Pedro R. D'Argenio

Algebras and Automata for Timed and Stochastic Systems

D'Argenio, Pedro Ruben.

Algebras and Automata for Timed and Stochastic System / Pedro Ruben D'Argenio. - Doctoral Thesis, University of Twente, 1997.
ISBN 90-365-1363-4
Subject headings: automaton / process algebra / system specification / verification / real-time / performance analysis.



The research reported in this dissertation has been supported by the Netherlands Computer Science Research foundation (SION) with financial support from the Netherlands Organization for Scientific Research (NWO) within the scope of the project NWO/SION 612-33-006.

The research has been carried out under the auspicies of the Institute for Programming Research and Algorithmics (IPA) and within the context of the Centre for Telematics and Information Technology (CTIT).

IPA Dissertation Series 1999-10

CTIT PhD-Thesis Series 99-25
ISSN 1381-3617

Printed by Print Partners Ipskamp, Enschede.

ALGEBRAS AND AUTOMATA FOR TIMED AND STOCHASTIC SYSTEMS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan the Universiteit Twente,
op gezagd van de rector magnificus,
prof.dr. F.A. van Vught
volgens besluit van het College voor Promoties
in het openbaar te verdedigen op
vrijdag 5 November 1999 te 16:45 uur.

DOOR

Pedro R. D'Argenio

geboren op 14 juli 1968

te La Plata, Buenos Aires, Argentinië.

Dit proefschrift is goedgekeurd door de promotor

prof. dr. H. Brinksma

A mi mamá y mi papá.

Decir el por qué es imposible e innecesario.
Las razones son inagotables y obvias a la vez.

Acknowledgements

I will be a little unconventional and reserve the first place to thank others than my supervisor. I would like to address the first acknowledgements to four persons that made it possible for me to actually start my doctoral education.

First of all, I would like to thank Carlos Ferrari. He is a remarkable person with very high humanitarian standards. Carlos is one of the closest friends of my father and transitively, a friend of mine. He is responsible in many ways for me taking up doctoral studies, but this is just a small part of the benefit of his friendship.

Javier Blanco introduced me to the scientific ideas of correctness, semantics, formal reasoning, to mention but a few of the important keywords of the subject of this dissertation. Later, he was my bridge over the Atlantic ocean, responsible for my introduction to the Dutch scientific environment. And he has been a great companion, too.

It took me a while to graduate from the University of La Plata. It was Juan Echagüe who made me do it. That was the moment we built our friendship. He was very supportive when I made the decision to do a PhD abroad. I learned from him how to confront such a big decision not only in the academic aspects but, more important, in my personal life.

During the last three months of 1994, I had the pleasure of visiting the Formal Methods Group at the Eindhoven University of Technology. It was there where I met Jos Baeten and had the chance to work with him and his people. Afterwards, Jos signed a big recommendation letter for the position I held during the past four years at Twente.

It has been a great pleasure to have Ed Brinksma as my supervisor. He had the big challenge to confront my stubbornness, which, by the way, is not an easy task. At the same time, Ed has always been supportive and encouraging concerning my work and career and I learned a big deal from him. I wonder if I could have had a better supervisor.

I have been delighted to work with Joost-Pieter Katoen. and the stochastic automata model were born after a remark of him in the corridor. Thereafter, we have not stopped working and creating new ideas on the subject. Both Ed and Joost-Pieter reviewed the manuscript draft before it was submitted. Their suggestions and comments helped to improve considerably this dissertation, specially in the presentation aspects.

Apart from Ed and Joost-Pieter, the other members of the Graduation Committee are Frits Vaandrager, Jos Baeten, Wang Yi, Boudewijn Haverkort, Ignas Nimegeers, and the Chairman, Peter Apers. I am grateful to them for accepting the (literally) heavy task of going through my dissertation. In particular Jos and Frits made several remarks that helped me to correct mistakes that, though they did not seem very important, they could

have annoyed or misled the reader. In this respect, Mariëlle Stoelinga is also acknowledge. She spotted a mistake that would have affected the quality of the thesis.

Since the first day, I enjoyed going to work every day. The Formal Methods and Tools group at the University of Twente is simply uncommon. You can rarely see such an enjoyable group of smart and kind people fitting together so well. I would like to acknowledge the friendship of Theo Ruys and Lex Heerink. Theo went through the adventure of sharing an office with me since he came. No better fellow could I have had as an officemate. Lex had the challenging task of making this foreign guy feel comfortable in his land. He certainly did. Holger Hermanns, Rom Langerak, Pim Kars, and Dino Distefano have also been marvelous companions. Holger has also contributed with technical discussions that have influenced the contents of this dissertation. Arend Rensink gave helpful answers to my questions and provided great conversations. Jan Tretmans has been quite helpful when I just arrive to Enschede. I also would like to thank Albert, Axel, Han, Henk, Jan F., and René. And Joke Lammerink, of course, always there to take care of all these things I cannot do very well.

I have learned a lot working and coauthoring with other people. Many of them, I have already mentioned. There are still a few to whom I am also entirely grateful. They are Jan Springintveld, Frits Vaandrager, Chris Verhoef, and Sjouke Mauw.

I am also grateful to many people at the ex-TIOS and CTIT, in particular, Marloes Castañeda and Ila Reinders; colleagues and friends at the PROMISE group and at the IPA school; people at InCo, Montevideo, in particular, Alfredo Olivero and Luis Sierra; people at DoCS, Uppsala, in particular, Wang Yi and Justin Pearson. I should also acknowledge that it was in Uppsala where I started to write my thesis. I also want to thank my friends and colleagues at the Computer Science Department in La Plata, in particular, Armando De Giusti, Gabriel Baum, and Gustavo Rossi. I have also enjoyed my short visits to the University of Erlangen Nürnberg, Verimag (Grenoble), and IRISA (Rennes); I would like to acknowledge the hospitality of Ulrich Herzog, Sergio Yovine, and Sophie Pinchinat.

In this job, one has the chance to meet a lot of nice people at conferences, schools, and other kind of meetings. It is unavoidable to learn from them things that range from science to social behaviour. I would like to mention a few whose influence was deeper and I have not yet mentioned. They are Christel Baier, Dimitra Giannakopoulou, and Roberto Segala. The list grows immensely large. If you had the chance to have a glass of wine, beer, or soft drink with me during these events, you are certainly acknowledged here.

A big thanks to my friends that make my life interesting in The Netherlands. To name a few: Paula, Aljoša, Anna, Susana, Ciro, Sonia, Val, and Lyande. Y por supuesto, todos mis amigos en la Argentina que siempre me han apoyado: Gabriel, Cecilia, Mariel, Fabián, mis viejos amigos, los de la facu, y los más nuevos también.

Un "muchas gracias" con un beso grande y un fuerte abrazo a mi papá, mi mamá, Laura, Nacho, Paula, Coqui, Uli, Manu, mi tía Inés, y el resto de la familia.

Enschede, 1999
Pedro R. D'Argenio.

Contents

Part One

Introduction

Kékszakállú:
Megérketzünk. – Íme lássad:
Ez a Kékszakállú vára.
Nem tündököl, mint atyádé.
Judit, jössz-e még utánam?

(Bluebeard:
Here we are now. Now at last you see
Before you, Bluebeard's Castle.
Not a gay place like your father's.
Judith, answer. Are you coming?)

Béla Bartók. *A Kékszakállú herceg vára (Blubeard's Castle).*
Libretto by Béla Balázs. Hungary, 1918.

Chapter 1

Introduction

Computers have changed our lives forever. Three decades ago, the use of computer systems was reduced to specialised environments. The computational power of the system that put the first man on the moon was smaller than the computational power of a modern VCR. Nowadays, our interaction with some kind of computational-based device is just unavoidable. According to R.H. Bourgonjon (1998), the number of microprocessors that we encounter daily in our lives has increased from 2 in 1985 to 50 in 1995. We find these microprocessors in consumer electronics —such us TV's, VCR's, CD players—, transportation systems —including airplanes and air traffic control, cars, railway systems—, telecommunication systems, industrial plants, computer networks, medical equipment, ... You name it! The malfunction of any of these systems has a variety of consequences ranging from simply annoying to life-threatening ones. For many of such systems, it is crucial that they provide a correct and efficient service. In order to gain confidence that such devices satisfy our standards of service, it has been recognised that *formal analysis* has to be carried out as part of their development.

Formal methods provide mathematically-based languages, techniques and automated tools for specifying and verifying this kind of systems (Clarke and Wing, 1996). They have proved to be a powerful and effective tool. Examples of their application abound. Areas in which they were successfully applied include consumer electronics (e.g. Helmink et al., 1994; Bengtsson et al., 1996a; Romijn, 1999), hardware (e.g. Barret, 1989; Boyer and Yu, 1996; O'Leary et al., 1999), automobile industry (e.g. Ruys and Langerak, 1997; Lindahl et al., 1998), aerospace industry (e.g. Eastbrook et al., 1997; Crow and Di Vito, 1998), and weather catastrophe prevention (e.g. Kars, 1998).

Although originally focussed on software specification, the advance in formal methods is such that, nowadays, they provide support for every stage of the life cycle of system engineering, including testing (e.g. Tretmans, 1992; Heerink, 1998; Springintveld et al., 1999), and maintenance (e.g. Bowen et al., 1993; van Zuylen, 1993).

In this dissertation we concentrate on the study of formal methods for the *design* phase of the life cycle of system engineering.

In the formal analysis of systems, there are several aspects that should be taken into account:

- *Control.* The basic aspects of a design is to obtain a correct control flow, i.e., a suitable causal dependency between events. The temporal order in which events happen is crucial for the correctness of a system. For instance, it is evident that a buffer must first accept an input in order to produce an output. The control aspects are essential in the analysis of systems and cannot be overlooked.

- *Data.* Many systems strongly rely on data manipulation. Data usually determines the control flow of the system. Sometimes, data consistency is in the main interest of the system under consideration (e.g. databases). Some other systems, such as many protocols, do not rely heavily on data and can be abstracted in the formal analysis.

- *Time.* A timed system is a system whose behaviour is constrained by requirements concerning the occurrence of events with respect to (real) time. These timing conditions may speak about the acceptable performance of the system or a deadline that should be met.

In this thesis we study and develop formalisms for the design and analysis of systems in which *time* is an important factor.

1.1 Real-Time Systems

Most of the systems we encounter in our daily life require *real-time* interaction. For example, when we withdraw money from a cash dispenser, we expect the machine to react within a reasonable amount of time. But our patience has limits and if the transaction delays more than expected, we feel annoyed. For some other systems, it has far more critical consequences if they do not react on time. For instance, a gate controller in a rail-road crossing must react within a very specific period of time when a train is approaching, and close the gate. A small delay could cause a collision with fatal consequences.

In these two examples, we can clearly differentiate between two classes of real-time constraints: those that require that a system *must* react in time, and those that it *should* react in time but occasionally may not. If a system belongs to the first category it is referred to as a *hard real-time* system, if it belongs to the second one, as *soft real-time* system.

The analysis of hard real-time systems requires a full exploration of its behaviour searching for undesirable situations. The violation of a timing requirement is unacceptable. Notice the absolute characteristics of the judgment. There are only two options: a system is correct or not. For instance, the rail-road crossing system must satisfy the safety property stating that it is always the case that, if a train is in the crossing, then the gate is closed.

The act of formally analysing whether a system satisfies a property or not, is known as *verification*.

