transition in the parallel composition if $q_1 \xrightarrow{a,g,r} q_1'$ is a transition in A_1 and $q_2 = q_2'$, or if $q_2 \xrightarrow{a,g,r} q_2'$ is a transition in A_2 and $q_1 = q_1'$. The step $(q_1, q_2) \xrightarrow{\tau, g_1 \wedge g_2, r_1 \cup r_2} (q_1', q_2')$ is a transition in the parallel composition if $q_1 \xrightarrow{a,g_1,r_1} q_1'$ is a transition in A_1 and $q_2 \xrightarrow{b,g_2,r_2} q_2'$ is a transition in A_2, and a and b are complementary actions.

Example 7.5.7 Figure 7.15 shows the automaton `Sender` and the automaton `Sender` $\|$ (`Buffer` $\setminus$ $\{$`receive`$\}$).

Compositionality results for trace inclusion have been proven in several settings, see e.g. [LV96a]. It also holds for automata we consider.

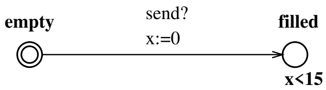


Figure 7.16: Action restriction

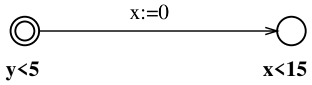


Figure 7.17: $\mathcal{N}(\mathtt{Sender}, \mathtt{Buffer} \setminus \{\mathtt{receive}\})$

Proposition 7.5.8 *Trace inclusion is compositional, that is,*

$$A_1 \sqsubseteq_{\mathrm{TR}} A_2 \implies A_1 \parallel B \sqsubseteq_{\mathrm{TR}} A_2 \parallel B.$$

It is crucial here that the automata we consider do not contain committed locations, urgent channels and shared variables. However, the auxiliary automata that we use to establish trace inclusion within Uppaal do contain committed locations in some cases. This, of course, does not affect the compositionality result. The following proposition is an immediate consequence of the definitions.

Closed systems semantics

Definition 7.5.9 Let C be a set of action names and A be an automaton. Then $A \setminus C$ is the automaton obtained from A by disabling all action with names in C, (i.e. by removing all transitions labeled by an action in $C? \cup C!$.)

Example 7.5.10 The automaton $\mathtt{Buffer} \setminus \{\mathtt{receive}\}$ is shown in Figure 7.16.

Proposition 7.5.11 $\mathrm{Act}_{A\parallel B} = \mathrm{Act}_A \cup \mathrm{Act}_B$ *and* $\mathrm{Act}_{A\setminus C} = \mathrm{Act}_A \setminus (C? \cup C!)$.

Uppaal interprets a network of automata as a closed system. This means that the semantics $\mathcal{N}(A_1, \ldots, A_n)$ Uppaal assigns to a network consisting of components $A_1, \ldots, A_n$ is given by the automaton

$$\mathcal{N}(A_1, \ldots A_n) = (A_1 \parallel \ldots \parallel A_n) \setminus \mathrm{Names}$$

Example 7.5.12 In Figure 7.17 presents the Uppaal semantics

$$\mathcal{N}(\mathtt{Sender}, \mathtt{Buffer} \setminus \{\mathtt{receive}\})$$

of the network consisting of the components $\mathtt{Sender}$ and $\mathtt{Buffer} \setminus \{\mathtt{receive}\}$.

Thus, the network does not have any visible actions, and therefore its traces only contain time passage actions. Since we are interested in describing sets of traces, we cannot do with closed systems only. With Uppaal we can, of course, only verify closed systems. However, the reachability problems of automata can be expressed in terms of reachability properties of closed systems, by simply adding an automaton to the open system that synchronizes with every visible action. See Appendix A, Section A.1.3 for the details.

Verification of trace inclusion

Within automaton–based verification, it is common practice to describe both the implementation and the specification of a system as automata. Then an automaton A is said to be a (correct) *implementation* of another one B if $A \sqsubseteq_{\mathrm{TR}} B$. [6]

Although it is in general undecidable whether $A \sqsubseteq_{\mathrm{TR}} B$, Alur and Dill [AD94] have shown (in a timed automaton setting without location invariants) that deciding whether or not $A \sqsubseteq_{\mathrm{TR}} B$ can be reduced to reachability checking, provided that B is deterministic. The basic step in this reduction is the construction of an automaton which we call B^{err} here. In our setting, B^{err} is constructed by adding a location $error$ to B and transitions $q \xrightarrow{a\ g} error$ for all locations q and action labels a in such a way that this transition is enabled if no other a–transition is enabled from q. Furthermore, an internal transition from q to $error$ with the guard $\neg\mathrm{Inv}(q)$ is added and all location invariants are removed. The basic result that we need in the verification is that, if the visible actions of A are included in those of B, then

$$A \sqsubseteq_{\mathrm{TR}} B \iff error \text{ is not reachable in } \mathcal{N}(A, B^{err}).$$

The reader is referred to the Appendix A, Section A.1.5 for an elaboration of this.

Moreover, if B is not deterministic, we can try to make it so by renaming its actions and apply the method above. We can use a, what is called, *step refinement* f (or a conjectured one) for this relabeling. To put it very briefly, a step refinement is basically a function from the states of A to the states of B that induces a function from the a–transitions of A to the a–transitions of B. Thus, we can give a transition in A and its image in B the same fresh label and remove all sources of nondeterminism in B. This yields automata A^f and B^f such that

$$A^f \sqsubseteq_{\mathrm{TR}} B^f \implies A \sqsubseteq_{\mathrm{TR}} B,$$

(but not conversely). We refer the reader to Appendix A, Section A.1.5 for the details.

7.5.3 The Enhanced Protocol Model

The enhanced protocol model is presented in Figures 7.18 and 7.19 and are explained below. A major difference between the model in Section 7.4 and the model presented in here lies in the way in which communication between the nodes across the wires is handled. In Section 7.4 this is modeled as the transfer of single messages (PN or CN) that are sent only once, and can be overwritten and lost. As we have ecplained before, this abstraction is inappropriate, since in IEEE 1394 communication is done via signals continuously being driven across the wire. These signals persist at the input port of the receiving node, until the sending node changes its output port signal. An important difference between communication via messages

[6] In fact, coarser notions exist, viz. timed trace and timed trace inclusion, which abstract from certain irrelevant timing aspects that are still present in the traces. Timed trace inclusion has a rather technical definition and trace inclusion implies timed trace inclusion. Therefore, we deal with trace inclusion in the remainder rather than with timed trace inclusion.

and via channels is that one can distinguish two subsequent messages with the same contents, but it is not possible to distinguish two subsequent signals that are equal. Besides driving PN and CN signals, the wire can be left undriven (IDLE). Since the enhanced model presented in this section closely follows the draft IEEE 1394a standard, we believe that our model now adequately reflects the root contention protocol as specified in the IEEE standard. However, we can never formally prove this because the specification is partly informal.

We use the constructions and techniques from Section 7.5.1 to verify this model and we show that – tacitly assuming the basic constraints B_1, B_2 and B_3 – the constraints Eq_1 and Eq_2 are both necessary and sufficient for the correctness of the protocol. Recall that the parameters take values in $\mathbb{R}^{\geq 0}$. These constraints were given beforehand in the informal note [LaF97]. However, Section 7.4 and [Nyu97] also give (different) constraints that ensure protocol correctness. Moreover, we consider in each step of our analysis the constraints needed for the correctness of this step.

This section presents the Uppaal automata modeling RCP. We will see that Equation Eq_1 is needed to ensure that these models faithfully represent the protocol. The next section is concerned with the verification of these models. In the sequel, the term "experimental results" is used for results that have been obtained by checking a number of instances with Uppaal, rather than by a rigorous proof.

The Protocol model automata

Figures 7.18 and 7.19 display the Uppaal templates of the `Wire` and `Node` automata of the enhanced model. These template automata are instantiated to a total system ($\texttt{Impl}^{\texttt{B}}$) of two nodes $\texttt{Node}_1$ and $\texttt{Node}_2$, connected by bi–directional communication channels ($\texttt{Wire}_{1,2}$ for messages from $\texttt{Node}_1$ to $\texttt{Node}_2$, and $\texttt{Wire}_{2,1}$ for the opposite directions). We require synchronization between the action that model communication between the nodes and wire, but no synchronization is required for the actions $root$ and $child$. Thus, model of the root contention protocol is given by

$$\texttt{Impl}^{\texttt{B}} \equiv (\texttt{Node}_1 \parallel \texttt{Wire}_{1,2} \parallel \texttt{Wire}_{2,1} \parallel \texttt{Node}_2) \setminus C,$$

where C is the set of actions used to send signals over the wires, that is $C = \{snd\_req_{1,2}$, $snd\_ack_{1,2}$, $snd\_idle_{1,2}$, $snd\_req_{2,1}$, $snd\_ack_{2,1}$, $snd\_idle_{2,1}$, $rec\_req_{1,2}$, $rec\_ack_{1,2}$, $rec\_idle_{1,2}$, $rec\_req_{2,1}$, $rec\_ack_{2,1}$, $rec\_idle_{2,1}\}$.

First of all, the PN message is now called req (parent request), and the CN message ack (acknowledgement). A number of timing parameter constants is defined to include the root contention wait times and the cable propagation delay into the model. The root contention wait times, like rc_fast_min, have been set to the values as specified in Figure 7.5. The actions like snd_ack and rec_req are used to send and receive ack and req messages by the nodes through the wires. The slow/fast differentiation causes the `Node` automaton to be rather symmetric.

Starting in the root contention location, a node picks a random bit (slow or fast). Simultaneously, a timer x is reset, and an $idle$ message is sent to the `Wire`, which models the removal of the PN signal. Independently, but at about the same time, the other contending `Node` also sends an $idle$, possibly followed by a renewed req. Therefore, the receipt of this $idle$ and req message is interleaved with the choice of the random bit and with the sending of the $idle$ message. In this way, the two contending `Node` automata can operate autonomously. The `Wire` templates have been extended, compared to the model in Section 7.4 such that they can now transmit PN (req), CN (ack), and IDLE ($idle$) messages. These messages mark the leading

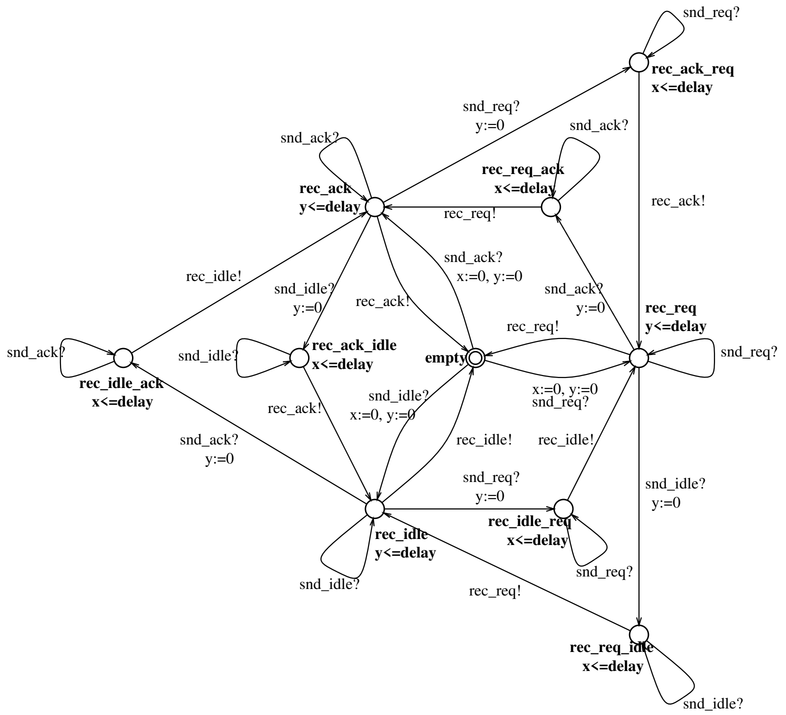


Figure 7.18: The Uppaal Wire automaton template

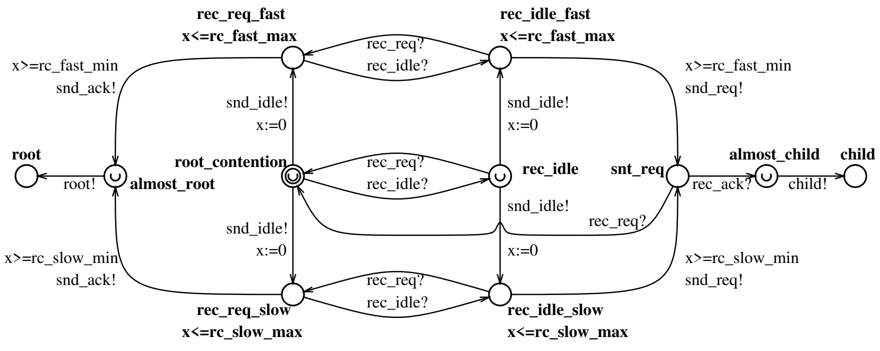


Figure 7.19: The Uppaal Node automaton template

edge of a new signal being driven across the wire. Until a new message arrives, signals continue to be driven across the wire. Furthermore, the wires now comprise a two–place buffer, such that two messages at the same time can travel across a wire. The IEEE standard does not specify how many signals can be in the wire simultaneously. However, the following experimental results shows that two–place buffer is necessary and sufficient to model the communication channels. The necessity result (Experimental result 7.5.13) has been established by checking for reachability of the locations where either of the wires contains two signals. The sufficiency result (Experimental result 7.5.14) has been established by checking that no input to the wire occurs if it already contains two messages. Technically, we constructed an automaton $\mathtt{Wire}^{\mathtt{u\_i}}$ by adding locations $unexp\_input$ (unexpected input) to each of the wires and transitions $q \xrightarrow{a} unexp\_input$ for all the locations $q = rec\_ack\_idle$, rec_ack_req, rec_req_ack, rec_idle_req, rec_idle_ack, rec_req_idle and $a = snd\_idle$, snd_req, snd_ack. Then we checked that the location $unexp\_input$ is unreachable indeed. See Appendix A, Section A.1.4.

Experimental result 7.5.13 *For all parameter values, the wire may contain two signals simultaneously at some point in time, that is, one of the locations rec_ack_idle, rec_ack_req, rec_req_ack, rec_idle_req, rec_idle_ack or*
rec_req_idle is reachable in $\mathtt{Impl}^{\mathtt{B}}$.