In contrast, the analysis of soft real-time systems requires a parameter of adequacy. For instance, we find it reasonable that at least 95% of the times we withdraw money, the cash dispenser does not make us wait. The soft real-time requirements are typically concerned with the *performance* characteristics of systems and are usually related to stochastic aspects of various forms of time delay, such as, for example, mean and variance of a message transfer delay, service waiting times, etc. In addition, a stochastic study of the system also allows for *reliability* analysis, such as, average time during which the system operates correctly. Nevertheless, not all the parameters are soft. Some requirements still must be satisfied. After all the cash dispenser should eventually react by delivering our withdrawal or reporting that it is out of order.

> The contribution of this dissertation is divided in two parts. The first part focuses on the development of a formal language for the specification of *hard* real-time systems built on top of existing theory. In the second part, we develop a formal framework for the specification and analysis of *soft* real-time systems.

Since the characteristics of the information we want to collect from soft real-time system is stochastic, we also refer to these systems as *stochastic systems*. For simplicity, we refer to hard real-time systems as *timed systems*.

1.2 Algebras and Automata

Probably the simplest way to represent behaviour is by means of *automata*. An automaton is a graph containing nodes and directed, labelled edges. Nodes represent the possible states of a system and edges (or transitions) represent activity by joining two nodes: one being the source state, that is, the state before "executing" the transition, and the other, the target state, which is the state reached after executing the transition. A simple and self-explanatory example is given in Figure 1.1.

Automata have been extended in many forms and used for many purposes, including testing (e.g. Chow, 1978; Tretmans, 1992; Heerink, 1998), programming language semantics (e.g. Plotkin, 1981; Winskel, 1993), model checking (e.g. Clarke and Emerson, 1981; Queille

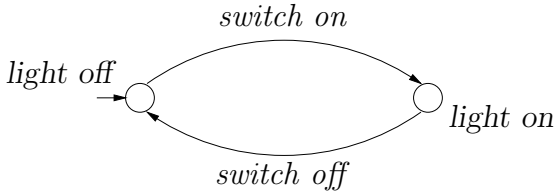


Figure 1.1: An example of an automaton

and Sifakis, 1982), and many other forms of verification. All these techniques provide a solid mathematical framework for the validation of systems. For instance, in model checking, the requirements are expressed by formulas in an appropriate logic and they are checked to be satisfied by the automaton that models the possible behaviours of a system. Such a satisfaction relation is a well defined mathematical relation. Alternatively, verification can be carried in terms of semantic relations over automata. That is, both the requirements and the system are specified by automata, the first one being usually simpler, and they are checked to be related by an equivalence or a preorder relation. Such a relation represents the "implementation" or "conformance" relation. Examples include language inclusion (Hopcroft and Ullman, 1979), failure semantics (Brookes et al., 1984), and *bisimulation* (Milner, 1980).

The process of specification is the act of writing things down precisely. The main benefit in doing so is intangible—gaining deeper understanding of the system being specified. It is through this specification process that developers uncover design flaws, inconsistencies, ambiguities, and incompleteness (Clarke and Wing, 1996). Although automata give a nice graphic representation of the system we want to describe, this same characteristic turns specifications unwieldy as the model gets larger and more detailed.

In order to specify complex systems we need a *structured* approach, a systematic methodology that allows us to build large systems from the composition of smaller ones. *Process algebras* were conceived for the hierarchical specification of systems. A process algebra is an algebra as it is understood in mathematics that is intended to describe processes. Each element of the process algebra represents the behaviour of a system in the same way as an automaton does. In addition, a process algebra provides operations that allow to compose systems in order to obtain more complex systems. Typical examples of these operations are the sequential composition and the parallel composition. For instance, if p and q represent two systems, we can compose them in parallel to obtain a larger system $p \,||\, q$. Notice that the behaviour modelled by this operation can be complex, but this is hidden behind the simple expression $p \,||\, q$.

As any algebra, a process algebra satisfies axioms or laws. The interest of having an axiomatisation for a process algebra is two-fold. First, the concept of algebra is fundamental in mathematics. Therefore, the given axiom system will help to understand the discussed process algebra and the concepts it involves. Second, the analysis and verification of systems described using the process algebra can be partially or completely carried out by mathematical proofs using the equational theory.

The works of Hoare (1978) and Milner (1980) marked the origin of the process-algebraic approach that led to a new broad research area.

In this thesis we introduce and study *process algebras* and *automata* for the *design* and *analysis* of *hard* and *soft real-time* systems.

Algebras and Automata for Timed Systems

Several models for the analysis and specification of timed systems have been devised, including timed Petri-nets (Sifakis, 1977), duration calculus (Chaochen et al., 1991), and a variety of extensions of automata with time information (e.g. Lynch and Vaandrager, 1996; Segala et al., 1998; Lynch et al., 1999).

A model that has been accepted with a large success is *timed automata* (Alur and Dill, 1990, 1994; Henzinger et al., 1994). Inherently, a timed system induces an infinite behaviour due to the need of representing dense time. Timed automata propose a symbolic way to describe this infinite behaviour. In so doing, it enables a full state space exploration by traversing a "symbolically finite" state space. This is mainly the reason for the popularity of this model.

Timed automata have served as a model for many model checking algorithms (e.g Alur et al., 1993; Henzinger et al., 1994; Yovine, 1993; Yi et al., 1994; Olivero, 1994; Pettersson, 1999) and tools that implement these algorithms such as KRONOS (Daws et al., 1996; Bozga et al., 1998), UPPAAL (Bengtsson et al., 1996b; Larsen et al., 1997) and HYTECH (Henzinger et al., 1995). These tools were used in a diversity of case studies (e.g. Ho and Wong-Toi, 1995; Daws and Yovine, 1995; Bengtsson et al., 1996a; Maler and Yovine, 1996; D'Argenio et al., 1997b; Lindahl et al., 1998). Timed automata were also extended to study hybrid systems (Alur et al., 1995) and served as a model for a theory of timed test derivation (Springintveld et al., 1999).

A process algebraic approach for timed systems has also been pursued. Originally, time was included in a discrete fashion (e.g. Groote, 1991; Moller and Tofts, 1990; Hansson, 1994; Nicollin and Sifakis, 1994; Baeten and Bergstra, 1996; Vereijken, 1997), that is, time is represented as a "tick" action that describes the passage of a single time unit. These process algebras are adequate to model digital systems but they cannot describe real-time systems in a natural way. For this reason, *dense time process algebras* were developed (e.g. Yi, 1990, 1991; Baeten and Bergstra, 1991; Davies et al., 1992; Klusener, 1993; Leduc and Léonard, 1994).

Some of these process algebras have been formally related to timed automata (e.g Nicollin et al., 1992; Fokkink, 1993; Bradley et al., 1995, see also Section 5.8). However, these relations only provide a semantic connection and none of them is complete in the sense that such a connection is bijective. Languages that completely represent timed automata have been defined (Lynch and Vaandrager, 1996; Alur and Henzinger, 1994; Yi et al., 1994; Pettersson, 1999) but none of them provide an algebraic framework. To our knowledge no axiomatic theory of timed automata has been devised until now.

Algebras and Automata for Stochastic Systems

The analysis of stochastic systems has received a lot of attention but, traditionally, outside the community of formal methods. Long ago, mathematicians defined models for the analysis of stochastic processes. These models, which include the so-called continuous time Markov chains, were used to analyse the performance of systems. But systems started to

become more complex and there was a need for a more sophisticated notation to specify performance models. This led to the definitions of models such as queueing networks (Kleinrock, 1975; Harrison and Patel, 1992) and a diversity of stochastic Petri-nets (Ajmone Marsan et al., 1995).

Models based on continuous time Markov chains represent the timing of events as a stochastic variable distributed only according to an exponential distribution. This restriction is the key for the vast analytical and numerical theory that supports continuous time Markov chains. However, restricting to exponential distributions is often not realistic and may lead to results that are not sufficiently accurate. For instance, in the analysis of high-speed communication systems or multi-media applications the correlation between successive packet arrivals tends to have a constant length; therefore, the usual Poisson arrivals and exponential packet lengths are no longer valid assumptions (Brinksma et al., 1995). More general models, such as *generalised semi-Markov processes* (Whitt, 1980; Glynn, 1989, see also Cassandras 1993; Shedler 1993), allow for the description of timings that may depend on any distribution.

Unfortunately, none of these models provides a suitable framework for the *composition* of distributed and communicating systems.

The formal methods community tackled the description and analysis of stochastic systems by means of so-called probabilistic process algebras (e.g Giacalone et al., 1990; Hansson, 1994; van Glabbeek et al., 1995; Baeten et al., 1995; Andova, 1999; Baier, 1999), although initially it was intended for verification purposes rather than performance analysis. In addition, they only deal with discrete probability distributions, and most of them do not include a notion of time.

It was not until the seminal work of Ulrich Herzog (1990) that process algebras and performance models were combined. *Stochastic process algebras* were thus introduced.

Taking advantage of the analytical framework provided by continuous time Markov chains, so-called Markovian process algebras were devised. They include TIPP (Hermanns and Rettelbach, 1994), PEPA (Hillston, 1996), EMPA (Bernardo and Gorrieri, 1998; Bernardo, 1999), MPA (Buchholz, 1994) and IMC (Hermanns and Rettelbach, 1996; Hermanns, 1998). The general approach, including any continuous distribution function, has also been addressed (Götz et al., 1993; Strulo, 1993; Harrison and Strulo, 1995; Brinksma et al., 1995; Katoen, 1996; Priami, 1996; Bravetti et al., 1998).

Each approach deals with the parallel composition in a different manner. When no interaction is involved, parallel composition can be easily defined in Markovian process algebras. If arbitrary distributions are allowed, this definition proved to be more complicated yielding complex or infinite semantic objects.

Synchronisation leads to more complicated decisions (see Hillston, 1994). Not all the process algebras provide a symmetric interaction. For instance, EMPA requires that at most one of the synchronising components is time dependent. The others must be "passive". The kind of synchronisation we are interested in is what Jane Hillston (1994) calls *patient communication*. A patient communication takes place when all the components that intend to synchronise, are ready to do it. This approach is adopted by Harrison and

Strulo (1995); Brinksma et al. (1995); Hermanns (1998). Hillston's PEPA also uses this approach but needs to approximate the distribution of the time in order to stay in the domain of exponential distributions. Notice that patient communication can model the aforementioned asymmetric synchronisation: a passive process is always ready to synchronise.

Another issue that arises as a consequence of considering parallel composition in interleaving models (such as automata-based models) is *non-determinism*. Only Hermanns' Interactive Markov Chains (IMC) and Harrison and Strulo's process algebra deal with non-determinism in a satisfactory way[1]. The rest of the process algebras solve non-determinism probabilistically, either by the so-called *race condition*, or by explicitly determining the branching probabilities.

1.3 Our Contribution

This dissertation consists of two parts. In the first part of the thesis we concentrate on timed systems. Timed automata is a well-established model for the analysis of hard real-time systems that has been used successfully on many occasions. However no algebraic theory of timed automata has been discussed so far. The main contribution of this part is thus a *process algebra for timed automata* which we call ♡ (read *hearts*). Its syntax contains the same "ingredients" as the timed automata model. Moreover, we provide an axiomatic theory that allows us to manipulate algebraically timed automata.