Experimental result 7.5.14 *The $unexp\_input$ locations in* $(\mathtt{Node}_1 \parallel \mathtt{Wire}^{\mathtt{u\_i}}_{1,2} \parallel \mathtt{Wire}^{\mathtt{u\_i}}_{2,1} \parallel \mathtt{Node}_2) \setminus C$ *are unreachable if and only if Eq_1 holds.*

The `Node` automaton is not input enabled, which means that it might block input actions (rec_ack, rec_req or rec_idle) in certain locations by being unable to synchronize. Experimental result 7.5.15, however, states that this never happens in the protocol. That is, provided that Eq_1 holds, no other input can occur than the input specified in `Node`. This implies equivalence between $\mathtt{Node}^{\mathtt{u\_i}}$ automaton and the `Node` automaton for parameter values meeting Equation Eq_1. This result has been established by adding a location called $unexp\_input$ to each component and synchronization transitions to this location from all locations in which input (via rec_idle, rec_req and rec_ack) would otherwise be blocked. See Appendix A, Section A.1.4 for a more formal treatment.

Experimental result 7.5.15 *The $unexp\_input$ locations in* $(\mathtt{Node}^{\mathtt{u\_i}}_1 \parallel \mathtt{Wire}^{\mathtt{u\_i}}_{1,2} \parallel \mathtt{Wire}^{\mathtt{u\_i}}_{2,1} \parallel \mathtt{Node}^{\mathtt{u\_i}}_2) \setminus C$ *are unreachable if and only if Eq_1 holds.*

7.5.4 Verification and Analysis

A key correctness property of the root contention protocol is that eventually, exactly one of the processes is elected as root. This property is described by the automaton `Spec` in Figure 7.20. This automaton is exactly the same as the automaton `Spec` from the previous section (Section 7.4), it is only expressed in a different syntax and the (trivial) task partition used in the previously to model fairness has been omitted.

We demonstrate that $\mathtt{Impl}^{\mathtt{B}}$ (the parallel composition of the two `Node` and `Wire` automata) is a correct implementation of `Spec`, provided that $\mathtt{Impl}^{\mathtt{B}}$ meets the timing constraint Equations Eq_1 and Eq_2 from Section 7.2.3.

Following the lines in Section 7.4 we do not prove $\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{Spec}$ at once but introduce three intermediate automata $\mathtt{I}^{\mathtt{B}}_1$, $\mathtt{I}^{\mathtt{B}}_2$, and $\mathtt{I}^{\mathtt{B}}_3$ in our verification. We use Uppaal and the methods

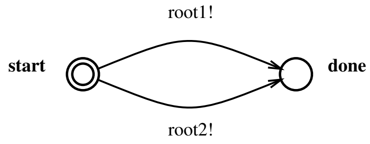


Figure 7.20: The specification automaton of the protocol

described in Section 7.5.2 to derive from numerous instances of the protocol for different parameter values that

$$\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_1^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_2^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_3^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{Spec}$$

if the parameters meet the timing constraints. Furthermore, we argue that $\mathtt{Impl}^{\mathtt{B}}$ is not a correct implementation of $\mathtt{Spec}$ if the parameters do not satisfy the constraints.

Here, $\mathtt{I}_1^{\mathtt{B}}$ is a timed automaton, which abstracts from all message passing in $\mathtt{Impl}^{\mathtt{B}}$ while preserving the timing information of $\mathtt{Impl}^{\mathtt{B}}$. The automaton $\mathtt{I}_2^{\mathtt{B}}$ is obtained from $\mathtt{I}_1^{\mathtt{B}}$ by removing all timing information. In $\mathtt{I}_3^{\mathtt{B}}$ internal choices are further contracted. Notice that the intermediate automata $\mathtt{I}_1^{\mathtt{B}}$, $\mathtt{I}_2^{\mathtt{B}}$ and $\mathtt{I}_3^{\mathtt{B}}$ are obtained from $\mathtt{Impl}^{\mathtt{B}}$ in the same way as $\mathtt{I}_1^{\mathtt{A}}$, $\mathtt{I}_2^{\mathtt{A}}$ and $\mathtt{I}_3^{\mathtt{A}}$ are obtained from $\mathtt{Impl}^{\mathtt{A}}$. In fact, we will see that, when restricted to the reachable states, $\mathtt{I}_2^{\mathtt{B}}$ is basically the automaton $\mathtt{I}_2^{\mathtt{A}}$ where all probabilistic information has been omitted, that $\mathtt{I}_3^{\mathtt{A}}$ is basically $\mathtt{I}_3^{\mathtt{B}}$ where the probabilistic information was left out. but that $\mathtt{I}_1^{\mathtt{B}}$ and $\mathtt{I}_1^{\mathtt{A}}$ differ in their timing behavior. Moreover, remark that, since timing aspects are only present in $\mathtt{Impl}^{\mathtt{B}}$ and $\mathtt{I}_1^{\mathtt{B}}$, the timing constraints only play a role in the first inclusion ($\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_1^{\mathtt{B}}$).

The first intermediate automaton

The intermediate automaton $\mathtt{I}_1^{\mathtt{B}}$ is displayed in Figure 7.21. It is a Uppaal equivalent of the timed I/O automaton model from Section 7.4 restricted to the reachable locations. It abstracts from the communication between the nodes and records the status (start, fast, slow, or done) for each of the two nodes. Also, $\mathtt{I}_1^{\mathtt{B}}$ has a clock x to impose timing constraints on events. The outgoing internal transitions from $start\_start$, $fast\_start$, $start\_fast$, $start\_slow$, and $slow\_start$ model the consecutive random bit selection of the two nodes. For example, $fast\_start$ corresponds to $\mathtt{Node}_1$ having picked the fast random bit, and $\mathtt{Node}_2$ still being in root contention. The internal transitions from $fast\_fast$ and from $slow\_slow$ back to $start\_start$ represent the protocol restart, which is an option if the two random bits are equal. The invariants on clock x cause both nodes to pick a random bit within a time interval $delay$ after the protocol (re–)start. Also, within an interval $[rc\_fast\_min - delay, rc\_fast\_max]$ or $[rc\_slow\_min - delay, rc\_slow\_max]$, depending on the random bit, either a root is selected ($root_1!$ or $root_2!$) or a restart of the protocol occurs.

The method described in Section 7.5.2 allowed us to establish trace inclusion between $\mathtt{Impl}^{\mathtt{B}}$ and $\mathtt{I}_1^{\mathtt{B}}$. Figure 7.22 describes how unlabeled transitions in $\mathtt{I}_1^{\mathtt{B}}$ and $\mathtt{Impl}^{\mathtt{B}}$ are relabeled, yielding $\mathtt{Impl}^{\mathtt{B}^r}$ and $\mathtt{I}_1^{\mathtt{B}^r}$. (See also Section 7.5.2 and Appendix A, Section A.1.5.) This relabeling of transitions has been constructed from the step refinement from $\mathtt{Impl}^{\mathtt{B}}$ to $\mathtt{I}_1^{\mathtt{B}}$ given in Section 7.4 The transitions relabeled with $retry$ synchronize with an auxiliary automaton

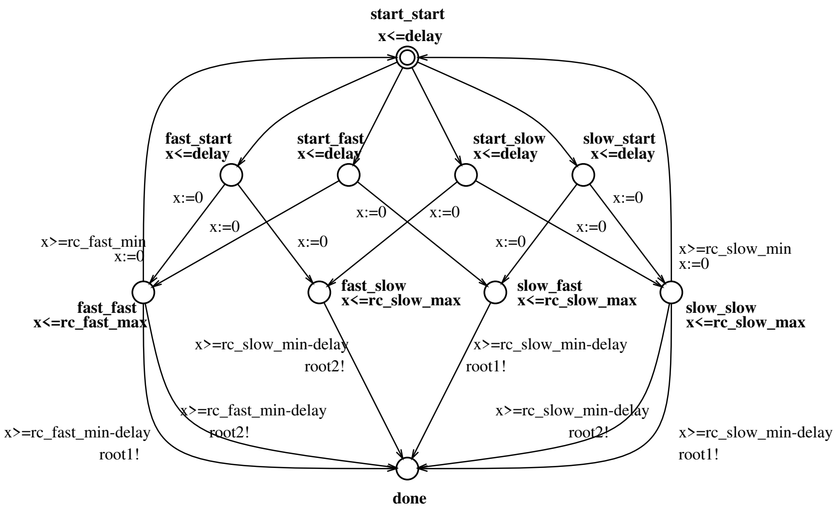


Figure 7.21: The Uppaal $\mathtt{I}_1^{\mathtt{B}}$ automaton of the root contention protocol

called EchoRetry, which takes this action as soon as root contention re–occurs, i.e. as soon as both $\mathtt{Node_1}$ and $\mathtt{Node_2}$ have taken their *snd_req*! transitions to the *snt_req* location. This requires the automata $\mathtt{Node_i}$, $\mathtt{Wire_{i,j}}$ and EchoRetry all to synchronize on the action $snd\_req_{i,j}$ and $\mathtt{Node_i}$, $\mathtt{Wire_{i,j}}$ and $\mathtt{I}_1^{\mathtt{B}}$ on $snd\_idle_{i\,j}$. We encoded multiway synchronization in Uppaal as described in Section A.1.2.

Manual parameter analysis shows that the error location is unreachable in $\mathtt{Impl}^{\mathtt{B}^r} \parallel \mathtt{I}_1^{\mathtt{B}^r}$, if the constraint *Eq1* and *Eq2* hold. Now, Experimental result 7.5.16 is a direct consequence of Lemma A.1.14.

Experimental result 7.5.16 *If the timing parameters in* $\mathtt{Impl}^{\mathtt{B}}$ *satisfy Equations Eq1 and Eq2, then* $\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_1^{\mathtt{B}}$.

In order to show the necessity of Equations 2, we need a liveness argument. The key liveness property is that eventually a leader is elected with probability one. Therefore, it is essential that root contention is resolved within the same pass (i.e. without renewed root contention) if both nodes pick different random bits. This is guaranteed by *Eq2*, c.f. Section 7.2.3. Since the probability to pick different random bits is strictly greater than zero in each pass, the nodes will eventually pick different bits, and thus elect a root, with probability one.

Experimental result 7.5.17 *Assume that the two nodes pick different random bits. Then the root contention protocol is resolved within one pass if and only if Equation 2 is satisfied.*

$\mathtt{I}_1^{\mathtt{B}}$	Impl
$start\_start \to fast\_start$	$root\_contention_1 \xrightarrow{snd\_idle12}_{\mathtt{Node}_1} rec\_req\_fast_1$
$start\_start \to start\_fast$	$root\_contention_2 \xrightarrow{snd\_idle21}_{\mathtt{Node}_2} rec\_req\_fast_2$
$start\_start \to start\_slow$	$root\_contention_2 \xrightarrow{snd\_idle21}_{\mathtt{Node}_2} rec\_req\_slow_2$
$start\_start \to slow\_start$	$root\_contention_1 \xrightarrow{snd\_idle12}_{\mathtt{Node}_1} rec\_req\_slow_1$
$start\_fast \to slow\_fast$	$root\_contention_1 \xrightarrow{snd\_idle12}_{\mathtt{Node}_2} rec\_req\_slow_1$
$slow\_start \to slow\_fast$	$root\_contention_2 \xrightarrow{snd\_idle21}_{\mathtt{Node}_2} rec\_req\_fast_2$
$start\_slow \to slow\_slow$	$root\_contention_1 \xrightarrow{snd\_idle12}_{\mathtt{Node}_1} rec\_req\_slow_1$
$fast\_start \to fast\_fast$	$root\_contention_2 \xrightarrow{snd\_idle21}_{\mathtt{Node}_2} rec\_req\_fast_2$
$start\_fast \to fast\_fast$	$root\_contention_1 \xrightarrow{snd\_idle12}_{\mathtt{Node}_1} rec\_req\_fast_1$
$start\_slow \to slow\_slow$	$root\_contention_1 \xrightarrow{snd\_idle12}_{\mathtt{Node}_1} rec\_req\_slow_1$
$fast\_start \to fast\_slow$	$root\_contention_2 \xrightarrow{snd\_idle21}_{\mathtt{Node}_2} rec\_req\_slow_2$
$fast\_fast \to start\_start$	$one\_req \xrightarrow{retry}_{\mathtt{EchoRetry}} start$
$slow\_slow \to start\_start$	$one\_req \xrightarrow{retry}_{\mathtt{EchoRetry}} start$

Figure 7.22: Relabeling $\mathtt{I}_1^{\mathtt{B}}$ and $\mathtt{Impl}^{\mathtt{B}}$.

The second intermediate automaton

The intermediate automaton $\mathtt{I}_2^{\mathtt{B}}$ is identical to $\mathtt{I}_1^{\mathtt{B}}$, except that all timing information has been removed. Thus, $\mathtt{I}_2^{\mathtt{B}}$ is the exactly same as the automaton $\mathtt{I}_2^{\mathtt{A}}$ from the previous section, if all probabilistic information, the task partition and the unreachable states are omitted from the latter automaton. Since weakening the guards and invariants in an automaton yields an automaton with more traces, we get Proposition 7.5.18, as expected.

Proposition 7.5.18 $\mathtt{I}_1^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_2^{\mathtt{B}}$.

The third intermediate automaton

Figure 7.23 shows intermediate automaton $\mathtt{I}_3^{\mathtt{B}}$, in which internal choices have been further contracted. Selection of the two random bits is no longer represented via separate, subsequent transitions, but done at once via a single transition. Thus, $\mathtt{I}_3^{\mathtt{B}}$ is the exactly same as $\mathtt{I}_3^{\mathtt{A}}$, if we delete all probabilistic information, the task partition and the unreachable states in the latter automaton.

Since neither $\mathtt{I}_2^{\mathtt{B}}$ nor $\mathtt{I}_3^{\mathtt{B}}$ contains timing information, trace inclusion can be checked with standard methods, see [KS90]. Since we are interested in the applicability of the relabeling method, we use this one for establishing $\mathtt{I}_2^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_3^{\mathtt{B}}$. Again, we added labels to certain unlabeled transitions in $\mathtt{I}_2^{\mathtt{B}}$ and $\mathtt{I}_3^{\mathtt{B}}$, to obtain $\mathtt{I}_2^{\mathtt{B}f}$ from $\mathtt{I}_2^{\mathtt{B}}$ and (the deterministic automaton) $\mathtt{I}_3^{\mathtt{B}f}$ from $\mathtt{I}_3^{\mathtt{B}}$. Figure 7.24 lists the corresponding transitions in $\mathtt{I}_3^{\mathtt{B}}$ and $\mathtt{I}_2^{\mathtt{B}}$ that should get the same (fresh) labels after relabeling. Transitions not mentioned the table keep the same label. In particular, the transitions in $\mathtt{I}_2^{\mathtt{B}}$ leaving from $start\_start$ remain unlabeled.

It has been established by Uppaal that $\mathtt{I}_2^{\mathtt{B}f} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_3^{\mathtt{B}f}$. Now, Proposition 7.5.19 is an immediate corollary of Lemma A.1.14.