In the second part we concentrate on stochastic systems. Although many models have been successfully used for the analysis of performance and reliability of soft real-time systems, none of them provide a general and suitable model to represent systems compositionally. Based on generalised semi-Markov processes and timed automata, we define a new model which we call *stochastic automata*. The stochastic automata model is a general stochastic model that provides an adequate framework for composition.

Several stochastic process algebras have been introduced in the last decade. The most successful ones restrict to the so-called Markovian case. General approaches to stochastic process algebras followed a less successful path, mainly due to the lack of a suitable model to represent this general models in a compositional fashion.

Using stochastic automata as the underlying semantic model we define ♤ (read *spades*), a *process algebra for stochastic automata*. This stochastic process algebra considers arbitrary distributions and non-determinism. A particular characteristic is that parallel composition and synchronisation can be easily defined in the model, and it can be algebraically decomposed in terms of more primitive operations according to the so-called *expansion law*. We remark that this last property is not present in any of the other existing general stochastic process algebras.

In order to show the feasibility of the analysis of systems specified in ♤, we report on

[1]EMPA considers non-determinism as well, but it was recently shown that it may violate compositional consistency (Bernardo, 1999).

algorithms and prototypical tools to study correctness as well as performance and reliability. The tools were used to analyse several case studies which are reported as well.

1.4 Outline of the Dissertation

This thesis is divided in four parts. The first one is intended to report the introductory concepts. It includes this chapter and Chapter 2 which contains the preliminaries. There, we introduce and discuss intuition and foundations of many aspects which are traditional to concurrency and process theory such as transition systems, bisimulation, process algebras, and properties of each one of them. Although the reader is assumed to have a background in concurrency theory, Chapter 2 intends to recall all those concepts that will be used and redefined in the context of timed and stochastic systems. Thus, intuition or the rationale behind traditional concepts are not repeated in the other chapters except when necessary.

The second and the third part discuss the timed and the stochastic theories respectively. They have been structured as follows

1. concrete semantic model,
2. symbolic semantic model,
3. syntax and semantics of the process algebra,
4. its equational theory, and
5. applications.

This structure is somehow present already in Chapter 2, although there, the concrete and symbolic model are the same and the last part has been omitted.

The contributions on timed systems are reported in the second part. It includes the following chapters:

Chapter 3. We give a formalisation of timed transition systems. A notion of bisimulation for this model is defined.

Chapter 4. The timed automata model is revisited. We adapt it to our framework and give semantics in terms of timed transition systems. Several notions of equivalences are defined and related.

Chapter 5. ♡, the process algebra for timed systems, is introduced. Its semantics is defined in terms of timed automata. Equivalences and congruences are discussed.

Chapter 6. Several equational theories for ♡ are defined. We study their soundness and completeness with respect to the equivalences defined in Chapter 4 and use it to derive an expansion law.

Chapter 7. We discuss several applications of ♡ including specification and analysis of timed systems, and algebraic reasoning of timed automata.

Some preliminary results of this first part have been published in (D'Argenio and Brinksma, 1996a,b; D'Argenio, 1997a,b).

In the third part we report on our contributions to the specification and analysis of stochastic systems. It includes the following chapters:

Chapter 8. We introduce probability transition systems that allow for arbitrary probability distributions and define a probabilistic bisimulation on this model.

Chapter 9. Stochastic automata are introduced. We discuss different semantics in terms of probabilistic transition systems. Equivalences at the symbolic level are defined and related. We formally compare stochastic automata with generalised semi-Markov processes.

Chapter 10. We introduce ♤, a process algebra for the description of stochastic systems, and define its semantics in terms of stochastic automata. Congruence relations are discussed.

Chapter 11. We study several axiomatisations and laws for ♤ and prove soundness and completeness with respect to the congruences reported in Chapter 10. An expansion law is derived from the equational theory.

Chapter 12. We define the theoretical background of algorithms for the quantitative and qualitative analysis of ♤ specifications. We report on prototypical tools that implement those algorithms and discuss their use in a variety of case studies.

Some preliminary results of this part have appeared in the following articles: (D'Argenio et al., 1997a, 1998a,b,c, 1999b).

In Chapter 13 we give some concluding remarks and discuss possible directions for further research. There is also a fourth part that includes appendices containing the most involved technical details. Appendices A, B, and C report the proofs of theorems appearing in the second part, while Appendices E, F, and G contain the proofs of the third part. In Appendix D we recall some basic concepts of probability theory.

An overview of the chapter dependencies of this dissertation is given in Figure 1.2. Part two and three can be read in an independent fashion. Occasionally, part three will make a reference to the second part for the sake of not being redundant. This will be explicitly indicated.

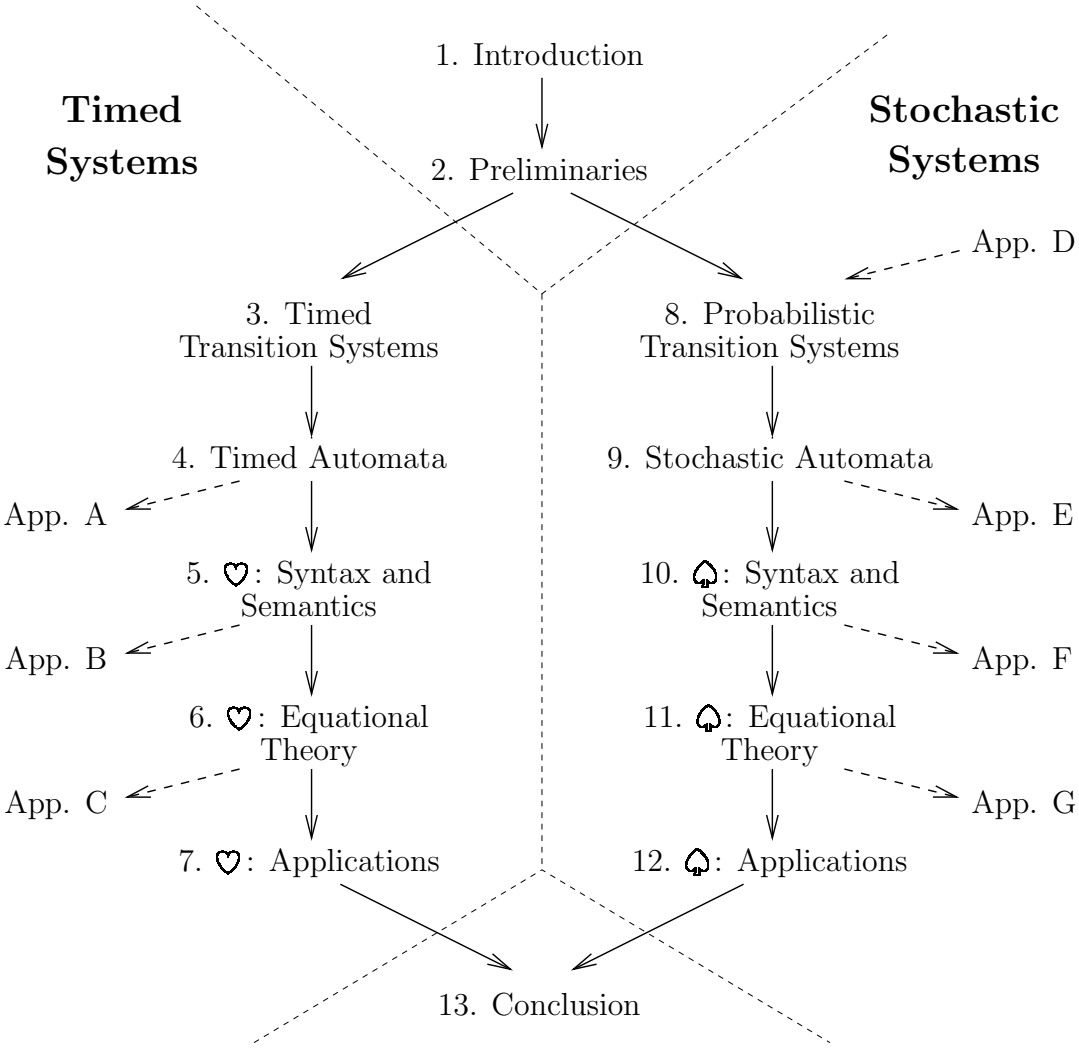


Figure 1.2: Schematic view of the dissertation

Chapter 2

Preliminaries

In this chapter we recall the different methods and techniques used to define and study process algebras. This introductory chapter is based on the extensive literature in process algebra that has been produced along the last two decades. Main references are the text books (Hoare, 1985; Milner, 1989; Baeten and Weijland, 1990), and the articles (Bergstra and Klop, 1985; Bolognesi and Brinksma, 1989; Baeten and Verhoef, 1995).

2.1 Transition Systems

Automata, in their many different variants, are probably the most used models to represent the behaviour of systems. An automaton is a graph containing nodes and edges. Nodes represent the possible states of a system. Edges —called transitions in the context of system behaviour— represent activity by joining two nodes: one being the source state, that is, the state before "executing" the transition, and the other, the target state, which is the state reached after executing the transition.

Automata are usually decorated in several ways, giving rise to many different subclasses, each one intended to represent different kinds of behaviours. Throughout this thesis, for instance, we will define 5 classes of automata. To begin with, we recall the definition of labelled transition systems, which are automata whose edges are labelled with so-called actions.

Definition 2.1. A *(labelled) transition system* is a structure $TS = (\Sigma, \mathcal{L}, \longrightarrow)$ where:

- Σ is a set of *states* which we denote by $\sigma, \sigma', \sigma_0, \sigma_1, \ldots$;
- $\mathcal{L}$ is a set of *labels* which we denote by $\ell, \ell', \ell_1, \ldots$; and
- $\longrightarrow$ is a set of *transitions*, such that $\longrightarrow \subseteq \Sigma \times \mathcal{L} \times \Sigma$.

To simplify notation, we write $\sigma \xrightarrow{\ell} \sigma'$ instead of $(\sigma, \ell, \sigma') \in \longrightarrow$. We also write in general $\sigma \xrightarrow{\ell}$ if there exists $\sigma' \in \Sigma$ such that $\sigma \xrightarrow{\ell} \sigma'$, and if this is not the case, we write $\sigma \not\xrightarrow{\ell}$.

On many occasions it will be convenient to distinguish an initial state $\sigma_0 \in \Sigma$. In this case we called the structure (TS, σ_0) a *rooted (labelled) transition system.* □

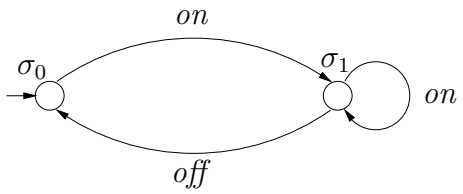


Figure 2.1: A transition system representing a switch in a stairway

Example 2.2. Figure 2.1 shows a rooted transition system describing the behaviour of an automatic switch that controls a light as it can be found in a staircase or corridor in a hotel. Circles represent states. Their names are written beside them; in this case they are σ_0 and σ_1. The initial state σ_0 is indicated with a small incoming arrow. Arrows represent transitions. Their labels are given next to them; in this case they are *on* and *off*.

The behaviour of the system can be described as follows. The state of the system in which the light is off is represented by σ_0, and σ_1 represents the state in which the light is on. People arrive at the staircase and turn *on* the switch ($\sigma_0 \xrightarrow{on} \sigma_1$). After a while the switch turns itself *off* ($\sigma_1 \xrightarrow{off} \sigma_0$). It could be the case that when the light is on, people press the switch anyway to reset the timer that controls the switching off. In such a case the light stays on ($\sigma_1 \xrightarrow{on} \sigma_1$).

We will use this simple switch as a running example. The intention is to show how it can be described in the different models we will discuss along this thesis. In this way we can compare the accuracy with which we can model the switch system, once we have added time and probability features. □

When studying the behaviour of systems it is important to decide whether two systems described in terms of transition systems behave in the same manner. There are many reasons why it is important to equate systems. For instance, checking that a particular system completely satisfies its specification can amount to checking equivalence of the transition system modelling the system and the transition system describing the specification. Another reason is that whenever two systems show equivalent behaviour, one of them could be replaced by the other as part of a bigger system (up to some additional considerations).

Many different ways to equate transitions systems have been defined, most of them during the last two decades. For a broad overview, the reader is referred to (van Glabbeek, 1990) and (van Glabbeek, 1993). In this thesis we will concentrate on several notions of bisimulation. A bisimulation (Milner, 1980; Park, 1981) can be seen as a game played by two players —each one of them following the rules given by a (rooted) transition system— in which one player proposes to perform a move —a transition— and the other is obliged to imitate it. At any moment of the game the role of proposing player and imitating player can be interchanged. Two players are bisimilar if none of them is capable to find a strategy to defeat his opponent.

Definition 2.3. Let $TS = (\Sigma, \mathcal{L}, \longrightarrow)$ be a transition system. A relation $R \subseteq \Sigma \times \Sigma$ is

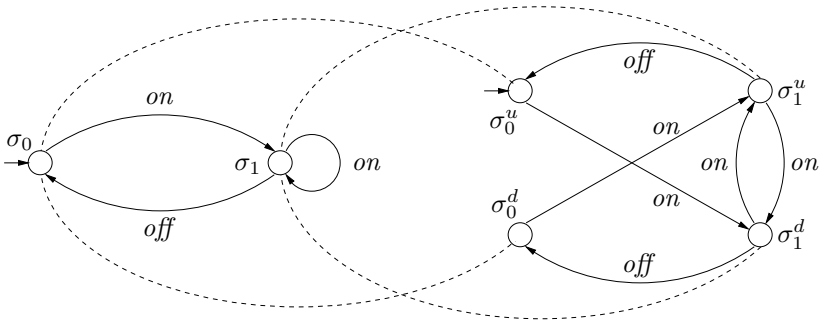


Figure 2.2: Two bisimilar transition systems

a *bisimulation relation* if R is symmetric and for all $\langle \sigma_1, \sigma_2 \rangle \in R$ and for all $\ell \in \mathcal{L}$ the following transfer property holds:

$$\text{if } \sigma_1 \xrightarrow{\ell} \sigma_1' \text{ then } \sigma_2 \xrightarrow{\ell} \sigma_2' \text{ for some } \sigma_2' \in \Sigma \text{ such that } \langle \sigma_1', \sigma_2' \rangle \in R.$$

We say that two states σ_1 and σ_2 are *bisimilar*, notation $\sigma_1 \sim \sigma_2$, if there is a bisimulation R that relates them, i.e., $\langle \sigma_1, \sigma_2 \rangle \in R$.

Two rooted transition systems (TS_1, σ_1) and (TS_2, σ_2) are *bisimilar*, if their initial states σ_1 and σ_2 are bisimilar in the (disjoint) union of TS_1 and TS_2. □

It is not difficult to prove that $\sim$ is the largest bisimulation relation and that it is an equivalence relation. The curious reader is referred to (Milner, 1989).

Notation. Since we require that a bisimulation should be a symmetric relation, in many occasions we will give a relation and claim that its symmetric closure —i.e., the smallest symmetric relation that contains it— is a bisimulation. For such a reason, given a relation R over a certain domain, we use the notation R^{s} to refer to the symmetric closure of the relation R. Similarly we let R^* denote the reflexive-transitive closure of R, that is, the smallest reflexive and transitive relation containing R. Finally, R^{eq} denotes the smallest equivalence relation containing R. We write Id for the identity relation. □

Example 2.4. While in Example 2.2 the switch was intended to be a simple button, the transition system on the right of Figure 2.2 describes a switch with two positions: up and down. Thus, σ_0^u is the state in which the light is off and the switch is in upper position, σ_1^u represents the state in which the light is on and the switch is up, and σ_0^d and σ_1^d are the respective on and off states when the switch is down.

It is not difficult to check that the relation

$$R = \{\langle \sigma_0, \sigma_0^u \rangle, \langle \sigma_0, \sigma_0^d \rangle, \langle \sigma_1, \sigma_1^u \rangle, \langle \sigma_1, \sigma_1^d \rangle\}^{\mathsf{s}}$$

is a bisimulation. Relation R is depicted with dotted lines in Figure 2.2. □

As we have just done, to prove that two systems are bisimilar we have to exhibit a bisimulation. But usually this relation has information that we might find somehow redundant. So, we would like to give instead a smaller relation which omits that information. Hence, we define the notion of bisimulation up to $\sim$. Notice that it slightly differ from the traditional definition of Milner (1989).

Definition 2.5. Let $TS = (\Sigma, \mathcal{L}, \longrightarrow)$ be a transition system. A relation $R \subseteq \Sigma \times \Sigma$ is a *bisimulation up to* $\sim$ if R is symmetric and for all $\langle \sigma_1, \sigma_2 \rangle \in R$ and $\ell \in \mathcal{L}$ the following transfer property holds:

$$\text{if } \sigma_1 \xrightarrow{\ell} \sigma_1' \text{ then } \sigma_2 \xrightarrow{\ell} \sigma_2' \text{ for some } \sigma_2' \in \Sigma \text{ such that } \langle \sigma_1', \sigma_2' \rangle \in (\sim \cup R)^*. \qquad \square$$

In fact, finding a bisimulation up to $\sim$ suffices to show bisimilarity. This is stated by the following theorem.

Theorem 2.6. *Let R be a bisimulation up to $\sim$. Then $R \subseteq (\sim \cup R)^* \subseteq \sim$.*

The proof of this theorem is analogous to and simpler than the proof of Theorem 3.6.

In many cases, one would like to have completely automatic machinery to analyse the behaviour described by transition systems. To do so, algorithms for checking equivalences such as bisimulation have been devised (e.g. Paige and Tarjan, 1987; Kanellakis and Smolka, 1990; Fernandez, 1989), as well as algorithms to check if a transition system satisfies a given property expressed in a certain modal logic (e.g. Clarke et al., 1986; Godefroid and Wolper, 1991; Cleaveland, 1990; Wolper et al., 1983), and algorithms to derive complete test suites to test system implementations (e.g. Chow, 1978), among other kind of automatic analysis. All these algorithms do not operate over arbitrary transition systems. Since all of them require a complete search of the state space and transitions, an essential requirement is that transition systems are finite and hence termination of every algorithm is ensured.

Other times, one may admit that a complete automatic analysis of the transition system is either impossible or too ambitious. In this case, a usual technique is simulation (e.g. Eertink, 1995). Simulation allows to study different possible executions of the systems and hence only the current state of the execution that is being traced is relevant. In such a case there is no need to require finiteness of the transition system. It is only important that from each state there is a finite number of outgoing transitions from which we can select one to execute. This kind of transition systems is called finitely branching[1].

Definition 2.7. Let $TS = (\Sigma, \mathcal{L}, \longrightarrow)$ be a transition system. TS is *finitely branching* if for all $\sigma \in \Sigma$ the set

$$\{\sigma \xrightarrow{\ell} \sigma' \mid \ell \in \mathcal{L} \wedge \sigma' \in \Sigma\}$$

is finite. We say that TS is *finite* if in addition it satisfies the property that the set of states Σ is finite. $\square$

[1] One may argue that for simulation purposes one still can have an infinite branching transition system if it is appropriately coded in a symbolic way. But in this case the obtained "symbolic" transition system needs to be finitely branching. In fact, in the rest of the thesis we discuss two classes of automata that abstractly model (uncountably!) infinitely branching transition systems.

2.2 ♧ – A Common Language for Untimed Behaviours

Process algebras were introduced at the end of the 70's. The works of Tony Hoare (1978, 1985), on CSP, and Robin Milner (1980, 1989), on CCS, mark this point. During the 80's, process algebras grew to end up as quite a mature method to model and analyse concurrent and communicating systems. Apart from CSP and CCS two other process algebras reached a distinct place in formal methods: ACP (Bergstra and Klop, 1985; Baeten and Weijland, 1990) and LOTOS (Brinksma, 1988; Bolognesi and Brinksma, 1989). In this section we discuss a process algebra for which we have selected the most relevant ingredients present in the algebras enumerated before. We call it *clubs*, standing for *Common Language for Untimed Behaviours*, and denote it by ♧.

The basic ingredient in process algebras is the *action name*. An action name —or simply, action— describes an activity that is performed by a system. This activity is assumed to be atomic, that is, it cannot be interrupted by any other activity when it is executed. Moreover, its execution does not consume any time. This concept is fundamental for the so-called interleaving models which are at the basis of this thesis.

Assume a given process p that represents the behaviour of a system. An action a can be *prefixed* to p, thus becoming the process $a;p$, which first performs the action a and then shows the same behaviour as p. The most primitive process is the process $\mathbf{0}$ (read *nil*) which does not show any behaviour; in other words, it does not do anything. In many cases a system offers the possibility to select one out of two possible behaviours p and q to be executed. This operation is called *(non-deterministic) choice* or *sum* and denoted by $p+q$. The previously described operations are called *basic operations* since all other operations can be eliminated in favour of these.

The key operation for modelling concurrent systems, as it is the intention of process algebras, is the *parallel composition* or *merge*. $p \,||_A\, q$ describes a process that executes p and q in parallel forcing the synchronisation of actions in the set A of action names. Either p or q can execute actions which are not in A independently. These actions interleave with the actions (not in A) of the other process. We will also consider two special notions of parallel composition. The *left merge* $p \,\lfloor\!\lfloor_A\, q$ behaves exactly in the same way that $p \,||_A\, q$ does with the only exception that the first action (not in A) should be performed independently by process p. Similarly, the *communication merge* $p \,|_A\, q$ is a process that behaves like $p \,||_A\, q$ with the restriction that the first action (in A) should be a synchronisation action. The left and communication merge are not really intended for the modelling of systems. Nevertheless they are essential to define a finite equational theory (Moller, 1990) as we will do in the next section.

The last operation we consider is *(action) renaming*. Given a function f that takes an action name and changes it into another action name, process $p[f]$ behaves like process p but with the actions changed according to f.

Apart from the just mentioned operations, we allow for *recursive definitions*. To do so, we consider *process variables* which are variables that can take processes as values. So, suppose p is a process that contains the process variable X. The *recursive equation* $X = p$ defines the value of X as the one which solves that equation. In general, we choose the

unique solution to be the the minimal one following the approach of Milner (1989) instead of following the more general approach of Baeten and Weijland (1990) in which more than one possible solution is considered. Thus, equation $X = a;X$ defines in X a process that performs infinitely many a's, and $Y = a;\mathbf{0}+Y$ defines in Y the process that only performs one a. We can also define process variables by means of systems of recursive equations called *recursive specifications*, in which one process variable may be defined in terms of a process containing many other variables. Summarising, we define ♧ as follows.