Proposition 7.5.19 $\mathtt{I}_2^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_3^{\mathtt{B}}$.

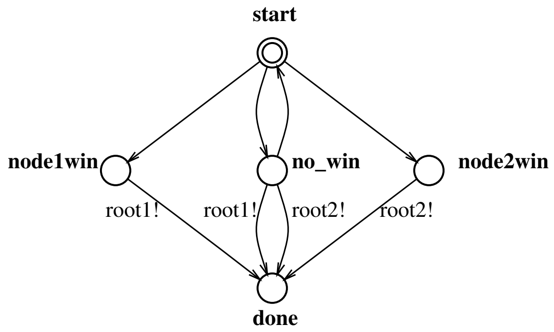


Figure 7.23: The Uppaal $\mathtt{I}_3^{\mathtt{B}}$ automaton of the root contention protocol

$\mathtt{I}_3^{\mathtt{B}}$	$\mathtt{I}_2^{\mathtt{B}}$
$start \xrightarrow{win_2?} node2win$	$start\_fast \xrightarrow{win_1!} slow\_fast$
$start \xrightarrow{win_1?} node1win$	$slow\_start \xrightarrow{win_2!} slow\_fast$
$start \xrightarrow{win_1?} node1win$	$fast\_start \xrightarrow{win_2!} fast\_slow$
$start \xrightarrow{win_2?} node2win$	$start\_slow \xrightarrow{win_1!} slow\_fast$
$start \xrightarrow{no\_win?} no\_win$	$start\_slow \xrightarrow{no\_win!} slow\_slow$
$start \xrightarrow{no\_win?} no\_win$	$start\_slow \xrightarrow{no\_win!} slow\_slow$
$start \xrightarrow{no\_win?} no\_win$	$fast\_start \xrightarrow{no\_win!} slow\_slow$
$start \xrightarrow{no\_win?} no\_win$	$start\_fast \xrightarrow{no\_win!} fast\_fast$

Figure 7.24: Corresponding transitions getting the same label

Since the specification automaton Spec is deterministic, we only need to check for reachability of the *error* location in the automaton $\mathtt{Spec}^{err}$ to obtain Proposition 7.5.20. (As in the previous case ($\mathtt{I}_2^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_3^{\mathtt{B}}$) trace inclusion and timed trace inclusion are the same. But now, because Spec is deterministic, the method we use to establish timed trace inclusion this is exactly the usual method for establishing trace inclusion.)

Proposition 7.5.20 $\mathtt{I}_3^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{Spec}$.

By transitivity of $\sqsubseteq_{\mathrm{TR}}$ we get that the Equations 1 and 2 are sufficient.

Experimental result 7.5.21 *If Equations 1 and 2 are met by* $\mathtt{Impl}^{\mathtt{B}}$*, then* $\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{Spec}$.

Combining the Results 7.5.14, 7.5.15, 7.5.17 and 7.5.21 yields the final conclusion.

Experimental result 7.5.22 *The root contention protocol is correct if and only if the timing parameters satisfy Equations Eq_1 and Eq_2.*

7.5.5 Concluding Remarks

We analyzed a large number of protocol instances with Uppaal. From these experiments, we derived that the constraints which are necessary and sufficient for correct protocol operation are exactly those from [LaF97]. Although these experiments do not ensure correctness, we are quite convinced that the constraints we derived are exactly those required.

Some minor points of incompleteness have been found: The IEEE specification only specifies the propagation delay of signals but not the delay needed to process incoming and outgoing signals. Moreover, the IEEE standard only provides specific values for the timing parameters and not the general parameter constraints, although these give some useful insight in the correctness of the protocol and in restrictions on future applications. We also found some small errors in the informal notes [Nyu97, LaF97].

The fact that the modeling and verification took us a relatively short time illustrates once again that model checkers can be used effectively in the design and evaluation of industrial protocols. Especially the iterative modeling via trial and error is valuable when it comes to understanding the properties of a model. Our case study has added that this also holds in the presence of parameters: once an appropriate model and conjectured timing constraints have been obtained with Uppaal, rigorous parameter analysis could be tackled with another tool or method.

In our case, the very recent work [HRSV01] has given the full evidence of several experimental results in this section. By feeding Eq_1 and Eq_2 together with the basic assumptions B_1, B_2 and B_3 to a prototype parametric extension of Uppaal, the Experimental result 7.5.16 has been established. Due to lack of memory, the necessity of the constraints could only be established partially by the tool.

In our experience, using the current Uppaal2k implementation to establish trace inclusion suffers from several disadvantages. The practical modeling and verification is, as pointed out above, not a problem. However, the proof that the properties we verified indeed established trace inclusion, involved several technicalities. Firstly, due to its closed world interpretation, timed languages cannot be described in Uppaal directly. It is however no conceptual problem to extend Uppaal such that this would be possible. Secondly, the check for language inclusion ($A \sqsubseteq_{\mathrm{TR}} B$, B deterministic) is not implemented in Uppaal. However, we are not aware of any other freely available timed model checker which can do this. This might be remarkable since verification often involves checking language inclusion and it is already known for some time [AD94] how to reduce language inclusion to a reachability problem.

Thirdly, the fact that Uppaal does not support multisynchronization enforces the need of committed locations and this makes the underlying theory more complicated. Relatively small adaptations of Uppaal would overcome these problems and make the verification of trace inclusion a lot easier.

7.6 Third approach: parametric model checking of the Root Contention Protocol

The last verification in this chapter reports on the verification of RCP using the prototype parametric extension of Uppaal from Chapter 6. We analyze the models $\mathtt{Impl}^{\mathtt{A}}$ and $\mathtt{Impl}^{\mathtt{B}}$ from Sections 7.4 and 7.5, where we deal with all five parameters in the system. Unlike the verifications in those sections, we now express the correctness of the protocol by a safety property in Uppaal's logic and find the constraints needed and sufficient to satisfy this property. In addition, we model check the inclusion $\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_1^{\mathtt{B}}$ parametrically. The purpose of the research presented in this section is not only to prove the correctness of RCP but also to investigate the performance and the usability of the new prototype extension of Uppaal.

When verifying the properties mentioned, we profit from the fact that the parameter constraints have already been found in the preceding sections. So, rather than having the tool generating the constraints, which is a time and memory consuming task, we establish the sufficiency of the constraints by providing these as initial constraints to the tool and show that they imply the property under investigation. Necessity is established by showing that the property does not hold for the negated constraints. As in Chapter 6, significant speed ups are gained by eliminating parameters with the techniques developed in Section 6.4.

All experiments were performed on a 366 MHz Celeron, except the check for $\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_1^{\mathtt{B}}$, which was performed in a 333 MHz SPARC Ultra Enterprise. The reader is referred to Fig. 7.25 for an overview of results.

7.6.1 Protocol Specification

A crucial safety property for the system is that at most one of the nodes becomes root and at most one of the nodes becomes child. This is expressed by the invariance property Φ below. Note that Φ is similar to the specification automaton `Spec` from the previous sections.

$$\Phi \equiv \forall\Box\ .\ (\neg(\mathrm{Node}_1.\mathrm{root} \wedge \mathrm{Node}_2.\mathrm{root}) \wedge \neg(\mathrm{Node}_1.\mathrm{child} \wedge \mathrm{Node}_2.\mathrm{child}))$$

Of course, the protocol is also required to eventually elect a leader, but here, we will restrict our attention to the safety property.

As we saw in the previous two verifications, there are three parameter equations that play a role in the protocol correctness.

$$delay < rc\_fast\_min. \qquad (Eq_0)$$
$$2 \cdot delay < rc\_fast\_min. \qquad (Eq_1)$$
$$2 \cdot delay < rc\_slow\_min - rc\_fast\_max. \qquad (Eq_2)$$

As before, we assume a basic constraint C_0, given by

$$C_0 \equiv rc\_fast\_min \leq rc\_fast\_max \leq rc\_slow\_min \leq rc\_slow\_max.$$

Note that this is exactly the initial constraint from Section 7.4, but slightly more restrictive than in Section 7.5. Whenever a property is checked with the prototype parametric model checker, C_0 is given as an initial constraint to the tool.

7.6.2 Verification and analysis

Safety for `Impl`$^{\mathtt{A}}$

The manual verification reported on in Section 7.4 implies that the safety property Φ holds for the model $\mathtt{Impl}^{\mathtt{A}}$ if the parameters meet the Equation Eq_0. This has been re–established mechanically with the prototype extension of Uppaal. For this, we provided the equation Eq_0 as an initial constraint, together with the basic constraint C_0, to the prototype tool and then we checked that Φ holds. (Uppaal models of $\mathtt{Impl}^{\mathtt{A}}$ were available from the research reported on in Section 7.4.) Thus, we have the following result.

Proposition 7.6.1 *If* $v \models C_0 \wedge Eq_0$, *then* $[\![\mathcal{A}]\!]_v \models \Phi$.

Moreover, Remark 7.4.14 on page 199, suggests that the Eq_0 is probably not needed for the safety property to hold and that Eq_2 is needed for liveness only, not for safety. This raises the question whether Φ holds for all parameter values. As Φ is an invariance property and this question is just an instance of the universality problem. Since $\mathtt{Impl}^{\mathtt{A}}$ is an L/U automaton ($L = \{rc\_fast\_min, rc\_slow\_min\}, U = \{rc\_fast\_max, rc\_slow\_max, delay\}$), universality problems can be reduced to non–parametric model checking (see Corollary 6.4.6 and the text just above it). This is done by substituting 0 for each parameter in L and ∞ for all parameters in U. Standard Uppaal established the following result.

Proposition 7.6.2 $\mathcal{A}[(0,\infty)] \models \Phi$.

Now Proposition 6.4.3 yields the desired result.

Corollary 7.6.3 *For all parameter valuations* v, $[\![\mathcal{A}]\!]_v \models \Phi$.

Safety for $\mathtt{Impl}^{\mathtt{B}}$

We check the property Φ for $\mathtt{Impl}^{\mathtt{B}}$. From the investigation in Section 7.5, it follows that Φ is satisfied provided that the parameters meet the equations Eq_1 and Eq_2. Several experiments with the prototype for different initial constraints yielded that the property is already satisfied for Eq_0. Recall, however, that Eq_2 is needed for liveness and that Eq_1 guarantees that the two–place buffers are an appropriate model of the communication channel and also that unexpected input signals do not to occur.

We established the following with our prototype model checker.

Proposition 7.6.4 *If* $v \models C_0 \wedge Eq_0$, *then* $[\![\mathtt{Impl}^{\mathtt{B}}]\!]_v \models \Phi$.

Figure 7.25 shows three performance measures for this result. The first numbers, 1.6 minutes and 36 MB, refer to the direct verification of the Proposition 7.6.4 above for the model $\mathtt{Impl}^{\mathtt{B}}$.

The same result can be checked more efficiently (11 seconds, 13 MB) by substituting ∞ for rc_fast_max and for rc_slow_max and substituting rc_fast_min for rc_slow_min. The new model, together with the initial constraint $C_0 \wedge Eq_0$, satisfies the invariant property; As $\mathtt{Impl}^{\mathtt{B}}$ is an L/U automaton, Proposition 6.4.3 now implies 7.6.4.

Further performance improvements can be made by reducing the verification of the property to nonparametric model checking. In order to do so, we consider the model $\mathtt{Impl}^{\mathtt{B}^<}$. In this model, we substitute $rc\_fast\_max = rc\_slow\_max = \infty$ and $delay = rc\_fast\_min = rc\_fast\_max$. The model we thus obtain has no constants and only one parameter. This means we can check any property for this model by substituting $1 = \mathtt{d} = rc\_fast\_min = rc\_fast\_max$. The model is now completely parametric and satisfies Φ. Application of the Propositions 6.4.8, 6.4.12 and6.4.3 imply Proposition 7.6.4.

7.6.3 $\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}^{\mathtt{B}}_1$

Experimental result 7.5.16 from Section 7.5 states that $\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}^{\mathtt{B}}_1$, provided that the timing parameters in $\mathtt{Impl}^{\mathtt{B}}$ satisfy Eq_1 and Eq_2. This was established experimentally by checking the statement for a large number of protocol instances. Here, the inclusion $\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}^{\mathtt{B}}_1$

model	*property*	*initial constr*	*reduced?*	*Uppaal*	*time*	*memory*
$\mathtt{Impl}^{\mathtt{A}}$	Φ	$C_0 \wedge Eq_0$	no	par	2.9 h	185 Mb
$\mathtt{Impl}^{\mathtt{A}}$	Φ	—	yes	std	1 s	800 Kb
$\mathtt{Impl}^{\mathtt{B}}$	Φ	C_0	no	par	1.6 m	36 Mb
$\mathtt{Impl}^{\mathtt{B}}$	Φ	$C_0 \wedge Eq_0$	partly	par	11 s	13 Mb
$\mathtt{Impl}^{\mathtt{B}}$	Φ	—	yes	std	1 s	800 Kb
$\mathtt{Impl}^{\mathtt{B}}$	$\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_1^{\mathtt{B}}$	$C_0 \wedge Eq_0 \wedge Eq_1$	no	par	2.6 h	308 Mb

Figure 7.25: Experimental results for the root contention protocol

was reduced to the nonreachability of error states added to $\mathtt{I}_1^{\mathtt{B}}$. The prototype model checker allows to verify this result formally. We take the same models as before. So in particular, we use the same reduction to nonreachability of error states.

Proposition 7.6.5 *If* $v \models C_0 \wedge Eq_1 \wedge Eq_2$*, then* $[\![\mathtt{Impl}^{\mathtt{B}}]\!]_v \sqsubseteq_{\mathrm{TR}} [\![\mathtt{I}_1^{\mathtt{B}}]\!]_v$.

Moreover, we established that the error states mentioned above are reachable in the following cases: if the parameters obey $C_0 \wedge (2 \cdot delay = rc\_fast\_min \wedge Eq_2)$ or if they satisfy $C_0 \wedge Eq_1 \wedge 2 \cdot delay = rc\_slow\_min - rc\_fast\_max$. Recall that the constructions presented in Section 7.5.2 only yield sufficient conditions for trace inclusion. This means that even though error states are reachable, we might still have $\mathtt{Impl}^{\mathtt{B}} \sqsubseteq_{\mathrm{TR}} \mathtt{I}_1^{\mathtt{B}}$. Thus, a necessity result for the equations is not given here.

Moreover, $\mathtt{I}_1^{\mathtt{B}}$ falls outside the class of L/U automata, and, therefore, we cannot apply Proposition 6.4.12 to derive a more general result for the equations to be necessary for the unreachability of error states. We conjecture that a more general theory of parametric automata, where parameters are partitioned into lower bound, upper bound and mixed parameters, would be capable of doing so.

7.7 Conclusions

When looking back on the three verification exertions in this chapter, we notice the following points.