Definition 2.8. The language ♧ is defined according to the following grammar

$$p \quad ::= \quad \mathbf{0} \mid a;p \mid p+p \mid p \,||_A\, p \mid p \,\between_A\, p \mid p \,|_A\, p \mid p[f] \mid X$$

where a is an *action name* belonging to the set $\mathcal{A}$ of action names, $A \subseteq \mathcal{A}$ is a *synchronisation set*, $f : \mathcal{A} \to \mathcal{A}$ is a *renaming function*, and X is a *process variable* belonging to the set $\mathcal{V}$ of process variables.

A process variable is defined by an equation of the form $X = p$ where p is a ♧ term. These kind of equations are called *recursive equations*. A *recursive specification* is a set E of recursive equations. Sometimes we will consider a distinguished process variable in the recursive specification E called *root*. □

In general we will assume that prefixing and renaming have precedence over any other operator, and that + is the operation with lowest precedence. Thus, $a;\mathbf{0} + X[f] \,||_A\, b;Y$ represents the term $(a;\mathbf{0}) + ((X[f]) \,||_A\, (b;Y))$. Notice that we have not specified the precedence in some cases and hence some terms without parentheses, as for instance $a;X[f]$ or $a;\mathbf{0} \,||_A\, X \,\between_B\, b;Y$, are ambiguous. We will always put parentheses to solve ambiguity in those cases.

Example 2.9. In Example 2.2 we have described a switch in a stairway that can be switched on by people who arrive to the stair and it automatically switches off after a fixed period of time. So we have two processes in the system. The first is a simple process that describes the arrival of people:

$$\mathit{Arrival} \;=\; \mathit{on};\mathit{Arrival}$$

The second is the switch itself:

$$\begin{aligned} \mathit{SwitchOff} &= \mathit{on};\mathit{SwitchOn} \\ \mathit{SwitchOn} &= \mathit{on};\mathit{SwitchOn} \\ &\quad + \mathit{off};\mathit{SwitchOff} \end{aligned}$$

The complete system is described by the interaction of the switch and the people that arrive. Since people want to turn on the switch, both processes should synchronise on action *on*:

$$\mathit{System} \;=\; \mathit{Arrival} \,||_{on}\, \mathit{SwitchOff}$$

To simplify notation, we write $||_a$ instead of $||_{\{a\}}$ whenever the synchronisation set contains only one action. □

$$a; p \xrightarrow{a} p \qquad \frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \qquad \frac{p \xrightarrow{a} p'}{q + p \xrightarrow{a} p'}$$

$$\frac{p \xrightarrow{a} p'}{p \,||_A\, q \xrightarrow{a} p' \,||_A\, q}\, a \notin A \qquad \frac{p \xrightarrow{a} p'}{q \,||_A\, p \xrightarrow{a} q \,||_A\, p'}\, a \notin A \qquad \frac{p \xrightarrow{a} p' \quad q \xrightarrow{a} q'}{p \,||_A\, q \xrightarrow{a} p' \,||_A\, q'}\, a \in A$$

$$\frac{p \xrightarrow{a} p'}{p \,|\!|\!|_A\, q \xrightarrow{a} p' \,||_A\, q}\, a \notin A \qquad \frac{p \xrightarrow{a} p' \quad q \xrightarrow{a} q'}{p \,|_A\, q \xrightarrow{a} p' \,||_A\, q'}\, a \in A$$

$$\frac{p \xrightarrow{a} p'}{p[f] \xrightarrow{f(a)} p'[f]} \qquad \frac{p \xrightarrow{a} p'}{X \xrightarrow{a} p'}\, X = p$$

Table 2.1: Transition rules for ♧

The behaviour of ♧ terms is given in terms of transition systems. The states of the transition system are ♧ terms. The transition relation is defined following Plotkin's style (Plotkin, 1981). In the so-called *structured operational semantics*, the transitions that a process p can possibly perform are defined in terms of the transitions performed by the subprocesses out of which p is built.

Rules in Table 2.1 capture the behaviour of ♧ terms as it was previously described. Process $\mathbf{0}$ does not have any outgoing transition meaning that it cannot do anything. $a; p$ performs an action a and then behaves like p. $p + q$ can perform any transition p or q can perform, and continues with the behaviour of the chosen process. $p \,||_A\, q$ can independently perform any transition p or q can perform if this transition does not require synchronisation, i.e., if it is not labelled with an action in A. Synchronisation actions can be performed by $p \,||_A\, q$ if both p and q can perform the same synchronisation action. $p \,|\!|\!|_A\, q$ can only perform transitions labelled with actions not in A whenever p can perform this transition. It continues by executing the "remaining" behaviour of p in parallel with q. $p \,|_A\, q$ can only perform transitions labelled with actions in A whenever p and q have an outgoing transition labelled with the same synchronisation action. Afterwards, the process behaves as a normal parallel composition. For each outgoing transition of p labelled with a, $p[f]$ has an outgoing transition renamed by $f(a)$. Finally, a variable X, defined according to the equation $X = p$, has the same outgoing transitions as its defining process p. Formally, we have:

Definition 2.10. The semantics of ♧ is defined according to the (labelled) transition system $TS(♧) = (♧, \mathcal{A}, \longrightarrow)$ where ♧ and $\mathcal{A}$ are as in Definition 2.8 and $\longrightarrow$ is the least relation satisfying the rules in Table 2.1. The semantics of a process $p \in ♧$ is defined by the rooted transition system $(TS(♧), p)$. □

Example 2.11. It is not difficult to check that the transition system defined by the process *System* in Example 2.9 is the one depicted in Figure 2.1 where state σ_0 coincides with process $Arrival \,||_{on}\, SwitchOff$ and σ_1 with $Arrival \,||_{on}\, SwitchOn$. □

In the previous section we mentioned that one of the reasons why we are interested in equating processes is that we would like to have the possibility of replacing a process by an equivalent one. Since the process will be replaced in a context, we need that such equivalence is a *congruence*. An equivalence relation is a congruence if for every pair of equivalent processes p and q, and any context $\mathbf{C}[\ ]$, the processes $\mathbf{C}[p]$ and $\mathbf{C}[q]$ are also equivalent. The following theorem states that bisimilarity is a congruence for any ♧ context.

Theorem 2.12. *Let p and q two ♧ terms such that $p \sim q$. For any ♧ context $\mathbf{C}[\ ]$, it holds that $\mathbf{C}[p] \sim \mathbf{C}[q]$.*

In many cases, congruence theorems can be proven using standard results in structured operational semantics by simply inspecting the shape of the rules (de Simone, 1984; Bloom et al., 1995; Groote and Vaandrager, 1992; Baeten and Verhoef, 1993; Fokkink and van Glabbeek, 1996, and many others). This is exactly the case with the previous theorem. We recall the *path* format of (Baeten and Verhoef, 1993).

Let Rel, Rel_i be *transition relations* (as they could be any transition relation $\xrightarrow{a}$), and let $Pred$, $Pred_i$ be *state predicates*. Let op be an n-ary operation (for example, the binary operations $+$ or $||_A$). A rule is in *path* format if it has one of the following two forms,

$$\frac{\{p_i \; Rel_i \; y_i \mid i \in I\} \cup \{Pred_j \; q_j \mid j \in J\}}{\mathsf{op}(x_1, \ldots, x_n) \; Rel \; p} \tag{2.1}$$

$$\frac{\{p_i \; Rel_i \; y_i \mid i \in I\} \cup \{Pred_j \; q_j \mid j \in J\}}{Pred \; \mathsf{op}(x_1, \ldots, x_n)} \tag{2.2}$$

where $vars = \{x_1, \ldots, x_n\} \cup \{y_i \mid i \in I\}$ is a set of distinct variables (over terms), and p, p_i, q_j are terms with free variables in $vars$[2]. We say that a set of rules is in *path* format if all of its rules are.

Based on (Groote and Vaandrager, 1992), Baeten and Verhoef (1993) proved that bisimilarity is a congruence for any operation whose semantics is defined with rules in *path* format, provided they satisfy the so-called well-foundedness condition. Later, Fokkink (1994b) dropped that condition thus proving the general case. It is not difficult to check that rules in Table 2.1 are in *path* format, and that consequently $\sim$ is a congruence for all ♧ operations, from which Theorem 2.12 follows.

More recently, Rensink (1997) has proved a general congruence theorem for recursion with the only additional requirement that the rules should not have *look-ahead*, that is, the free variables of the p_i's, q_j's can only be those in $\{x_1, \ldots, x_n\}$. Thus, we can state the following theorem.

[2]We have actually defined the so-called *pure path format*, which is sufficient for the purposes of this thesis. Moreover, we have omitted two formats of rules since they can always be equivalently defined in terms of those given here.

Theorem 2.13. *Let $\tilde{H} \stackrel{\text{def}}{=} \{H_i \mid i \in I\}$ denote in general an indexed family over the index set I. Let $\tilde{p}$ and $\tilde{q}$ be two indexed families of ♧ processes with process variables only in $\tilde{X}$ such that, for every family $\tilde{r}$ of ♧ processes and for every $i \in I$, $p_i[\tilde{r}/\tilde{X}] \sim q_i[\tilde{r}/\tilde{X}]$. Define $Y_i = p_i[\tilde{Y}/\tilde{X}]$ and $Z_i = q_i[\tilde{Z}/\tilde{X}]$, $i \in I$. Then $Y_i \sim Z_i$, for all $i \in I$.*

Consider the recursive equation $X = a;\mathbf{0} + X$. Transition $X \xrightarrow{a} \mathbf{0}$ can be deduced in infinitely many ways since a new, different proof tree can be constructed by choosing the right hand side of the $+$ at different occasions. In this case, we say that variable X is defined with *unguarded recursion*. Unguarded recursion can also induce infinite branching in the obtained transition system as in the case of $Y = a;\mathbf{0} + (Y \mid_a a;b;\mathbf{0})$, in which the number of b's that is performed depends on the number of times the right hand side of $+$ is chosen when proving $Y \xrightarrow{a}$. Unguarded recursion is not always a desirable property. Especially if one thinks of the deduction system as something to be run on a computer. In the following, we formally characterise the notion of *guarding*.

Definition 2.14. An *occurrence of X is guarded* in a term $p \in$ ♧ if p has a subterm $a;q$ such that this occurrence of X is in q. A *term p is guarded* if all the occurrences of its process variables are guarded. A *recursive specification E is strictly guarded* if the right-hand side of every recursive equation in it is a guarded term, i.e., for all $X = p \in E$, p is guarded. A *recursive specification E is guarded* if it can be rewritten into a strictly guarded recursive specification by unfolding the equations. (By "properly unfolding the equations" we mean that whenever $X = p \in E$ and Y occurs unguarded in p, then we take the new specification $E' = (E \backslash \{X = p\}) \cup \{X = p[Y/q]\}$ provided that $Y = q \in E$.)

A recursive specification E is *regular* if it is guarded, it defines finitely many recursive equations (that is, E is a finite set), and for every recursive equation $X = p \in E$, the term p does not contain the operations $||_A$, $\between_A$, $|_A$, and $\_[f]$[3]. □

It is not difficult to prove that the semantics of a ♧ term whose variables are defined in a guarded recursive specification is a finitely branching transition system, and, conversely, that every finitely branching transition system can be written as a ♧ term with variables defined in a guarded recursive specification. Similarly, a ♧ term with variables defined in a regular recursive specification defines a finite transition system and vice-versa.