Although RCP is not a very complex protocol in itself, many different aspects play a role in its correctness. We tackled probabilistic, real–time, fair and parametric behavior, using three different methods. To keep the analysis process manageable, we separated concerns as much as possible. In doing so, the technique of stepwise abstraction via intermediate automata — not completely unexpectedly — turned out to be very advantageous.

We saw that parametric aspects play a role already in the modeling of the protocol. For some parameter values, the model reflects the IEEE 1394 standard appropriately, for other it does not: the one–place buffers in model $\mathtt{Impl}^{\mathtt{A}}$ are appropriate if Eq_0 holds, the two–place buffers if $\mathtt{Impl}^{\mathtt{B}}$ are so if Eq_1 holds. Moreover, Eq_1 ensures that no other inputs occur than those modeled in $\mathtt{Impl}^{\mathtt{B}}$. The automaton $\mathtt{Impl}^{\mathtt{A}}$ is input–enabled, so all inputs possibly occurring are modeled for sure. Proving that no unexpected inputs occur, as we did for $\mathtt{Impl}^{\mathtt{B}}$, is in fact very similar to proving invariants, which we did for $\mathtt{Impl}^{\mathtt{A}}$. Thus, some of the protocol invariants are encoded in $\mathtt{Impl}^{\mathtt{B}}$, whereas they are properties of $\mathtt{Impl}^{\mathtt{A}}$. The latter may be more elegant, but the input enabled variant of $\mathtt{Impl}^{\mathtt{B}}$ requires many extra transitions, whereas $\texttt{Impl}^{\texttt{B}}$ is a small and easy to understand automaton. In general, this problem would be overcome by adding extra syntactic construct to Uppaal. In the particular case of $\texttt{Impl}^{\texttt{B}}$, the use of variables instead of locations — which can already been done with the current Uppaal version — would already help.

The use of Uppaal to verify trace inclusion was a nice scientific exercise. It showed that trace–based verification with Uppaal is feasible, but also that in order to do it efficiently, an extension of Uppaal with several syntactic constructs would be necessary. This was of course to be expected, because Uppaal is not designed for trace based verification. The strong point of the Uppaal verification was that we could analyze the timing constraints in a quick manner.

As a matter of fact, the experimental verifications with Uppaal were a direct inspiration for the class of lower bound/upper bound automata in Chapter 6: while model checking the protocol for parameter instances, it became clear we could derive firm conclusions for the parametric version of the protocol.

The third verification, where we expressed the protocol specification directly in Uppaal's logic, was more straightforward than the automaton based approach. However, this approach does not support separations of concerns.

Another important remark is that the automata $\texttt{Impl}^{\texttt{A}}$ and $\texttt{Spec}$ are not bisimilar; there is only a one–way simulation from $\texttt{Impl}^{\texttt{A}}$ to $\texttt{Spec}$. This is because $\texttt{Spec}$ can elect process i as a leader with probability one, whereas $\texttt{Impl}^{\texttt{B}}$ can do so with probability $\frac{3}{4}$ at most. The deprobabilized version $\texttt{Impl}^{\texttt{A}^-}$ is, however, bisimilar to $\texttt{Spec}$, provided that the timing actions are hidden. The case for $\texttt{Impl}^{\texttt{B}}$, which is entirely non–probabilistic, is similar. It is bisimilar to $\texttt{Spec}$ if we hide timing actions. If we add probabilities, then there is a simulation from $\texttt{Impl}^{\texttt{B}}$ to $\texttt{Spec}$, but no bisimulation.

This shows that, although bisimulations are useful in many case, they are not sufficient for the analysis of probabilistic algorithms.

Acknowledgements We thank Judi Romijn for her explanation of some subtle points in IEEE 1394, her constructive criticism on early versions of our I/O automata model and other helpful remarks. We also thank the anonymous referees of [SS01] for a remark that has led to a clearer distinction between networks and automata.

APPENDIX A

A.1 Constructions on Uppaal Automata

This appendix treats several constructions on Uppaal automata used in the verification described in Section 7.5.

Organization of the section

We start with some notational conventions. We explain in Section A.1.2 how multi–way synchronization can be encoded in Uppaal. Then Section A.1.3 describes how reachability properties of open systems in Uppaal (automata) can be reduced to closed systems (networks). Section A.1.4 deals with input–enabling in Uppaal. Finally, Section A.1.5 explains how trace inclusion between two automata can be verified in Uppaal by reducing it to non–reachability of certain error states.

A.1.1 Notational Conventions

Besides the conventions adopted in Section 7.5.1, we find it convenient to use general boolean expressions in guards and invariants, whereas Uppaal only allows conjunctions in these. Therefore, we use $q \xrightarrow{a,g_1 \vee g_2,r} q'$ as an abbreviation for the two transitions $q \xrightarrow{a,g_1,r} q'$ and $q \xrightarrow{a,g_2,r} q'$. The guard $g \implies g'$ stands as an abbreviation for $\neg g \vee g'$. The negation $\neg g$ is an abbreviation for the guard obtained by replacing every $>$ in g by $\leq$, every $\geq$ by $<$, $\geq$ by $<$, $\leq$ by $<$ and every $\wedge$ by $\vee$ and $\vee$ by $\wedge$.

A.1.2 Encoding Multi-way Synchronization in Uppaal

As explained before, Uppaal only provides binary synchronization. We use committed locations and renaming to enforce synchronization between more than two action labels. If we want to have an $a!$-action in A_0 synchronize with n $a?$-actions in $A_1 \ldots A_n$ ($n > 1$), then the idea is as follows. Relabel a in A_i into a fresh label a_i, for all $i \geq 0$. Whenever A_0 performs an $a_0!$ action, an auxiliary automaton 'catches' it and 'distributes' it over the other automata. More precisely, the auxiliary automaton synchronizes on $a_0!$ and enforces – sequentially but without delay nor interruption – synchronization with $a_1? \ldots a_n?$ in $A_1 \ldots A_n$ respectively.

Example A.1.1 Consider the automata in Figure A.1. Being in their initial locations, either A_2 or A_3 takes an $a?$-action to synchronize with the $a!$-action in A_1. Synchronization of the three automata on $a!$, $a?$ and $a?$, can be mimicked by introducing an auxiliary automaton and renaming of actions, as in Figure A.2. Remark that it is also possible to integrate the auxiliary automaton within one of the other automata.

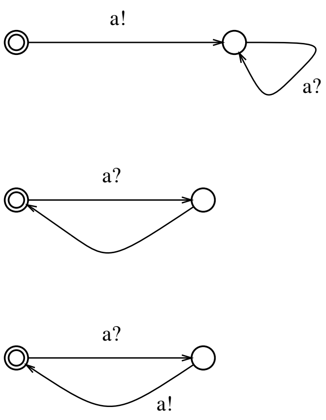


Figure A.1: Three automata: A_1, A_2, A_3

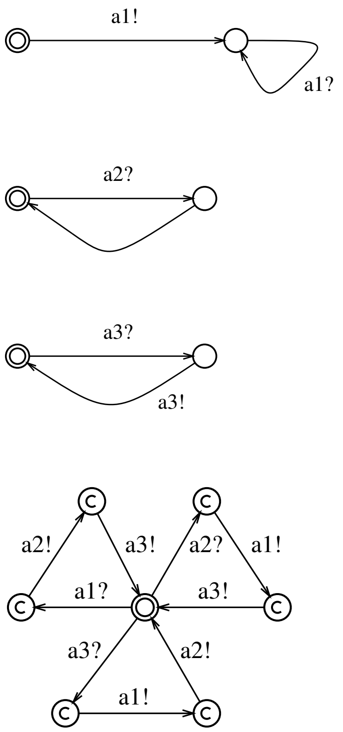


Figure A.2: Encoding multisynchronization with binary synchronization

A.1.3 Reducing Reachability Properties of Automata to Reachability Properties of Networks

This section describes how reachability properties of open systems (automata) can be reduced to closed systems (networks), which can then be verified by Uppaal. The basic idea is simple: we add an auxiliary automaton that synchronizes with every visible action in a given automaton.

Definition A.1.2 Let C be a set of action names. Define the automaton Snc_C as the automaton with one location q_C and transitions $q_C \xrightarrow{a} q_C$ for every a in $C? \cup C!$. For an automaton A, we define $Snc_A = Snc_C$, where C are the action names occurring in A. Moreover, we write q_A for q_C.

The following result allows us to check the reachability properties of an automaton A via the reachability properties the network $\mathcal{N}(A \parallel Snc_A)$, which can be checked by Uppaal. Its proof is straightforward.

Proposition A.1.3 *Let q be a location and v a clock valuation of A. Then (q, v) is reachable in A if and only if $((q, q_A), v)$ is reachable in $(A \parallel Snc_A) \setminus \mathrm{Names}$.*

If A is given in terms of restriction and parallel composition over the components A_1, $A_2, \ldots, A_n$, then we like to carry out the construction of Snc_A on A's components, rather than first computing A explicitly and then performing the construction.

In our verification, A has the form $(A_1 \parallel A_2) \setminus C$. We have checked its reachability properties via the network $\mathcal{N}(A_1, A_2, Snc_{C'})$. This is justified by the following argument, where C is a set of action names, $C' = \mathrm{Names} \setminus C$, $s = (q, v)$ is a state of A and $\overline{s} = ((q, q_{C'}), v)$.

$$
\begin{aligned}
&\overline{s} \text{ reachable in } \mathcal{N}(A_1, A_2, Snc_{C'}) \iff \\
&\overline{s} \text{ reachable in } (A_1 \parallel A_2 \parallel Snc_{C'}) \setminus \mathrm{Names} \iff \\
&\overline{s} \text{ reachable in } (A_1 \parallel A_2 \parallel Snc_{C'}) \setminus C \cup C' \iff \\
&\overline{s} \text{ reachable in } ((A_1 \parallel A_2) \setminus C) \parallel Snc_{C'}) \setminus C' \iff \\
&s \text{ reachable in } (A_1 \parallel A_2) \setminus C.
\end{aligned}
$$

A.1.4 Input Enabling in Uppaal

The *input actions* of an automaton A are the actions of the form $a?$. We call A *input enabled* if synchronization on input actions is always (in any state of the system) possible. More precisely, assume that the $a?$-transitions in A leaving from a location q are given by

$$q \xrightarrow{a?, g_1, r_1} q_1 \ldots, q \xrightarrow{a?, g_n, r_n} q_n.$$

Then A is input enabled iff the expression $\mathrm{Inv}(q) \implies \bigvee_{i=1}^{n}(g_i \wedge \mathrm{Inv}(q_i)[r_i])$ is equivalent to true. Here we use the convention that $\bigvee_{i=1}^{0} \ldots$ yields false. Moreover, $I[r]$ denotes the invariant that is obtained from I by replacing each clock variable in the reset set r by 0. We state that an automaton is input enabled if and only if its underlying TLTS is so, using the standard notion of input enabledness for TLTSs.

A non-input enabled automaton may block input, by being unable to synchronize on it. This is often considered as a bad property, since a component is usually not able to prevent the environment from providing inputs and this might indicate a modeling error. Therefore, it is relevant to know whether blocking of inputs can actually occur in a network of automata. So, one would like to find out whether or not the situation can occur that one component could provide an input to another one, while the latter automaton is not able to synchronize on it. To check this property, we make every component A input enabled by directing every input that would otherwise be blocked to a fresh location, called $unexp\_input_A$. Then we check for reachability of this location. More precisely, we construct the automaton $A^{\mathtt{u\text{-}i}}$ from A as follows. If the outgoing $a?$-transitions leaving from q are as above, then we add the transition $q \xrightarrow{a?, g} unexp\_input_A$, where g is equivalent to $\neg(\mathrm{Inv}(q) \implies \bigvee_{i=1}^{n}(g_i \vee \mathrm{Inv}(q_i)[r_i]))$. (In particular, if q does not have any outgoing $a?$-transitions, then we add $q \xrightarrow{a?} unexp\_input_A$.) We also add the transition $unexp\_input_A \xrightarrow{a?} unexp\_input_A$ for every input action $a?$ of A.

Proposition A.1.4 *The automaton $A^{\mathtt{u\text{-}i}}$ is input enabled.*

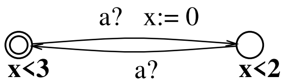


Figure A.3: An input enabled automaton

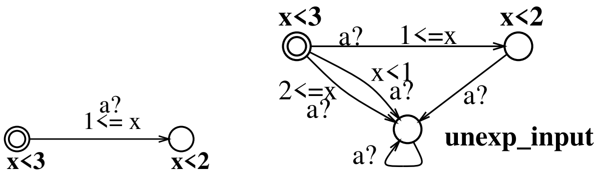


Figure A.4: Constructing an input enabled automaton

Example A.1.5 The automaton in Figure A.3 is input enabled. Notice that $x < 3 \implies ((x < 2)[x := 0])$ yields true. The construction to make an automaton input enabled is shown in Figure A.4. Notice that $\neg(x < 3 \implies 1 \leq x \wedge x < 2)$ is equivalent to $x < 1 \vee x \geq 2$.

By checking for reachability of the location $unexp\_input$, we can establish whether A and $A^{\mathtt{u\_i}}$ are semantically equivalent, i.e. whether their underlying TLTSs are exactly the same when restricted to their reachable states.

Proposition A.1.6 *The network $\mathcal{N}(A_1, \ldots, A_n)$ is semantically equivalent to the network $\mathcal{N}(A_1^{\mathtt{u\_i}}, \ldots, A_n^{\mathtt{u\_i}})$ if and only if none of the locations $unexp\_input_{A_i}$ is reachable in the network $\mathcal{N}(A_1^{\mathtt{u\_i}}, \ldots, A_n^{\mathtt{u\_i}})$.*

A.1.5 Verification of Trace Inclusion

The rest of this section describes how Uppaal can be used in some cases to check whether or not $A \sqsubseteq_{\mathrm{TR}} B$, via the construction of B^{err}, A^f and B^f.

Throughout this section, we assume that the visible actions of A are included in those of B. Recall that we assume that B does not contain committed locations or urgent channels.

The construction of B^{err}

If we want to check whether $A \sqsubseteq_{\mathrm{TR}} B$ for a deterministic automaton B, then we build an automaton B^{err}. This automaton is an adaption of construction in [AD94] and is constructed as follows. First, we add a location $error$ to B. Moreover, we add transitions $q \xrightarrow{a\ g} error$ for all locations q and all action labels a such that this transition is enabled if no other a-transition is enabled from q. Finally, we add an internal transition from q to $error$ with the guard $\neg\mathrm{Inv}(q)$ and remove all location invariants. This is formalized in the following definition.

Definition A.1.7 The automaton B^{err} is defined as follows.