2.3 An Equational Theory for ♧

In this section we introduce an equational theory for ♧. An *equational theory*, also called *abstract algebra* or *variety*, is a structure containing a *signature*, which is a set of operations together with their arity, and a set of *equations* or *axioms* of the form $t = t'$, where t and t' are terms containing only operations in the signature and usually some free variables. We will use the terms *axiomatic system* or *axiomatisation* to refer to the set of axioms.

[3]The notion of regular recursive specifications could be generalised to include all the operations by carefully avoiding recursion cycles within the so-called static operations. However, this definition is not that simple and it does not extend expressiveness. As a consequence, it is sufficient for us to restrict the definition.

A1	$p + q = q + p$	**M**	$p \,\|\|_A\, q = p \,\LL_A\, q + q \,\LL_A\, p + p \,|_A\, q$	
A2	$(p + q) + r = p + (q + r)$	**LM1**	$\mathbf{0} \,\LL_A\, p = \mathbf{0}$	
A3	$p + p = p$	**LM2**	$a; p \,\LL_A\, q = \mathbf{0}$	if $a \in A$
A4	$p + \mathbf{0} = p$	**LM3**	$a; p \,\LL_A\, q = a; (p \,\LL_A\, q)$	if $a \notin A$
		LM4	$(p + q) \,\LL_A\, r = p \,\LL_A\, r + q \,\LL_A\, r$	
R1	$\mathbf{0}[f] = \mathbf{0}$	**CM1**	$\mathbf{0} \,|_A\, p = \mathbf{0}$	
R2	$(a; p)[f] = f(a); (p[f])$	**CM2**	$a; p \,|_A\, a; q = a; (p \,||_A\, q)$	if $a \in A$
R3	$(p + q)[f] = p[f] + q[f]$	**CM3**	$a; p \,|_A\, q = \mathbf{0}$	if $a \notin A$
		CM4	$(p + q) \,|_A\, r = p \,|_A\, r + q \,|_A\, r$	
		CM5	$p \,|_A\, q = q \,|_A\, p$	

Table 2.2: Axiom system for ♧

The interest of giving an equational theory for ♧ and any other process algebra is two-fold. First, the concept of algebra is fundamental in mathematics. That implies that the given axiom system will help for the understanding of the discussed process algebra and the concepts it involves. Second, the analysis and verification of systems described using the process algebra can be partially or completely carried out by mathematical proofs using the axiomatisation.

Definition 2.15. The equational theory for ♧ is defined as follows. The signature contains the constant $\mathbf{0}$, the unary operators $a;\_$ and $\_[f]$, for every $a \in \mathcal{A}$ and $f : \mathcal{A} \to \mathcal{A}$, and the binary operations $+$, $||_A$, $\LL_A$, and $|_A$, for every $A \subseteq \mathcal{A}$. The axiom system is given in Table 2.2. We identify with $sig(♧)$ and $ax(♧)$ the signature and axiom system of ♧. □

An equational theory defines an abstract structure that talks in general about different possible models which we call algebras. An *algebra* is a structure $(M, F, \simeq)$ where M is a set of objects called *carrier set*, F is a set of functions over M, and $\simeq$ is a congruence relation on M for the functions in F[4].

We say that such an algebra *models* a given equational theory if for each operation in the signature there is a corresponding function in F with the same arity, and besides,

[4]The usual definition of algebra does not include the relation $\simeq$. Instead, it considers equality to be the identity in M. Our definition is a simple generalisation but it does not define "more" algebras since it could be easily translated into the traditional definition by considering the quotient $M/\simeq$ as carrier set, and the set of functions $F/\simeq$ containing the functions of F with the usual transformation to range over $M/\simeq$.

every law $t = t'$ proved using the axiom system (together with equational reasoning) holds when the $=$ symbol is replaced by the $\simeq$ relation, every operation in the terms t and t' is replaced by their respective function in the algebra, and the free variables are consistently replaced by any object in M. In this case, we usually say that the equational theory is *sound* for the algebra. Conversely, we say that an equational theory is *complete* for the algebra if for any two objects in M that are equivalent according to $\simeq$, they can be proved to be equivalent by using the axiom system.

In the following we state soundness and completeness of ♧ with bisimilarity being the equivalence relation. We ambiguously use the same symbol to represent an operation in the signature and its respective function in the algebra.

Theorem 2.16 (Soundness). *The equational theory of ♧ given in Definition 2.15 is sound for the algebra* $(♧, sig(♧), \sim)$

The style of proving this theorem is classic in process algebra (e.g. Milner, 1989; Baeten and Weijland, 1990; Baeten and Verhoef, 1995, see also Aceto et al. 1994 where a systematic approach is reported). Because of the characteristics of equational reasoning, it is enough to consider only the axioms instead of any possible law derived from them as the definition of soundness requires. We do that as follows: for each axiom $t = t'$ in $ax(♧)$ define a bisimulation relation R containing every possible pair $\langle p, p' \rangle$ where p and p' are respective instances of t and t' with a consistent substitution of the free variables. As an example, the relation $\{\langle p + p, p \rangle \mid p \in ♧\} \cup \{\langle p, p \rangle \mid p \in ♧\}$, which can be proved to be a bisimulation, states soundness for axiom **A4**.

When we introduced ♧ we mentioned that operations $\mathbf{0}$, $a;\_$, and $+$ are basic operations since the other operators can be *eliminated* in favour of them. That is, every term has a provable equivalent basic term. In fact this is valid for the subset of ♧ containing only closed terms, i.e., terms not containing process variables. We denote by $♧^c$ the set of *closed ♧ terms*. Besides, we denote by $♧^b$ the set of *basic ♧ terms*, i.e., the subset of $♧^c$ containing only those terms formed using the operations $\mathbf{0}$, $a;\_$, and $+$.

To state elimination we transform some axioms and derived theorems into a strongly terminating rewriting system (Klop, 1992) and show that the normal form (i.e., the term that cannot be reduced any further) is a basic term. This is a usual technique in process algebras (see e.g. Bergstra and Klop, 1985; Baeten and Weijland, 1990; Baeten and Verhoef, 1995). In fact, for our case we consider axioms **R1**–**R3**, **M**, **LM1**–**LM4**, and **CM1**–**CM4** as rewrite rules with direction left to right. For instance, from **M** we take the rewrite rule

$$p \,||_A\, q \;\rightarrow\; p \,\between_A\, q + q \,\between_A\, p + p \,|_A\, q$$

Since including **CM5** as a rewrite rule would not give a strongly terminating rewrite system (i.e., a system in which there is no infinite reduction sequence), we do not do so, but use it to derive three more rules which give us the desired rewriting system. These rules are actually an appropriate variation of **CM1**, **CM3**, and **CM4** in which the terms are commuted respect to the communication merge. For instance, using axioms **CM4**, **CM5**, and **A4** we can prove that

$$p \,|_A\, (q + r) = p \,|_A\, q + p \,|_A\, r$$

and transform it also in a rewrite rule.

Theorem 2.17 (Elimination). *For every closed term $p \in \clubsuit^c$ there exists a basic term $q \in \clubsuit^b$ such that $p = q$ can be proven using the axiom system $ax(\clubsuit)$.*

Elimination explains which are the fundamental concepts in the theory —namely, the basic operations— by stating which are the minimal elements we can construct, and still keep the full expressive power of the theory. But elimination does have another important role. It is a key tool in proving completeness. Because of elimination we can restrict to prove completeness of the sub-theory induced by $\clubsuit^b$ and then have completeness of $\clubsuit^c$ for free. To be more precise, take two terms p and q in $\clubsuit^c$ such that $p \sim q$. Because of elimination there are $p', q' \in \clubsuit^b$ such that $p = p'$ and $q = q'$ are provable in $ax(\clubsuit)$, and by soundness and transitivity of $\sim$, $p' \sim q'$. So proving that $p' = q'$ is provable in $ax(\clubsuit)$, we have that $p = q$ is also provable in $ax(\clubsuit)$. The proof that the sub-theory containing the basic operations in the signature and axioms **A1**–**A4** as axiom system is complete for bisimulation in $\clubsuit^b$ is given in (Milner, 1989).

Theorem 2.18 (Completeness). *The equational theory of $\clubsuit$ given in Definition 2.15 is complete for the algebra $(\clubsuit^c, sig(\clubsuit), \sim)$.*

Notice that completeness only holds for the set of closed terms. When process variables and recursion are introduced, additional laws are needed (Milner, 1984, 1989). As simple counterexample we mention that the processes $a;\mathbf{0}$ and Y, defined by $Y = a;\mathbf{0} + Y$, are bisimilar, but they cannot be proven equivalent in $ax(\clubsuit)$.

The elimination theorem tells us which are the basic operations. A more elaborated theorem, the so-called *expansion law*, complements elimination by saying *how* the non-basic operations like the parallel composition can be decomposed in terms of the basic ones.

Theorem 2.19 (Expansion law). *Let $p = \sum a_i; p_i$ and $q = \sum b_j; q_j$ where $\sum_{k\in K} r_k$ denotes the process $r_1 + r_2 + \cdots + r_n$, if $K = \{1 \ldots n\}$, or $\mathbf{0}$, if $K = \varnothing$. Then, the following equalities can be proven from the axiom system $ax(\clubsuit)$.*

$$p \,||_A\, q = \sum_{a_i \notin A} a_i; (p_i \,||_A\, q) + \sum_{b_j \notin A} b_j; (p \,||_A\, q_j) + \sum_{a_i = b_j \in A} a_i; (p_i \,||_A\, q_j)$$

$$p[f] = \sum f(a_i); (p_i[f])$$

The last property we discuss in this introductory chapter is conservative extension. When developing a complex equational theory on top of a more basic one it is interesting to know whether the properties of the (original) basic theory are preserved in the extended one. More precisely, we can say that, given two equational theories E_1 and E_2 such that the signature of E_1 is contained in the one of E_2, E_2 is a conservative extension of E_1 if exactly the same theorems regarding only the original terms, i.e., those from E_1, can be proven in the axiom systems of both E_1 and E_2,. Formally, we say that E_2 is a *conservative*

extension of E_1, if $sig(E_1) \subseteq sig(E_2)$, and for every pair of closed terms p and q whose operations are in $sig(E_1)$, $p = q$ is provable in $ax(E_1)$ if and only if it is also provable in $ax(E_2)$.

As an example we can state the following theorem in the context of ♧.

Theorem 2.20 (Conservative extension). *The equational theory of ♧ given in Definition 2.15 is a conservative extension of the basic equational theory whose signature are the basic operations and its axiom system contains only the axioms* **A1**–**A4**.

The technique for proving this theorem has been formalised in (Verhoef, 1994) and generalised in (D'Argenio and Verhoef, 1997). From the latter, we recall that conservative extension can be guaranteed if the following conditions hold:

1. Every axiom of $ax(E_1)$ is provable in $ax(E_2)$[5],
2. E_1 is sound and complete for $(M_1, F_1, \simeq_1)$ and E_2 is sound for $(M_2, F_2, \simeq_2)$, and
3. $(M_2, F_2, \simeq_2)$ is a *model conservative extension* of $(M_1, F_1, \simeq_1)$, that is, $M_1 \subseteq M_2$, $F_1 \subseteq F_2$, and for all $t, t' \in M_1$, $t \simeq_1 t' \iff t \simeq_2 t'$.