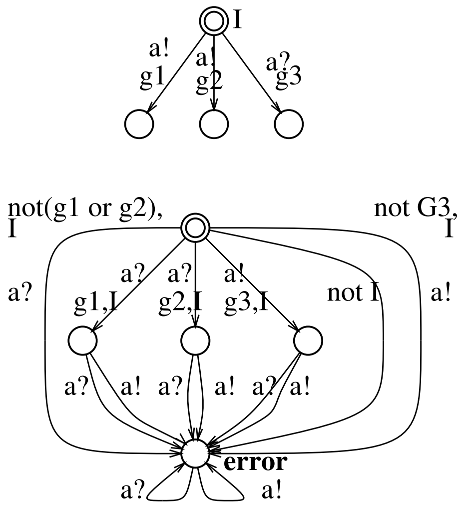


Figure A.5: An automaton and its error-construction

1. The locations of B^{err} are the locations of B together with the (fresh) location *error*,
2. the initial location of B^{err} is the initial location of B,
3. there are no location invariants in B^{err},
4. the transitions of the automaton B^{err} are given as follows, where q and q' range over locations in B, and a over visible actions in B and where $\overline{a}$ toggles the ! and ? in a, thus $\overline{b?} = b!$ and $\overline{b!} = b?$.

$$
\begin{aligned}
\bigcup_{q,a,q'} (\{q &\xrightarrow{\overline{a},g\wedge I,r}_{B^{err}} q' \mid q \xrightarrow{a,g,r}_B q', I = \mathrm{Inv}(q)\} \cup \\
&\{q \xrightarrow{\neg I} \mathit{error} \mid I = \mathrm{Inv}(q)\} \cup \\
&\{q \xrightarrow{\overline{a},g_{aq}\wedge I} \mathit{error} \mid I = \mathrm{Inv}(q), \\
&\quad g_{aq} = \neg \bigvee \{g \mid q \xrightarrow{a,g,r}_B q'\}\} \cup \\
&\{\mathit{error} \xrightarrow{a} \mathit{error}\}).
\end{aligned}
$$

Example A.1.8 Figure A.5 illustrates an automaton and its error-construction.

For a construction of B^{err} in the presence of urgent channels and shared variables, see [Jen99, JLS00]. This work also describes how to encode, what are called, *timed ready simulation relations* as reachability properties.

To decide whether $A \sqsubseteq_{\mathrm{TR}} B$, we check whether the *error*-location is reachable in the network consisting of A and B^{err}. Note that B^{err} is not deterministic, even not if B is so.

Lemma A.1.9 *Let B be a deterministic automaton. Every finite sequence over $actB \cup \mathbb{R}^{>0}$ is a trace of B^{err}. Such a sequence is a trace of B if and only if none of the executions in B^{err} with this trace reach the error location.*

Proposition A.1.10 *Assume that B is a deterministic automaton. Then $A \sqsubseteq_{\mathrm{TR}} B \iff$ error is not reachable in $(A \parallel B^{err}) \setminus \mathrm{Names}$.*

The construction of A^r and B^r

If B is non-deterministic, then we can try to make it deterministic by renaming its labels and then use the above method to verify trace inclusion.

Definition A.1.11 A *renaming function* is a function $h : \mathrm{Names} \to \mathrm{Names} \cup \{\tau\}$. For an automaton (resp. a TLTS) A, we denote by A^h the automaton (resp. the TLTS) obtained by replacing every visible action $a?$ in A by $h(a)?$ and $a!$ by $h(a)!$.

Lemma A.1.12 *Let A, B be automata (resp. TLTSs) and let h be a renaming function on A. Then*

$$A \sqsubseteq_{\mathrm{TR}} B \implies A^h \sqsubseteq_{\mathrm{TR}} B^h.$$

The preceding result shows that, for proving $A_1 \sqsubseteq_{\mathrm{TR}} B_1$, it suffices to find A_2, B_2 and a renaming function h such that: B_2 is deterministic, ${A_2}^h = A_1$, ${B_2}^h = B_1$ and $A_2 \sqsubseteq_{\mathrm{TR}} B_2$. The latter can be verified by Uppaal. Notice that $A_2 \not\sqsubseteq_{\mathrm{TR}} B_2$ does not imply $A_1 \not\sqsubseteq_{\mathrm{TR}} B_1$. Moreover, even if $A_1 \sqsubseteq_{\mathrm{TR}} B_1$, it is not always possible to find the required A_2, B_2 and h. (For instance, if A_1 consists of the transition $s \xrightarrow{a} s'$ and B_1 of $t \xrightarrow{\tau} t'$ and $t' \xrightarrow{a} t''$, then no such a h exists – note that τ is not in the domain of h.)

In our verification, we have constructed h, A_2 and B_2 from A_1, B_1 and a given (conjectured) step refinement f. First, we describe this construction for TLTSs L_1 and K_1, yielding TLTSs L_2 and K_2. Informally, a step refinement from L_1 to K_1 is a function f from the states of L_1 to the states of K_1 such that every transition $s \xrightarrow{a} s'$ of L_1 can be mimicked in K_1 by the transition $f(s) \xrightarrow{a} f(s')$. Here, we also allow internal transitions in L_1 to be mimicked by remaining in the same state in K_1. Formally, a *step refinement* from L_1 to K_1 is a function mapping the state space of L_1 to the state space of K_1 such that, if s is an initial state of L_1, then $f(s)$ is an initial state in K_1. For all transitions $s \xrightarrow{a} t$ of L_1 leaving from a reachable state s, we require that either $f(s) \xrightarrow{a}_{K_1} f(t)$ or that a is internal and $f(s) = f(t)$. A step refinement from an automaton A to an automaton B is a step refinement from the underlying TLTS of A to the underlying TLTS of B.

The idea behind the construction of L_2 and K_2 (on the level of TLTSs) from f and L_1 and K_1 is as follows. The step refinement f yields a correspondence between transitions in K_1 and transitions in L_1: the transition $s' \xrightarrow{a} t'$ in L_1 corresponds to all transitions in K_1 with $s \xrightarrow{a} t$ with $f(s) = s'$ and $f(t) = t'$. To remove the nondeterministic choices in K_1, we rename the actions on corresponding transitions with the same, fresh label, both in K_1 and in L_1. The TLTSs obtained from L_1 and K_1 in this way are denoted by ${L_1}^f$ and ${K_1}^f$ respectively.

Formally, the TLTSs ${L_1}^f$ and ${K_1}^f$ are obtained from L_1 and K_1 as follows: We start with the same state spaces and transition relations as in L_1 and K_1 respectively. Then, for all sources of nondeterminism in K_1, that is, all transitions $t \xrightarrow{a} t'$ in K_1 such that either $a = \tau$ or $\exists t'' \neq t' [t \xrightarrow{a}_{K_1} t'']$, we change the label a into $c_{t,a,t'}$. This yields $t \xrightarrow{c_{t,a,t'}}_{{K_1}^f} t'$ in ${K_1}^f$. Then, for all transitions in L_1, we replace $s \xrightarrow{a} s'$ by $s \xrightarrow{c_{f(s),a,f(s')}}_{{L_1}^f} s'$ in ${L_1}^f$ if and only if $f(s) \xrightarrow{c_{f(s),a,f(s')}}_{{K_1}^f} f(s')$. We require for the relabeling function $c_{\cdot,\cdot,\cdot}$ that

1. the label $c_{s,a,t}$ is not a visible action of A and B,

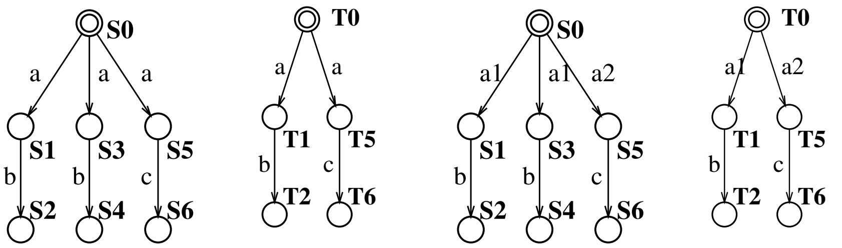


Figure A.6: The TLTSs L_1, K_1 and ${L_1}^f$ and ${K_1}^f$

2. $c_{s,a,t} \neq c_{s,a,t'}$ for $t \neq t'$,

3. $c_{s,a,t} \neq \tau$, and

4. $c_{s,a,s'} \neq c_{t,b,t'}$ for $a \neq b$.

Here, requirement 1 ensures that $c_{s,a,t}$ is fresh. The second and third requirements ensure that outgoing transitions of the same state get different labels and that no actions are relabeled into τ. The last requirement ensures that transitions with different labels get different labels after relabeling. Together these requirements ensure that the relabeled automaton obtained by the construction above is deterministic.

Example A.1.13 Consider the TLTSs in Figure A.6, where we omitted time passage actions $s \xrightarrow{d} s$, where d can be arbitrary. It is clear that the function $s_i \mapsto t_i$ for $i = 0, 1, 2, 5, 6$ and $s_3 \mapsto t_1, s_4 \mapsto t_2$ is a step refinement from L_1 to K_1 and that both the transitions $s_0 \xrightarrow{a} s_1$ and $s_0 \xrightarrow{a} s_3$ should synchronize with the transition $t_0 \xrightarrow{a} t_1$. Hence these transitions get the same action label by the renaming. Similarly, $s_0 \xrightarrow{a} s_1$ and $s_0 \xrightarrow{a} s_3$ should synchronize and get the same labels after renaming (but different from the three labels mentioned previously).

Now, take h to be the identity on the names of A and B and let $h(c_{s,a,t}) = a$ for the new names. Note that τ is not in the domain of h, but that $h(c)$ can be τ. In the example above, h is given by $\{a_1 \mapsto a, a_2 \mapsto a, b \mapsto b, c \mapsto c\}$. Now, we have the following.

Lemma A.1.14 *Let L and K be TLTSs, f a function from the state space of L to the state space of K, and h as defined in the preceding text. Then*

1. ${L^f}^h = L$ *and* ${K^f}^h = K$.

2. $L^f \sqsubseteq_{\mathrm{TR}} K^f \implies L \sqsubseteq_{\mathrm{TR}} K$.

Moreover, we claim that $A^f \sqsubseteq_{\mathrm{TR}} B^f$ if f is a step refinement from A to B. (Then Lemma A.1.14 yields that $A \sqsubseteq_{\mathrm{TR}} B$, which is a basic result for step refinements.) We do not use this claim in the verification.

Since TLTSs are usually infinite, it is relevant to investigate whether this construction on TLTSs can be lifted to automata. This means that, if L_1 is the TLTS underlying A and K_1 the one underlying B, we wish to find automata A^f and B^f whose underlying TLTSs are ${L_1}^f$ and ${L'}^f$ respectively.

We conjecture that this construction can indeed be lifted to the level of automata, provided that r is given by a (finite) expression over the clocks and locations of the automaton. Although the existence of a step refinement already implies trace inclusion, this construction (on automata) would yield a method to check this inclusion with Uppaal. Although a relabeling procedure can be complex in general, in our case study it is much faster than checking whether the function r is a step refinement.

Bibliography

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In E.A. Emerson and A.P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 419–434. Springer-Verlag, 2000.

[ABK98] L. Aceto, A. Burgueño, and K.G.Larsen. Model checking via reachabilty testing for timed automata. In *Proc. of the 3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Lisbon, Portugal, volume 1384 of *Lecture Notes in Computer Science*, pages 263–280. Springer-Verlag, 1998.

[ABS01] A. Annichini, A. Bouajjani, and M. Sighireanu. TReX: a tool for reachability analysis of complex systems. In G. Berry, H. Comom, and A. Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification,* Paris, France, volume 2102 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[AD94] R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[Agg94] S. Aggarwal. Time optimal self-stabilizing spanning tree algorithms. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1994. Available as Technical Report MIT/LCS/TR-632.

[AHJ01] L. de Alfaro, T.A. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In K. G. Larsen and M. Nielsen, editors, *Proceedings CONCUR 01,* Aalborg, Denmark, volume 2154 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[AHV93] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. *25th Annual ACM Symposium on Theory of Computing (STOC'93)*, pages 592–601, 1993.

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[AL92] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings*

REX Workshop on Real-Time: Theory in Practice, Mook, The Netherlands, June 1991, volume 600 of *Lecture Notes in Computer Science*, pages 1–27. Springer-Verlag, 1992.

[Alf97] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

[Alf99] L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In J.C.M. Baeten and S. Mauw, editors, *Proceedings CONCUR 99,* Eindhoven, The Netherlands, volume 1664 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[Alu98] R. Alur. Timed automata. In *NATO-ASI Summer School on Verification of Digital and Hybrid Systems*. Springer-Verlag, 1998. To appear.

[And98] Don Anderson. *FireWire system architecture: IEEE 1394*. MindShare, Inc., 1998. ISBN 0-201-69470-0.

[And99a] S. Andova. Process algebra with interleaving probabilistic parellel composition. Technical Report CSR 99-04, Eindhoven University of Technology, 1999.

[And99b] S. Andova. Process algebra with probabilistic choice. In *Proceedings of 5th AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)* Bamberg, Germany, May 1999, volume 1601 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[Apt81] K.R. Apt. Ten years of Hoare's logic: A survey–part I. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, 1981.

[BA95] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer-Verlag, 1995.

[Bai96] C. Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In R. Alur and T.A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification,* New Brunswick, NJ, USA, volume 1102 of *Lecture Notes in Computer Science*, pages 38–49. Springer-Verlag, July/August 1996.

[Bai98] C. Baier. On algorithmic verification methods for probabilistic systems. Habilitation Thesis, Universität Mannheim, 1998.

[BBK87] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51(1/2):129–176, 1987.

[BBS95] J. C. M. Baeten, J. A. Bergstra, and S. A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 121(2):234–255, 1995.

[BDG97] M. Bernardo, L. Donatiello, and R. Gorrieri. A tutorial on EMPA: a theory of concurrrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 1997.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A Model-Checking Tool for Real-Time Systems. In Hu and Vardi [HV98], pages 546–550.

[BEM00] C. Baier, B. Engelen, and M. Majster–Cederbaum. Deciding bisimilarity and similarity for probabilistic systems. *Journal of Computer and System Sciences*, 60(1):187–231, 2000.

[BH97] C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. In O. Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 119–130. Springer-Verlag, 1997.

[BJ91] K.G. Larsen B. Jonsson. Specification and refinement of probabilistic processes. In *Proceedings* 6^{th} *Annual Symposium on Logic in Computer Science,* Amsterdam, pages 266–277. IEEE Computer Society Press, 1991.

[BK86] J.A. Bergstra and J.W. Klop. Verification of an alternating bit protocol by means of process algebra. In W. Bibel and K.P. Jantke, editors, *Math. Methods of Spec. and Synthesis of Software Systems '85, Math. Research 31*, pages 9–23, Berlin, 1986. Akademie-Verlag. First appeared as: Report CS-R8404, CWI, Amsterdam, 1984.