In the context of structured operational semantics, model conservative extension reduces, in many occasions, to check two simple conditions. The first one can be carried out by simple inspection of the set of rules that defines the semantics of the terms (in our case the rules in Table 2.1). The new rules, that is, the rules introduced in the extended process algebra, should not introduce new relations or predicates for the terms in the old signature. Formally, take SOS_1 and SOS_2 to be the set of rules defining the semantics of the terms in $sig(E_1)$ and $sig(E_2)$ respectively. Then it should be the case that $SOS_1 \subseteq SOS_2$ and for every new rule in $(SOS_2 - SOS_1)$ —which has the shape of rule (2.1) or (2.2), see page 20— the operation op should not belong to the old signature $sig(E_1)$[6]. The second condition requires that the equivalence relation which is being axiomatised is defined in terms of predicates and relations only, as it is the case of bisimulation. Under these conditions conservative extension follows.

So, for Theorem 2.20, the original set of rules are those for $a;\_$ and $+$ in Table 2.1 (i.e., the three first rules), all the others are the "new rules". Clearly, the new rules only define the semantics of the new operations $||_A$, $\between_A$, $|_A$, and $\_[f]$. From the previous discussion, the theorem is hence proved.

[5]Actually (D'Argenio and Verhoef, 1997) asks that $ax(E_1) \subseteq ax(E_2)$, but this condition can be straightforwardly relaxed to what we ask here.

[6]This condition is stronger than that in (D'Argenio and Verhoef, 1997), which, however, is much more technically involved. For our purposes, the stronger condition will suffice.

Part Two

Timed Systems

'Er, how are we for time?' he said 'have I just got a min—'
And so the Universe ended.

Douglas Adams, *The Restaurant at the End of the Universe*. London, 1980

Chapter 3

Timed Transition Systems

Timed transition systems are a variant of transition systems which incorporate information about time. The usual way is to consider non-negative real numbers as part of the label on the transition relation. These numbers are intended to represent the necessary amount of time to move from the source state to the target state. There have been several definitions of timed transition systems. The most widely used contains transitions labelled with an action name or with a time label (a non-negative real number). Thus this kind of timed transition system is nothing else but a transition system with $\mathcal{A} \cup \mathbb{R}_{\geq 0}$ as set of labels. Usually some consistency requirements are imposed on the timed transitions —such as time additivity and time determinism (see Yi, 1990)— to ensure that time progresses as expected.

Another approach is to consider instead transitions labelled with both action names and time. Thus, $\sigma \xrightarrow{a(d)} \sigma'$ means that, at a state σ, it is possible to execute action a after idling for d time units and move to the new state σ'. This approach has been used in real-time ACP (Baeten and Bergstra, 1991; Klusener, 1993), to name the most popular case, and is the one that we use as a concrete model to describe real time behaviours.

In this chapter, we define timed transition systems and give the notion of equivalence we use to equate timed behaviour. This equivalence is a simple variation of bisimulation that takes the passage of time into account. The model we present is based on the model defined by Klusener (1993), although its formalisation is inspired by (Yi, 1990).

3.1 The Model

Timed transition systems are a variant of transition systems which incorporate information about time. Time is added by including a class of labels taken from a *time domain*. A time domain is a totally ordered monoid with an addition operation $+$ and a neutral element 0 that is also the minimum in the ordering, such that $d \leq d + d'$ for all d and d' belonging to the time domain, and $d + d' \leq d$ if and only if $d' = 0$. We choose the set $\mathbb{R}_{\geq 0}$ of non-negative real numbers as our time domain.

Definition 3.1. A *timed transition system* is a structure $TTS = (\Sigma, \mathcal{A} \times \mathbb{R}_{\geq 0}, \longrightarrow, \mathcal{U})$ where

- Σ is a set of *states*;
- $\mathcal{A}$ is a set of *actions*;
- $\longrightarrow \subseteq \Sigma \times (\mathcal{A} \times \mathbb{R}_{\geq 0}) \times \Sigma$ is the *transition relation*; and
- $\mathcal{U} \subseteq \mathbb{R}_{\geq 0} \times \Sigma$ is the *until predicate*.

We write $a(d)$ instead of (a, d) and, as for transition systems (see Definition 2.1), we use the shorthand notations $\sigma \xrightarrow{a(d)} \sigma'$, $\sigma \xrightarrow{a(d)}$, and $\sigma \not\xrightarrow{a(d)}$. We also write $\mathcal{U}_d(\sigma)$ if $\langle d, \sigma \rangle \in \mathcal{U}$.

In addition, for all $\sigma \in \Sigma$ and $a \in \mathcal{A}$, TTS should satisfies the following axioms:

Until $\quad \forall d, d' \in \mathbb{R}_{\geq 0}.\ \mathcal{U}_d(\sigma) \wedge d' < d \implies \mathcal{U}_{d'}(\sigma)$;

Delay $\quad \forall d \in \mathbb{R}_{\geq 0}.\ \sigma \xrightarrow{a(d)} \implies \mathcal{U}_d(\sigma)$.

As for transition systems, it will be convenient to distinguish an initial state $\sigma_0 \in \Sigma$. In this case we call the structure (TTS, σ_0), a *rooted timed transition system*. □

The intended meaning of a transition $\sigma \xrightarrow{a(d)} \sigma'$ is that, whenever the system is in state σ, it can change to state σ' by performing an action a after idling d units of time. Intuitively, $\mathcal{U}_d(\sigma)$ means that the system can idle in a state σ at least d units of times. Axiom **Until** states that if the system can idle d units of time in a state σ, then it is also able to idle there for less time. Axiom **Delay** says that every time that the system is in a state σ in which an action a can be performed after idling d units of time, it should be explicitly said that the system in state σ can idle until that time.

The reader might think that predicate $\mathcal{U}$ is redundant, however this is not the case. Predicate $\mathcal{U}$ is crucial to model features like urgency, time divergence and time deadlock. We briefly explain each of them.

- A system in state σ is urged to perform an action a at time d if it cannot idle any further in that state, and it has the possibility to do this action. Formally we can say that state σ is *urgent* at time d if $\sigma \xrightarrow{a(d)}$ for some action a and there is no $d' > d$ such that $\mathcal{U}_{d'}(\sigma)$.
- A system is *time divergent* in state σ if it can idle there for ever, that is, if $\mathcal{U}_d(\sigma)$ for all $d \in \mathbb{R}_{\geq 0}$. Notice that, in this case, if $\sigma \xrightarrow{a(d)}$, for a particular d, the system may still choose not to do action a and keep idling. Compare this with the case of urgency in which the action becomes a mandatory escape as time progresses.
- Finally, a system has a *time deadlock* in a state σ if it can idle there until a time d at which it is not allowed to idle anymore but at the same time there is no possible action outgoing from σ, i.e., $\mathcal{U}_d(\sigma)$, $\sigma \not\xrightarrow{a(d)}$ for any $a \in \mathcal{A}$, and there is no $d' > d$ such that $\mathcal{U}_d(\sigma)$.

The features we have just discussed are quite important to model and analyse systems. In fact, they are crucial for the verification of so-called hard real-time systems. For instance, urgency is the key component to describe time-outs, and a time deadlock may represent an endpoint in the lifetime of the described system, usually with catastrophic consequences.

Example 3.2. In this example we consider the same switch in a stairway that we already described in Example 2.2, but we explicitly model an additional feature. We fix the time when the light must be turned off. So, assuming that the light is turned off 2 minutes after the last time the on button was pressed, the switch can be modelled by the timed transition system $(\Sigma, \mathcal{A} \times \mathbb{R}_{\geq 0}, \longrightarrow, \mathcal{U})$ where

$$\begin{array}{ll} \Sigma = \{\sigma_0, \sigma_1\} & \mathcal{A} = \{on, off\} \\ \sigma_0 \xrightarrow{on(d)} \sigma_1 \iff 0 \leq d & \mathcal{U}_d(\sigma_0) \iff 0 \leq d \\ \sigma_1 \xrightarrow{on(d)} \sigma_1 \iff 0 \leq d \leq 2 & \mathcal{U}_d(\sigma_1) \iff 0 \leq d \leq 2 \\ \sigma_1 \xrightarrow{off(2)} \sigma_0 & \end{array}$$

Notice that in state σ_0 the system is allowed to idle indefinitely, while in state σ_1, when the light is on, the system cannot idle beyond 2 minutes. After idling 2 minutes the system is forced to execute either the action *off* or *on*. Notice that action *off* can only take place after idling exactly 2 minutes and not before. While in σ_0 the time diverges, in σ_1 an urgency situation arises at time 2, and actually, it urges for a decision between turning off the light or resetting the timer by pressing on again.

If we choose σ_0 to be the initial state, we can model the switch as a rooted timed transition system. □

The kind of timed transition system we have defined is particular to this thesis. However their relation with existing models is straightforward. To begin with, our model is based on (Klusener, 1993). The main difference is that Klusener uses absolute time, that is, the time stamp (i.e. the d in $\xrightarrow{a(d)}$ and $\mathcal{U}_d$) indicates the time elapsed since the very beginning of the process, while we use relative time, meaning that the time stamp refers to the time elapsed since the current state has been reached (or, alternatively, since the last action occurred). In fact, predicate $\mathcal{U}$ is borrowed from (Klusener, 1993). Here, we formalised its behaviour in a relative time setting by adding the axioms **Until** and **Delay**.

But probably the most widely used timed transition system model is the one that considers discrete actions (i.e. actions in $\mathcal{A}$) and timed actions separately as labels of the transition relation (see Yi, 1990, 1991, for a formalisation). Under the usually accepted conditions of time determinism and time additivity property (Yi, 1990), this model is equivalent to ours.

3.2 Timed Bisimulation

To equate timed transition systems we use a variant of bisimulation semantics (see Definition 2.3), which is called timed bisimulation.

Definition 3.3. Let $(\Sigma, \mathcal{A} \times \mathbb{R}_{\geq 0}, \longrightarrow, \mathcal{U})$ be a timed transition system. A relation $R \subseteq \Sigma \times \Sigma$ is a *timed bisimulation relation* if R is symmetric and for all $\langle \sigma_1, \sigma_2 \rangle \in R$ and for all $a \in \mathcal{A}$ and $d \in \mathbb{R}_{\geq 0}$, the following transfer properties hold:

1. if $\sigma_1 \xrightarrow{a(d)} \sigma_1'$ then $\sigma_2 \xrightarrow{a(d)} \sigma_2'$ for some $\sigma_2' \in \Sigma$ such that $\langle \sigma_1', \sigma_2' \rangle \in R$; and
2. if $\mathcal{U}_d(\sigma_1)$ then $\mathcal{U}_d(\sigma_2)$.

We say that two states σ_1 and σ_2 are *timed bisimilar*, notation $\sigma_1 \sim_t \sigma_2$, if there is a timed bisimulation R that relates them, i.e., $\langle \sigma_1, \sigma_2 \rangle \in R$.

Two rooted timed transition systems (TTS_1, σ_1) and (TTS_2, σ_2) are *timed bisimilar*, if their initial states σ_1 and σ_2 are timed bisimilar in the (disjoint) union of TTS_1 and TTS_2. □

Notice that by symmetry of R, the second transfer property could be also written as "$\mathcal{U}_d(\sigma_1)$ if and only if $\mathcal{U}_d(\sigma_2)$".

Relation $\sim_t$ can be proved to be a timed bisimulation and an equivalence relation following the same proof as for $\sim$. Notice that by definition $\sim_t$ contains all timed bisimulations, which makes it the largest one. As in Section 2.1, we refer to (Milner, 1989) for the proof that shows $\sim$ is an equivalence and the largest bisimulation. Thus, we state the following theorem.