[BK90] J.C.M. Baeten and J.W. Klop, editors. *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[BK98] C. Baier and M.Z. Kwiatkowska. On the verification of qualitative properties of probabilistic processes under fairness constraints. *Information Processing Letters*, 66(2):71–79, 1998.

[BLdRT00] G. Bandini, R.L. Lutje Spelberg, R.C.H. de Rooij, and W.J. Toetenel. Application of parametric model checking – the root contention protocol using LPMC. In *Proceedings of the 7th ASCI Conference,* Beekbergen, The Netherlands, Febuary 2000, pages 73–85, 2000.

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, Reading, Massachusetts, USA, 1999.

[BS87] T. Bolognesi and S.A. Smolka. Fundamental results for the verification of observational equivalence: a survey. In H. Rudin and C.H. West, editors, *Protocol Specification, Testing, and Verification, VII,* Zürich, Switzerland, pages 165–178. North-Holland, May 1987.

[BS00] C. Baier and M.I.A. Stoelinga. Norm functions for probabilistic bisimulations with delays. In J. Tiuryn, editor, *Proceedings of 3rd International Conference on Foundations of Science and Computation Structures (FOSSACS),* Berlin, Germany, March 2000, volume 1784 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2000.

[BST00] G. Bandini, R.L. Lutje Spelberg, and W.J. Toetenel. Parametric verification of the IEEE 1394a root contention protocol using LPMC. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications (RTCSA 2000),* Cheja Island, South Korea, December 2000, 2000.

[Cas93] C.G. Cassandras. *Discrete event systems. Modeling and Performance Analysis*. Aksen Associates – Irwin, 1993.

[CC92] L. Christoff and I. Christoff. Efficient algorithms for verification of equivalences for probabilistic processes. In K.G. Larsen and A. Skou, editors, *Proceedings of the 3rd International Workshop on Computer Aided Verification,* Aalborg, Denmark, volume 575 of *Lecture Notes in Computer Science*, pages 310–321. Springer-Verlag, 1992.

[CDSY99] R. Cleaveland, Z. Dayar, S. A. Smolka, and S. Yuen. Testing preorders for probabilistic processes. *Information and Computation*, 154(2):93–148, 1999.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Cambridge, Massachusetts, 1999.

[CHM90] J. Cheriyan, T. Hagerup, and K. Mehlhorn. Can a maximum flow be computed in $\o(nm)$ time? In M. Paterson, editor, *Proceedings* 17^{th} *ICALP,* Warwick, volume 443 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[Chr90a] I. Christoff. Testing equivalence and fully abstract models of probabilistic processes. In Baeten and Klop [BK90].

[Chr90b] I. Christoff. *Testing equivalences for probabilistic processes*. PhD thesis, Department of Computer Science, Uppsala University, Sweden, 1990.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill, Inc., 1990.

[Coh80] D.L. Cohn. *Measure Theory*. Birkhaeuser, Boston, 1980.

[Coh89] L.J. Cohen. *An introduction to the philosophy of induction and probability*. Clarendon Press, Oxford, 1989.

[CS01] A. Collomb–Annichini and M. Sighireanu. Parameterized reachability analysis of the IEEE 1394 Root Contention Protocol using TReX. In P. Pettersson and S. Yovine, editors, *Proceedings of the Workshop on Real-Time Tools (RT-TOOLS'2001)*, 2001.

[CW87] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the* 19^{th} *Annual ACM Symposium on Theory of Computing,* New York City, pages 1–6. ACM, 1987.

[D'A99] P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. PhD thesis, University of Twente, November 1999. Available via `http://wwwhome.cs.utwente.nl/~dargenio`.

[dAKN$^+$00] L. de Alfaro, M. Z. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In S. Graf and M. Schwarzbach, editors, *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems,* Berlin, Germany, volume 1785 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[DGJP99] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labeled Markov systems. In *International Conference on Concurrency Theory*, pages 258–273, 1999. full version available via `http://www-acaps.cs.mcgill.ca/~ prakash`.

[DGRV00] M.C.A. Devillers, W.O.D. Griffioen, J.M.T Romijn, and F.W. Vaandrager. Verification of a leader election protocol: Formal methods applied to IEEE 1394. *Formal Methods in System Design*, 16(3):307–320, June 2000.

[DHKK01] P.R. D'Argenio, H.H. Hermans, J.-P. Katoen, and R. Klaren. Modest - a modelling and description language for stochastic timed systems. In *Proceeding of PAPM–ProbMiV 2001,* Aachen, Germany, volume 2165 of *Lecture Notes in Computer Science*, pages 87–104. Springer-Verlag, 2001.

[Dij69] E. W. Dijkstra. Structured programming. In *Second NATO Conference on Software Engineering Techniques, Rome, Italy*, pages 84–88. NATO, 1969.

[Dil98] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In Hu and Vardi [HV98], pages 197–212.

[DKRT97a] P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time! In E. Brinksma, editor, *Proceedings of the Third Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Enschede, The Netherlands, volume 1217 of *Lecture Notes in Computer Science*, pages 416–431. Springer-Verlag, April 1997.

[DKRT97b] P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time! In *Proc. of the 3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Enschede, The Netherlands, volume 1217 of *Lecture Notes in Computer Science*, pages 416–431. Springer-Verlag, 1997.

[DNH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[DP01] J. Desharnais and P. Panangaden. Continuous stochastic logic characterises bisimulation of continuous–time Markov processes, 2001. available via `http://www-acaps.cs.mcgill.ca/ ~ prakash/`.

[Fer90] J.-C. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13:219–236, 1989/1990.

[FLP85] M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[FS01] C. Fidge and C. Shankland. But what if i don't want to wait forever? In J.M.T. Romijn S. Maharaj, C. Shankland, editor, *Proceedings of the International Workshop on Application of Formal Methods to the IEEE1394 Standard* Berlin, March 2001, 2001. submitted.

[GBS94] R. Gupta, S. Bhaskar, and S.A. Smolka. On randomization in sequential and distributed algorithms. *ACM Computing Surveys*, 28 (1):7–86, 1994.

[gDUoT] TVS group Delft University of Technology. PMC: Prototype Model Checker. `http://tvs.twi.tudelft.nl/toolset.html`.

[GHR93] N. Götz, U. Herzog, and M. Rettelbach. TIPP - a stochastic process algebra. In *Proceedings of the Workshop on Process Algebra and Performance Modelling*, 1993. Technical Report,University of Edinburgh.

[GJS90] A. Giacalone, C. C. Jou, and S. A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In M. Broy and C.B. Jones, editors, *Proceedings of the Working Conference on Programming Concepts and Methods*, pages 443–458. North–Holland, 1990.

[Gla93] R.J. van Glabbeek. The linear time – branching time spectrum II (the semantics of sequential systems with silent moves). In E. Best, editor, *Proceedings CONCUR 93,* Hildesheim, Germany, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1993.

[Gla01] R.J. van Glabbeek. The linear time — branching time spectrum I. The semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North-Holland, 2001.

[GLV97] S.J. Garland, N.A. Lynch, and M. Vaziri. IOA: A language for specifiying, programming, and validating distributed systems, September 1997. Available through URL `http://larch.lcs.mit.edu:8001/~garland/ioaLanguage.html`.

[Gly89] P. W. Glyn. 1989. *Proceedings of IEEE*, 77 (1):4–23, 1989.

[GSST95] R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings* 5^{th} *Annual Symposium on Logic in Computer Science,* Philadelphia, USA, pages 130–141. IEEE Computer Society Press, 1995.

[GV98] W.O.D. Griffioen and F.W. Vaandrager. Normed simulations. In A.J. Hu and M.Y. Vardi, editors, *Proceedings of the 10th International Conference on Computer Aided Verification,* Vancouver, BC, Canada, volume 1427 of *Lecture Notes in Computer Science*, pages 332–344. Springer-Verlag, 1998.

[GW89] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In G.X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, 1989. Full version available as Report CS-R9120, CWI, Amsterdam, 1991.

[Hac75] I. Hacking. *The emerge of Probability*. Cambridge University Press, New York, 1975.

[Hal50] P.R. Halmos. *Measure Theory*. Van Nostrand Reinhold Company Inc, New York, 1950.

[Han91] H.A. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, Department of Computer Systems, Uppsala University, 1991. DoCS 91/27.

[Han94] H.A. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real-Time Safety Critical Systems*. Elsevier, 1994.

[Har87] D. Harel. Statecharts: a visual approach to complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[Her99] H.H. Hermanns. *Interactive Markov Chains*. PhD thesis, University of Erlangen–Nürnberg, July 1999. Available via `http://wwwhome.cs.utwente.nl/~hermanns`.

[HHK95] M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings 36^{th} IEEE Symposium on Foundations of Computer Science*, pages 453–462. IEEE, 1995.

[HHW97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *Software Tools for Technology Transfer*, volume 1, 1997. Also available via `http://www-cad.eecs.berkeley.edu/ ~tah/-HyTech/`.

[Hil96] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[HJ90] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proc. IEEE Real-Time Systems,* Orlando, Florida*, 1990*, pages 278–287. IEEE Computer Society Press, 1990.

[HJ94] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):515–535, 1994.

[HJL93] C. Heitmeyer, R. Jeffords, and B. Labaw. A benchmark for comparing different approaches for specifying and verifying real-time systems. In *Proceedings 10th International Workshop on Real-Time Operating Systems and Software*, 1993.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, Englewood Cliffs, 1985.

[HP00] O.M. Herescu and C. Palamidessi. Probabilistic asynchronous π-calculus. In J. Tiuryn, editor, *Proceedings of 3rd International Conference on Foundations of Science and Computation Structures (FOSSACS),* Berlin, Germany, March 2000, volume 1784 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2000.

[HRSV01] T.S. Hune, J.M.T. Romijn, M.I.A. Stoelinga, and F.W. Vaandrager. Linear parametric model checking of timed automata. In T. Margeria and W. Yi, editors, *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems,* Genova, Italy, volume 2031 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[HRSV02] T.S. Hune, J.M.T. Romijn, M.I.A. Stoelinga, and F.W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 2002. To appear.

[HSLL97] K. Havelund, A. Skou, K. G. Larsen, and K. Lund. Formal modelling and analysis of an audio/video protocol: An industrial case study using uppaal. In *Proc. of the 18th IEEE Real-Time Systems Symposium,* San Francisco CA, USA, pages 2–13, 1997.

[HT92] T. Huynh and L. Tian. On some equivalence relations for probabilistic processes. *Fundamenta Informaticae*, 17:211–234, 1992.

[Hui01] M. Huisman. *Reasoning about JAVA programs in higher order logic with PVS and Isabelle*. PhD thesis, Computer Science Institute, February 2001.

[Huy50] Chr. Huygens. *Van Rekeningh in spelen van geluck (in Dutch)*. 1650. Latin version *De rationiniis in ludo aleae* appeared in 1660. A modern dutch version, annotated and translated by W. Kleijne has been published by *epsilon uitgaven*, Utrecht.

[HV98] A.J. Hu and M.Y. Vardi, editors. *Proceedings of the 10th International Conference on Computer Aided Verification,* Vancouver, BC, Canada, volume 1427 of *Lecture Notes in Computer Science*. Springer-Verlag, June/July 1998.

[IEE96] IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Std 1394-1995, August 1996.

[IEE00a] IEEE Computer Society. IEEE Standard for a High Performance Serial Bus – Amendment 1. Std 1394a-2000, May 2000.

[IEE00b] IEEE Computer Society. P1394.3 Draft Standard for a High Performance Serial Bus (Supplement). Draft 1.2, June 2000.

[IEE01a] IEEE Computer Society. P1394.3 Draft Standard for a High Performance Serial Bus (Supplement). Draft 0.17, June 2001.

[IEE01b] IEEE Computer Society. P1394b Draft Standard for a High Performance Serial Bus (Supplement). Draft 1.11, March 2001.

[Jen99] H.E. Jensen. *Abstraction-Based Verification of Distributed Systems*. PhD thesis, Department of Computer Science, Aalborg University, Denmark, June 1999.

[JLS00] H. E. Jensen, K.G. Larsen, and A. Skou. Scaling up Uppaal - automatic verification of real-time systems using compositionality and abstraction. In J. Mathai, editor, *Proceedings of the 6th International School and Symposium on Formal Techniques and Fault Tolerant Systems (FTRTFT00),* Pune, India, September 2000, volume 1926 of *Lecture Notes in Computer Science*, pages 19–30. Springer-Verlag, 2000.

[JS90] C-C. Jou and S.A. Smolka. Equivalences, congruences and complete axiomatizations for probabilistic processes. In Baeten and Klop [BK90], pages 367–383.

[JY01] B. Jonsson and W. Yi. Compositional testing preorders for probabilistic processes. *Theoretical Computer Science*, 2001.

[KHR97] L. Kühne, J. Hooman, and W.P. de Roever. Towards mechanical verification of parts of the IEEE P1394 serial bus. In I. Lovrek, editor, *Proceedings of the 2nd International Workshop on Applied Formal Methods in System Design,* Zagreb, pages 73–85. University of Zagreb, Faculty of Electrical Engineering and Computing, 1997.

[KL92] A. Skou K. Larsen. Compositional verification of probabilistic processes. In W.R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 456–471. Springer-Verlag, 1992.

[KLL+97] K.J. Kristoffersen, F. Laroussinie, K.G. Larsen, P. Pettersson, and W. Yi. A compositional proof of a real-time mutual exclusion protocol. In M. Bidoit and M. Dauchet, editors, *Proceedings TAPSOFT'97: Theory and Practice of Software Development,* Lille, France, volume 1214 of *Lecture Notes in Computer Science*, pages 565–579. Springer-Verlag, April 1997.

[KNP02] M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In P. Stevens and J.P. Katoen, editors, *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Grenoble, France, Lecture Notes in Computer Science. Springer-Verlag, April 2002. to appear.

[KNS01] M. Z. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the ieee1394 firewire root contention protocol. In J.M.T. Romijn S. Maharaj, C. Shankland, editor, *Proceedings of the International Workshop on Application of Formal Methods to the IEEE1394 Standard* Berlin, March 2001, 2001. submitted.

[KNSS01] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 268, 2001.

[KS90] P. Kannelakis and S. Smolka. CCS expressions, finite state processes and three problems of equivalence. *Information and Computation*, pages 43–68, 1990.

[LaF97] D. LaFollette. SubPHY Root Contention, Overhead transparencies, August 1997. Available through URL `ftp://gatekeeper.dec.com/pub/-standards/io/1394/ P1394a/Documents/97-043r0.pdf`.

[Lam87] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, February 1987.

[LLPY97] K.G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structures and state-space reduction. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society press, 1997.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1(1+2):134–152, 1997. Also available via `http://www.docs.uu.se/docs/rtmv/-papers/ lpw-sttt97.pdf`.

[LR81] D. Lehmann and M. Rabin. On the advantage of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of the 8th ACM Symposium on Principles of Programming Languages*, pages 133–138, 1981.

[LS91] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.

[LSS94] N.A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings of the 13th Annual ACM Symposium on the Principles of Distributed Computing*, pages 314–323, Los Angeles, CA, 1994.

[LT89] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[LTA98] R.F. Lutje Spelberg, W.J. Toetenel, and M. Ammerlaan. Partition refinement in real-time model checking. In A.P. Ravn and H. Rischel, editors, *Proceedings of the Fifth International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98),* Lyngby, Denmark, volume 1486 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, 1998.

[Lut97] S.P. Luttik. Description and formal specification of the Link layer of P1394. In I. Lovrek, editor, *Proceedings of the 2nd International Workshop on Applied Formal Methods in System Design,* Zagreb, pages 43–56, 1997. Also available as Report SEN-R9706, CWI, Amsterdam. See URL `http://www.cwi.nl/~luttik/`.

[LV95] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations, I: Untimed systems. *Information and Computation*, 121(2):214–233, 1995.

[LV96a] N.A. Lynch and F.W. Vaandrager. Action transducers and timed automata. *Formal Aspects of Computing*, 8(5):499–538, 1996.

[LV96b] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations, II: Timing-based systems. *Information and Computation*, 128(1):1–25, July 1996.

[Lyn96] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., San Fransisco, California, 1996.

[MCB84] M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[McI99] A.K. McIver. Quantitative program logic and performance. In J.-P. Katoen, editor, *Proceedings of 5th AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)* Bamberg, Germany, May 1999, volume 1601 of *Lecture Notes in Computer Science*, pages 19–32. Springer-Verlag, 1999.

[McM92] K.L. McMillan. The SMV system, draft, February 1992. Available through URL `www.cs.cmu.edu/~modelcheck/smv.html`.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs, 1989.

[MM99] C.C. Morgan and A.K. McIver. pGCL: formal reasoning for random algorithms. In *South African Computer Journal*, volume 22, pages 14–27, 1999. Special issue on the 1998 Winter School on Formal and Applied Computer Science.

[MMT91] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings CONCUR 91,* Amsterdam, volume 527 of *Lecture Notes in Computer Science*, pages 408–423. Springer-Verlag, 1991.

[MS00] S. Maharaj and C. Shankland. A survey of formal methods applied to leader election in IEEE 1394. *Journal of Universal Computer Science*, pages 1145–1163, 2000.

[NCI99] M. Narashima, R. Cleaveland, and P. Iyer. Probabilistic temporal logics via the modal mu-calculus. In W. Thomas, editor, *Proceedings of the International Conference on Foundations of Science and Computation Structures (FOSSACS)*, volume 1578, pages 288–305, 1999.

[Nyu97] Takayuki Nyu. Modified Tree-ID Process for Long-haul Transmission and Long PHY_DELAY, Overhead transparencies, 1997. Available through URL `ftp://gatekeeper.dec.com/pub/standards/-io/1394/P1394a/Documents/97-051r1.pdf`.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5^{th} *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[Pet96] I. Peterson. *Fatal Defect, chasing killer computer bugs*. Vintage, 1996.

[PSL97] A. Pogosyants, R. Segala, and N.A. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. In M. Mavronicolas and Ph. Tsigas, editors, *Proceedings of 11th International Workshop on Distributed Algorithms (WDAG'97),* Saarbrucken, Germany, September 1997, volume 1320 of *Lecture Notes in Computer Science*, pages 111–125. Springer-Verlag, 1997. Also, Technical Memo MIT/LCS/TM-555, Laboratory for Computer Science, Massachusetts Institute of Technology.

[PSL98] A. Philippou, O. Sokolsky, and I. Lee. Weak bisimulation for probabilistic systems. In D. Sangiorgi and Robert de Simone, editors, *Proceedings CONCUR 98,* Nice, France, volume 1466 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[PT87] R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[Put94] M. Puterman. *Markov Decision Processes*. John Wiley and Sons, 1994.

[RE98] W.P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge Tracts in Theoretical Computer Science 47. Cambridge University Press, 1998.

[Rei85] W. Reisig. *Petri nets – an introduction*. EATCS Monographs on Theoretical Computer Science, Volume 4. Springer-Verlag, 1985.

[SC01] R. Segala and S. Cattani. Personal communication, 2001.

[Seg95a] R. Segala. Compositional trace–based semantics for probabilistic automata. In *Proc. CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, pages 234–248, 1995.

[Seg95b] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1995. Available as Technical Report MIT/LCS/TR-676.

[Seg96] R. Segala. Testing probabilistic automata. In *Proc. CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 299–314, 1996.

[SGSL98] R. Segala, R. Gawlick, J.F. Søgaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. *Information and Computation*, 141(2):119–171, March 1998.

[Sha99] C. Shankland. Using E-LOTOS to pick a leader. *Proceedings of the Workshop on Formal Methods Computation,* Ullapool, UK, September 1999, pages 143–162, 1999.

[SL95] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[SS01] D.P.L. Simons and M.I.A. Stoelinga. Mechanical verification of the IEEE 1394a root contention protocol using Uppaal2k. *Springer International Journal of Software Tools for Technology Transfer (STTT)*, 2001. Accepted for publication.

[Sto99a] M.I.A. Stoelinga. Gambling for leadership: Verification of root contention in IEEE 1394. Technical Report CSI-R9904, Computing Science Institute, University of Nijmegen, 1999.

[Sto99b] M.I.A. Stoelinga. Gambling for leadership: Verification of root contention in IEEE 1394. Technical Report CSI-R9904, Computing Science Institute, University of Nijmegen, 1999.

[Sto01] M.I.A. Stoelinga. Fun with FireWire: Experiences with verifying the IEEE1394 Root Contention Protocol. In J.M.T. Romijn S. Maharaj, C. Shankland, editor, *Proceedings of the International Workshop on Application of Formal Methods to the IEEE1394 Standard* Berlin, March 2001, pages 35–38, 2001. Also, Technical Rapport CSI-R0107, Computing Science Institute, University of Nijmegen, March 2001. Submitted.

[SV99a] C. Shankland and A. Verdejo. Time, E-LOTOS and the FireWire. In M. Ajmone Marsan, J. Quemada, T. Robles, and M. Silva, editors, *Proceedings of the Workshop on Formal Methods and Telecommunications, (WFMT'99)* Zaragoza, Spain, September 1999, pages 103–119. Univ. of Zaragoza Press, 1999.

[SV99b] M.I.A. Stoelinga and F.W. Vaandrager. Root Contention in IEEE 1394. In J.-P. Katoen, editor, *Proceedings of 5th AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)* Bamberg, Germany, May 1999, volume 1601 of *Lecture Notes in Computer Science*, pages 53–75. Springer-Verlag, 1999. Also, Technical Rapport CSI-R9905, Computing Science Institute, University of Nijmegen, May 1999.

[SV02a] M.I.A. Stoelinga and F.W. Vaandrager. Gambling together in Monte Carlo: Step refinements for probabilistic automata. Technical Report CSI-R02xx, Computing Science Institute, University of Nijmegen, 2002. In preparation.

[SV02b] M.I.A. Stoelinga and F.W. Vaandrager. Testing scenario's for probabilistic automata. Technical Report CSI-R0201, Computing Science Institute, University of Nijmegen, 2002. To appear.

[Tan81] A.S. Tanenbaum. *Computer networks*. Prentice-Hall International, Englewood Cliffs, 1981.

[Upp] Uppaal distribution. Available via `http://www.uppaal.com`.

[Var85] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *IEEE Symposium on Foundations of Computer Science*, pages 327–338, 1985.

[Wei89] W.P. Weijland. *Synchrony and asynchrony in process algebra*. PhD thesis, University of Amsterdam, 1989.

[Wol98] P. Wolper. Verification: dreams and reality. Inaugural lecture to the course *the algorithmic verification of reactive systems*, 1998. available via the URL `www.montefiore.ulg.ac.be/~pw/cours/francqui.html`.

[WSS97] S.-H. Wu, S.A. Smolka, and E. Stark. Composition and and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 1-2:1–38, 1997.

[Yi90] Wang Yi. Real-time behaviour of asynchronous agents. In Baeten and Klop [BK90], pages 502–520.

[Yi94] W. Yi. Algebraic reasoning for real-time probabilistic processes with uncertain information. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Proceedings of the Third International School and Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94),* Lübeck, Germany, September 1994, volume 863 of *Lecture Notes in Computer Science*, pages 680–693. Springer-Verlag, 1994.

[YL92] W. Yi and K. Larsen. Testing probabilistic and nondeterministic processes. *Proc. Protocol, Specification, Testing, Verification XII*, pages 47–61, 1992.

[Yov98] S. Yovine. Model checking timed automata. In G. Rozenberg and F.W. Vaandrager, editors, *Lectures on Embedded Systems*, volume 1494 of *Lecture Notes in Computer Science*, pages 114–152. Springer-Verlag, October 1998.

Samenvatting

> Het begrip toeval is niet tegenstrijdig, het wordt pas tegenstrijdig als ik probeer het te formaliseren.
> H. Freudenthal

Motivatie

Dit proefschrift gaat over het correct bewijzen van software en hardware systemen. In het bijzonder leggen we ons toe op het correct bewijzen van *reactieve systemen* waarin kansen, tijdseisen en/of parameters een belangrijke rol spelen.

Reactieve systemen zijn hardware– of softwaresystemen die in continue wisselwerking met hun omgeving staan. Zij reageren op signalen uit hun omgeving en sturen op hun beurt ook weer signalen die hun omgeving beïnvloeden. Voorbeelden van zulke systemen zijn besturingssoftware voor kerncentrales en protocollen voor het oversturen van informatie over het Internet.

Formele Methoden en Formele Verificatie

Door hun complexe interactie met de omgeving is het ontwerp en de realisatie van reactieve systemen een lastig en foutgevoelig proces. Diverse voorbeelden, zoals de softwarefout die de crash van de Ariane 5 raket veroorzaakte en de hardwarefout in de Pentium V, onderstrepen dit.

Correctheid van soft– en hardware vormt traditioneel een belangrijk onderwerp van studie binnen het informatica–onderzoek. Dit onderzoek heeft een veelheid aan talen, technieken en tools (computergereedschappen) voortgebracht om hardware en software systemen te beschrijven en te analyseren. Technieken, talen en tools die een wiskundige onderbouwing hebben worden ook wel *formele methoden* genoemd.

Deze wiskundige aanpak heeft als voordeel dat ze heel precies is. Wanneer de systeemvereisten van een product formeel gespecificeerd worden, ligt de interpretatie vast en kunnen er in latere fases van de productontwikkeling minder gemakkelijk misverstanden over ontstaan. Bovendien kunnen formele redeneringen en analyses over (formele) systeemmodellen op hun correctheid gecontroleerd worden.

Formele methoden kunnen worden ingezet binnen het gehele ontwikkeltraject van software en hardware, van de inventarisatie van de wensen van de klant, het ontwerp, de implementatie, de testfase tot en met het systeemonderhoud. Dit proefschrift richt zich in het bijzonder op *formele verificatie.* Dit houdt in dat men van een systeem wiskundig bewijst dat het voldoet aan de gewenste systeemvereisten, ook wel *specificatie* genoemd. Ook hier zorgt de wiskundige aanpak voor een grote mate van zekerheid. Een systeem dat formeel correct bewezen is, heeft een gerede kans dat het ook doet wat het moet doen.

Dit laatste geldt in mindere mate voor correctheidsargumenten die gebaseerd zijn op informele of semi–formele methoden. Indien de correctheid van een systeem met informele redeneringen aangetoond wordt, kunnen er gemakkelijk onduidelijkheden, impliciete — en mogelijk onjuiste aannamen — en onvolkomenheden ontstaan. Informele en semi–formele methoden hebben het voordeel dat zij sneller en makkelijker toe te passen zijn en complexere systemen aankunnen. Hoewel zij tot nu toe de voorkeur van de industrie hebben, worden formele methoden steeds vaker toegepast.

Tijd, Parameters en Kansen

Dit proefschrift gebruikt formele methoden om fundamenteel inzicht te krijgen in systemen waarin kansen, parameters en tijd een belangrijke rol spelen. Kansen, tijd en parameters zijn alle drie belangrijke mechanismen om de werkelijkheid te beschrijven. Zij spelen ook een belangrijke rol in het modelleren en construeren van software en hardware.

Tijd speelt in veel reactieve systemen een belangrijke rol, omdat veel reactieve systemen tijdkritisch zijn, dat wil zeggen zij moeten aan bepaalde tijdseisen voldoen om correct te zijn. Zo is de timing van het openen en sluiten van de bomen van een spoorwegovergang kritisch, omdat de bomen gesloten moeten zijn voordat de trein bij de overgang is.

Parameters maken het mogelijk om een klasse van systemen met dezelfde structuur te beschrijven. Men kan zich dan afvragen voor welke parameterwaarden het systeem correct is en voor welke niet. Inzicht in de voorwaarden op de parameters die nodig zijn voor de correctheid van het systeem, leveren belangrijke informatie over het functioneren ervan. Zij geven bijvoorbeeld inzicht in beperkingen op de toepassing van het systeem in toekomstige applicaties of binnen een andere context.

Een belangrijke plaats in dit proefschrift wordt ingenomen door systemen waarin *kansen* een rol spelen, de zogeheten probabilistische systemen. Kansen zijn, niet alleen in de informatica, een belangrijk hulpmiddel om systemen te beschrijven en analyseren. Zo vormen kansen een belangrijk hulpmiddel om de prestatie (performance) van reactieve systemen te analyseren. Ook worden zij gebruikt om onbetrouwbare systeemcomponenten te modelleren.

In dit proefschrift zijn we echter vooral geïnteresseerd in de toepassing van kansen in gedistribueerde algoritmen en communicatieprotocollen. In deze computersystemen — een speciale klasse van reactieve systemen — worden bepaalde beslissingen genomen op basis van een kansmechanisme, bijvoorbeeld het gooien van een dobbelsteen.

Het gebruik van kansmechanismen in deze systemen heeft drie belangrijke voordelen. Probabilistische methoden zijn vaak efficiënter in tijd– of geheugengebruik dan niet–probabilistische. Bovendien kan het gebruik van probabilistische keuzen leiden tot uniformere oplossingen. Tenslotte zijn er problemen, zoals het Byzantijnse generaalsprobleem, waarvoor bewezen is dat klassieke oplossingen niet bestaan, maar probabilistische wel. Het gebruik van kansmechanismen heeft echter ook een belangrijk nadeel. Het ontwerp en de verificatie van op kansen gebaseerde reactive systemen zijn meestal zeer complex. Zelfs specialisten op dit gebied hebben oplossingen bedacht die later niet correct bleken. Dit onderstreept nogmaals de noodzaak van rigoureuze wiskundige analyse.

Alea jacta est

Dit proefschrift presenteert een aantal wiskundige methoden voor het formeel specificeren en verifiëren van systemen met kansen, tijdseisen en parameters, waarbij probabilistische systemen de hoofdmoot vormen. We gebruiken Formele methoden om fundamenteel inzicht te krijgen in de eigenschappen en het gedrag van dergelijke systemen. We houden ons bezig met vragen als: *Hoe kunnen we systemen met kansen, tijd en/of parameters het best beschrijven? Hoe kan het gedrag van dergelijke systemen geformaliseerd worden? Wat zijn goede methoden om zulke systemen correct te bewijzen?* Tenslotte bekijkt dit proefschrift een kleine industriële toepassing waarin tijd, kansen en parameters voorkomen, te weten het Root Contention–protocol uit de IEEE 1394 'FireWire' standaard.

Resultaten

De belangrijkste resultaten van dit proefschrift worden uiteengezet in de Hoofdstukken 4 tot en met 7. Zij kunnen als volgt worden samengevat.

- Hoofdstuk 4 presenteert een testing scenario dat een rechtvaardiging geeft voor de bestaande definitie van het externe gedrag van probabilistische systemen uit de literatuur.
- In Hoofdstuk 5 ontwikkelen we een bewijstechniek, namelijk een bepaald type probabilistische (bi–)simulatie. Deze (bi–)simulatie staat het toe om, in zekere mate, te abstraheren van interne berekeningsstappen, maar kan toch efficiënt geautomatiseerd worden.
- Hoofdstuk 6 presenteert een zogenaamde *model checking* techniek die het mogelijk maakt om voor getimede systemen de exacte parametervergelijkingen te synthetiseren die nodig zijn, wil het systeem een bepaalde eigenschap bezitten.
- In Hoofdstuk 7 worden methoden en technieken uit de voorgaande hoofdstukken gebruikt om het Root Contention–protocol uit de IEEE 1394 standaard correct te bewijzen. We analyseren zowel het probabilistische, als het parametrische en het tijdsgedrag van dit protocol.

Overzicht

Hoofdstuk 1 leidt het proefschrift in. Het bespreekt de modellen die in de rest van het proefschrift gebuikt worden voor het analyseren van reactieve systemen. Deze modellen zijn gebaseerd op toestandovergangssystemen. In dit hoofdstuk wordt uitgelegd hoe het basismodel kan worden uitgebreid met tijd, parameters en kansen. Voorts bespreekt Hoofdstuk 1 de basisideeën achter de verificatiemethoden die we verder ontwikkelen en toepassen in latere hoofdstukken. De modellerings– en verificatiemethoden worden geïllustreerd aan de hand van een klassiek voorbeeld uit de literatuur, namelijk een sterk vereenvoudigd model van een spoorwegovergang.

Hoofdstuk 2 van dit proefschrift bespreekt het probabilistische–automatenmodel (PA–model). Dit raamwerk is ontwikkeld door Roberto Segala voor het beschrijven en analyseren van reactieve systemen met discrete kansen. Het PA–model vormt de basis voor volgende hoofdstukken in het proefschrift en Hoofdstuk 2 legt de achterliggende gedachten en theoretische concepten uit. Binnen het PA–model worden systemen gemodelleerd als toestandsovergangsmodellen. Dat wil zeggen, op ieder moment bevindt het systeem zich in een bepaalde toestand en veranderingen in het systeem worden gemodelleerd met behulp van toestandsovergangen, van de oude toestand naar de nieuwe toestand van het systeem. Naast probabilistische keuze, is ook nondeterministische keuze essentieel in het PA–model. De combinatie van beide maakt dit model vrij lastig. In het bijzonder wordt uitgelegd hoe deze combinatie leidt tot de notie van *trace distributies* van een PA, die het externe gedrag van een probabilistisch systeem beschrijven.

In Hoofdstuk 4 wordt een testing scenario gepresenteerd voor probabilistische automaten. Dit wil zeggen dat we op een intuïtieve manier aangeven hoe het externe gedrag van een PA geobserveerd kan worden. Dit doen we door de uitvoer een aantal onafhankelijke executies van de PA te analyseren met behulp van statistische methoden. Iedere executie van de PA levert een rijtje acties. Bij een willekeurige verzameling rijtjes toetsen we de hypothese "Deze verzameling rijtjes is gegenereerd door de gegeven PA." De klassieke wiskundige theorie over hypothese toetsen zal de hypothese aannemen dan wel verwerpen. Zo kunnen we van iedere verzameling rijtjes bepalen of het redelijk is om aan te nemen dat deze van een gegeven PA afkomstig is. Het observeerbare gedrag van een PA wordt daarmee gedefinieerd als precies die verzamelingen rijtjes die redelijkerwijs afkomstig zijn van de PA, waarvoor de hypothese dus wordt aangenomen.

Dit testing scenario, dat wil zeggen deze definitie van observeerbaar gedrag, levert precies de trace distributies als observaties van een PA op. Preciezer gezegd, twee PAs hebben dezelfde trace distributies precies dan als zij dezelfde observaties hebben. Dit is een belangrijke aanwijzing dat we met de trace distributies inderdaad een geschikte definitie van het concept 'extern gedrag' te pakken hebben.

Een belangrijke technische bijdrage van dit hoofdstuk is het zogenaamde *approximatie–inductie principe*. Deze stelling zegt dat als twee eindig vertakkende PAs hetzelfde eindige gedrag hebben, zij ook hetzelfde oneindige gedrag hebben. Met andere woorden, om in te zien of twee PAs precies hetzelfde gedrag hebben, volstaat het te kijken naar de eindige gedragingen van de PAs.

Hoofdstuk 5 is gewijd aan bisimulatie– en simulatierelaties voor PAs. Bisimulaties zijn relaties op de toestandsruimte van een PA die toestanden met hetzelfde stapsgewijze gedrag aan elkaar relateren. Men kan bewijzen dat toestanden met hetzelfde stapsgewijze gedrag ook hetzelfde globale gedrag hebben, dat wil zeggen, dezelfde verzameling trace distributies. Bisimulaties vormen dus een methode om gelijkheid van trace distributies aan te tonen. Deze methode is veel eenvoudiger en gemakkelijker te automatiseren dan een direct bewijs van trace distributie–gelijkheid. Zoals uitgelegd in Hoofdstuk 2, komen correctheidsbewijzen vaak neer op het bewijzen van trace distributie–gelijkheid of trace distributie–inclusie. Daarmee zijn bisimulaties een nuttig hulpmiddel voor systeemverificatie.

Simulatierelaties kunnen worden gezien als asymmetrische versies van bisimulatierelaties. Het verschil is dat als een toestand s is gerelateerd aan een toestand t, t al het stapsgewijze gedrag van s heeft, maar misschien kan t meer. Een gevolg hiervan is dat alle trace distributies van s ook trace distributies zijn van t; andersom hoeft dat niet het geval te zijn. Simulaties zijn daarmee een (relatief eenvoudige) methode om trace distributie–inclusie te bewijzen. Aangezien correctheidsbewijzen vaak neerkomen op het aantonen van trace distributie–inclusie, zijn simulatierelaties, net als bisimulatierelaties, nuttig voor systeemverificatie.

Dit hoofdstuk geeft een overzicht van verschillende probabilistische (bi–)simulatierelaties uit de literatuur. Vervolgens wordt een nieuw type (bi–)simulatie geïntroduceerd, de zogenaamde *vertraagde (bi–)simulatie*, waarvan er vier varianten besproken worden. Deze (bi–)simulaties hebben de eigenschap dat zij op een bepaalde manier abstraheren van interne berekeningsstappen. Dit wil zeggen dat interne berekeningsstappen, die immers voor de buitenwereld onzichtbaar zijn, altijd gebruikt mogen worden om aan te tonen dat bepaalde toestanden hetzelfde stapsgewijze gedrag hebben. We geven een alternatieve definitie van vertraagde (bi–)simulaties in termen van *normfuncties.* Een belangrijk resultaat is dat vertraagde (bi–)simulaties effiënt door een computer berekend kunnen worden. In technische termen, de besproken relaties zijn beslisbaar in polynomiale tijd en ruimte.

Hoofdstuk 6 behandelt tijdparamterische systemen. We breiden het klassieke model van Alur & Dill [AD94], de zgn *getimede automaten*, uit met timing parameters. Dit levert het *parametrische getimede automaten model (PTA model)* op. In het model van Alur & Dill kunnen tijdseisen geformuleerd worden door middel van expressies met klokvariabelen. In het PTA model kunnen deze expressies ook tijdsparameters bevatten. Gegeven een uitspraak over het model, kunnen we bekijken voor welke waarden van de parameters deze uitspraak geldt, en voor welke parameterwaarden deze niet geldt. Het doel is om de precieze parametervergelijkingen af te leiden die nodig zijn om een uitspraak waar te maken.

De belangrijkste bijdrage van dit hoofdstuk is een precieze, wiskundige semantiek van het PTA model en een methode om bij een uitspraak de parametervergelijkingen te synthetiseren die nodig zijn om de uitspraak waar te maken. Deze methode hoeft in het algemeen niet te termineren. Het is echter bewezen [AHV93] dat geen enkele methode om parametervergelijkingen te vinden in alle gevallen termineert. In technische bewoordingen: parameter synthese is onberekenbaar. Het is uiteraard wel zo dat, als de beschreven methode termineert, deze ook het correcte resultaat oplevert. Deze methode om parametervergelijkingen te synthetiseren is geïmplementeerd in een prototype, dat we gebouwd hebben uitgaande van de bestaande tool Uppaal.

Een andere belangrijke bijdrage van het hoofdstuk is de definitie van een speciale klasse van PTAs waarvoor veel parametervragen wel termineren. In deze PTAs, *ondergrens/bovengrens automaten* genoemd, mag iedere parameter òf een bovengrens òf een ondergrens op de tijdsgrenzen in de PTA afdwingen, maar niet de ene keer een bovengrens en de andere keer een ondergrens.

We sluiten het hoofdstuk af met een aantal kleine, industriële case studies die aantonen dat de prototype implementatie op voorbeelden uit de praktijk goed en relatief snel werkt.

Hoofdstuk 7 betreft een case studie. We beschrijven en analyseren het Root Contention–protocol (RCP) uit de IEEE 1394 standaard. Deze standaard wordt onder de populaire benamingen FireWire en iLink verkocht in onder andere laptops en PC's. De standaard definieert een manier om multimedia–apparatuur, zoals TV's, video recorders en PC's, met elkaar te verbinden tot een netwerk dat hen in staat stel data, bijvoorbeeld video beelden, uit te wisselen. Het Root Contention–gedeelte uit deze standaard specificeert een algoritme (i.e. een methode) om een leider te kiezen uit twee processen. Deze leider regelt het dataverkeer over het netwerk in latere fasen van het 1394 protocol. Hiertoe maakt het algoritme gebruikt van zowel random bits ("muntwerpen") als timing delays. Bovendien zijn timing parameters essentieel voor het correct functioneren van het protocol. Daarmee is dit protocol bijzonder geschikt om de praktische bruikbaarheid te onderzoeken van de analysemethoden uit eerdere hoofdstukken.

Het hoofdstuk begint met een overzicht van verscheidene analysemethoden uit de literatuur die zijn toegepast om RCP te verifiëren. Vervolgens bespreken we drie analyses van RCP die we zelf hebben uitgevoerd.

Eerst geven we een handmatige verificatie, waarbij de nadruk ligt op de probabilistische en timing aspecten van het protocol. We beschrijven het protocol en zijn specificatie in termen van het PA model uit Hoofdstuk 2. Vervolgens gebruiken we varianten van de simulatie relaties uit Hoofdstuk 5 om te bewijzen dat het protocol aan zijn specificatie voldoet. Hiertoe introduceren we een aantal hulpautomaten die tussen de implementatie en de specificatie-automaat inliggen, zodat we het verificatieproces kunnen opdelen in kleine, overzichtelijke deelstappen. Dit heeft in het bijzonder de probabilistische analyse sterk vereenvoudigd: de belangrijkste probabilistische stap wordt gemaakt in een simulatierelatie tussen twee heel kleine automaten (10 toestanden).

Vervolgens geven we een geautomatiseerde verificatie van het protocol met behulp van de model checker Uppaal. We maken een verbeterde versie van protocolmodel dat de communicatie tussen de processen op een realistischere manier weergeeft. Van dit model analyseren we vooral het timing en parametrische gedrag. We gebruiken Uppaal om (aangepaste versie van) de simulatierelaties uit het voorgaande verificatieproces en onderzoeken voor welke parameterwaarden de simulatierelaties geldig zijn. Dit is enigszins omslachtig omdat Uppaal niet ontworpen is voor dit soort taken. Toch slagen we erin de gewenste eigenschappen te onderzoeken – een uitbreiding van Uppaal met enkele eenvoudige operaties zou dit proces echter vergemakkelijken. Omdat de standaardversie van Uppaal geen parametrische analyse kan uitvoeren en omdat de parametrische versie van Uppaal nog niet beschikbaar was toen we dit onderzoek uitvoerden, checken we de parametervergelijkingen op een experimentele manier. We controleren voor een groot aantal parameterwaarden of de gewenste correctheidseigenschappen gelden. Hieruit leiden we de algemene parametervergelijkingen af. Dit geeft weliswaar geen wiskundige zekerheid over deze juistheid van deze vergelijkingen, maar naar ons idee toch betrouwbare resultaten. De parametervergelijkingen die we afleiden zijn eerder, informeel, afgeleid in een webdocument van de ontwerpers van het Root Contention–protocol.

Tenslotte gebruiken we de prototype parametrische model checker uit Hoofdstuk 6 om de parametervergelijkingen te vinden. We analyseren de modellen uit beide voorgaande verificaties. De resultaten die we met de parametrische model checker verkrijgen komen overeen met de resultaten uit de eerdere analyses.

Curriculum Vitæ

11 augustus 1972	geboren te Eindhovern
1984 – 1990	Atheneum B, Lorentz Lyceum, Eindhoven
1990 – 1997	Studie Informatica, later Wiskunde & Informatica, Katholieke Universiteit Nijmegen
1997 – 2001	Assistent in Opleiding, afdeling Informatica voor Technische Toepassingen, Katholieke Universiteit Nijmegen
2001 – nu	Postdoctoral Researcher, Department of Computer Engineering, University of California at Santa Cruz, USA

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-1

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-2

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-3

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-4

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-5

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-6

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-7

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-8

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-9

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, Univ. Leiden. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06