Theorem 3.4.

1. *$\sim_t$ is a timed bisimulation relation, and*
2. *$\sim_t$ is an equivalence relation over the set of states Σ.*

As for bisimulation, we can define the notion of timed bisimulation up to $\sim_t$, and it is also the case that finding a timed bisimulation up to $\sim_t$ suffices to show timed bisimilarity.

Definition 3.5. Let $(\Sigma, \mathcal{A} \times \mathbb{R}_{\geq 0}, \longrightarrow, \mathcal{U})$ be a timed transition system. A relation $R \subseteq \Sigma \times \Sigma$ is a *timed bisimulation relation up to $\sim_t$* if R is symmetric and for all $\langle \sigma_1, \sigma_2 \rangle \in R$ and for all $a \in \mathcal{A}$ and $d \in \mathbb{R}_{\geq 0}$, the following transfer properties hold:

1. if $\sigma_1 \xrightarrow{a(d)} \sigma_1'$ then $\sigma_2 \xrightarrow{a(d)} \sigma_2'$ for some $\sigma_2' \in \Sigma$ such that $\langle \sigma_1', \sigma_2' \rangle \in (\sim_t \cup R)^*$; and
2. if $\mathcal{U}_d(\sigma_1)$ then $\mathcal{U}_d(\sigma_2)$. □

Theorem 3.6. *Let R be a timed bisimulation up to $\sim_t$. Then $R \subseteq (\sim_t \cup R)^* \subseteq \sim_t$.*

Proof. It is sufficient to prove that $(\sim_t \cup R)^*$ is a timed bisimulation. It is not difficult to check that $(\sim_t \cup R)^*$ is symmetric. So, it remains to check that the transfer properties hold. Since $(\sim_t \cup R)^* = \bigcup_{n \geq 0} (\sim_t \cup R)^n$ we prove that by induction on n.

The case for $n = 0$ is trivial. So, suppose $\langle \sigma_1, \sigma_2 \rangle \in \bigcup_{0 \leq h \leq n+1} (\sim_t \cup R)^h$. Then, there is a $\sigma \in \Sigma$ such that $\langle \sigma_1, \sigma \rangle \in \bigcup_{0 \leq h \leq n} (\sim_t \cup R)^h$ and $\langle \sigma, \sigma_2 \rangle \in \sim_t \cup R$.

1. Suppose $\sigma_1 \xrightarrow{a(d)} \sigma_1'$. Then, $\sigma \xrightarrow{a(d)} \sigma'$ and $\langle \sigma_1', \sigma' \rangle \in (\sim_t \cup R)^*$, for some $\sigma' \in \Sigma$, by induction hypothesis. Besides, $\sigma \xrightarrow{a(d)} \sigma'$ implies $\sigma_2 \xrightarrow{a(d)} \sigma_2'$ and either $\langle \sigma', \sigma_2' \rangle \in \sim_t$ or $\langle \sigma', \sigma_2' \rangle \in (\sim_t \cup R)^*$, for some $\sigma_2' \in \Sigma$, depending on whether $\langle \sigma, \sigma_2 \rangle \in \sim_t$ or $\langle \sigma, \sigma_2 \rangle \in R$. Thus, $\sigma_2 \xrightarrow{a(d)} \sigma_2'$ and $\langle \sigma_1', \sigma_2' \rangle \in (\sim_t \cup R)^*$.

2. Clearly $\mathcal{U}_d(\sigma_1)$ implies $\mathcal{U}_d(\sigma)$ and $\mathcal{U}_d(\sigma)$ implies $\mathcal{U}_d(\sigma_2)$, from which this transfer property follows. *Q.E.D.*

Chapter 4

Timed Automata

Although timed transition systems give an adequate mathematic framework to represent the behaviour of real-time systems, they certainly are inappropriate to do automatic analysis due to their intrinsically infinite size, let alone to use it as a formalism for the specification of timed systems. Therefore, there is a need to represent a timed system in a symbolic way. Probably, the most successful formalism to symbolically specify real-time systems is the timed automata model (Alur and Dill, 1990, 1994; Henzinger et al., 1994). A timed automata is a transition system annotated with timers. The timers, which are called *clocks*, can be reset to 0 and they increase according time passes. Constraints on the value of the clocks will determine whether a transition may or must be executed.

Timed automata have been successfully used in automatic verification of real-time systems, particularly by model checking properties given in an appropriate temporal logic (Alur et al., 1993; Henzinger et al., 1994). Many tools has been implemented using this methodology. In particular, the tools KRONOS (Yovine, 1993; Olivero, 1994; Bozga et al., 1998) and UPPAAL (Larsen et al., 1997; Pettersson, 1999) reached an appropriate level of maturity; they have been satisfactorily used in many actual case studies (see, e.g., Bengtsson et al., 1996a; Maler and Yovine, 1996; D'Argenio et al., 1997b).

In this chapter we introduce the timed automata model (actually, a slight variant of the traditional model introduced by Henzinger et al., 1994), and define its semantics in terms of timed transition systems. We also define some equivalence relations directly on timed automata following the bisimulation style. These notions take into account the symbolic information and, since they are strictly finer than timed bisimulation, they are particularly useful to prove equivalence of behaviour in many cases, without the need to go down to the concrete behaviour given by timed transition systems.

4.1 Clocks and Clock Constraints

In Example 3.2, we described a set of possible transitions by simply saying

$$\sigma_1 \xrightarrow{on(d)} \sigma_1 \iff 0 \leq d \leq 2$$

$\dfrac{v(x) \leq d}{\models v(x \leq d)}$	$\dfrac{v(x) - v(y) \leq d}{\models v(x - y \leq d)}$	$\dfrac{\models v(\phi) \quad \models v(\phi')}{\models v(\phi \wedge \phi')}$
$\dfrac{d \leq v(x)}{\models v(d \leq x)}$	$\dfrac{d \leq v(x) - v(y)}{\models v(d \leq x - y)}$	$\dfrac{\not\models v(\phi)}{\models v(\neg\phi)}$

Table 4.1: Satisfaction of a clock constraint in a valuation v

To do this, the constraint $0 \leq d \leq 2$ is crucial, and similarly for the definition of $\mathcal{U}_d$. In fact, constraints like this are the ones we are going to use to describe symbolically the timed behaviour of systems.

Apart from that, we are going to consider an additional ingredient: the so called *clock variables* or *clocks* for short. A clock is a variable that allows us to record the passage of time. Like a chronometer, they can be reset to 0 and inspected at any moment to see how much time has passed since their resets. By having several different clock variables we can measure and compare the timing of different events, as we define all clocks to increase at the same rate.

The way to inspect clocks is by means of constraints defined as follows.

Definition 4.1. Given a set of clock variables $\mathcal{C}$ we define the set Φ of *clock constraints* according to the following grammar

$$\phi \quad ::= \quad x \leq d \mid d \leq x \mid x - y \leq d \mid d \leq x - y \mid (\phi \wedge \phi) \mid (\neg\phi)$$

where $d \in \mathbb{R}_{\geq 0}$ and $x, y \in \mathcal{C}$. If ϕ is a clock constraint in Φ, we denote with $var(\phi)$ the set of clock variables occurring in ϕ. □

We notice that abbreviations like $\mathbf{tt}$, $\mathbf{ff}$, $x = d$, $x > d$, $x - y < d$, $\phi \vee \phi'$, $x \in [d, d')$, can be defined in Φ. For instance, $\mathbf{tt} \stackrel{\text{def}}{\iff} x - x \leq 0$ or $x \in [d, d') \stackrel{\text{def}}{\iff} (d \leq x) \wedge \neg(d' \leq x)$.

Constant numbers in clock constraints are usually restricted to non-negative integers. This is required for decidability matters when doing analysis of real time systems. Since in our work, we are not bound to decidability concerns, we will not impose this restriction. Nonetheless, this restriction must be considered as soon as any analysis technique is applied in any framework we are going to discuss afterwards.

The satisfaction of a constraint depends on the value each clock takes. Hence, the satisfaction of a constraint is subject to a *valuation*. A valuation is a function $v : \mathcal{C} \to \mathbb{R}_{\geq 0}$ that assigns a non-negative real value to each clock. We let $\mathbf{V}$ denote the set of all valuations.

We define the *satisfaction* of a constraint ϕ in a valuation v, denoted by $\models v(\phi)$, inductively in the usual way (see Table 4.1). We say that a constraint ϕ *holds*, and denote $\models \phi$, if ϕ is satisfied in every valuation v, that is for all v, $\models v(\phi)$.

Since a valuation is a snapshot of the clock values in a particular instance, we need operations that manipulate valuations consistently as time evolves. The first operation defines the valuation that is obtained after d units of time has passed since we observed an old valuation v. This valuation is denoted by $v + d$ and defined by

$$(v + d)(x) \stackrel{\text{def}}{=} v(x) + d$$

for all clock $x \in \mathcal{C}$. The second operation gives the new valuation after resetting a set of clocks C in a given valuation v. We denote it by $v[C \hookleftarrow 0]$, $C \subseteq \mathcal{C}$, and define, for all $x \in C$

$$v[C \hookleftarrow 0](x) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x \in C \\ v(x) & \text{if } x \notin C \end{cases} .$$

Recall that axiom **Until** in Definition 3.1 states that if the system can idle d units of time at a given sate, then it is also able to idle there for less time. We can also characterise the class of clocks constraints that meet a similar property. They are called past-closed constraints. A clock constraint $\phi \in \Phi$ is *past-closed* if for every valuation $v \in \mathbf{V}$ and time $d \in \mathbb{R}_{\geq 0}$,

$$\models (v + d)(\phi) \implies \models v(\phi)$$

We let $\overline{\Phi}$ denote the set of all past-closed constraints.

4.2 The Model

In Example 3.2, we described a simple system using timed transition systems. Since the number of transitions was uncountably large, we resorted to some symbolic description. Although not in our example, the number of states may also be uncountable and we would need to mention all of them also in a symbolic way.

Timed automata were introduced precisely to represent timed behaviour in such a symbolic manner. A timed automaton is basically a transition system whose arrows and nodes are decorated with clocks and clock constraints.

Definition 4.2. A *timed automaton* is a structure $TA = (\mathcal{S}, \mathcal{A}, \mathcal{C}, \rightarrowtriangle, \iota, \kappa)$ where:

- $\mathcal{S}$ is a set of *locations*;
- $\mathcal{A}$ is a set of *actions*;
- $\mathcal{C}$ is a set of *clocks*;
- $\rightarrowtriangle \subseteq \mathcal{S} \times (\mathcal{A} \times \Phi) \times \mathcal{S}$ is the set of *edges*;
- $\iota : S \to \overline{\Phi}$ is the *invariant assignment* function; and

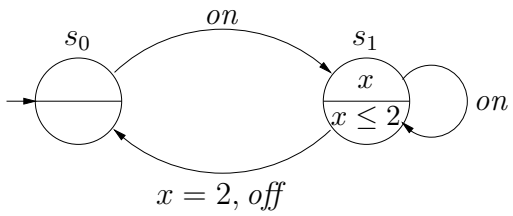


Figure 4.1: A timed automaton representing a switch in a stairway

- $\kappa : S \to \wp(\mathcal{C})$ is the *clock resetting* function.

We write $s \xrightarrow{a,\phi} s'$ instead of $(s, a, \phi, s') \in \longrightarrow$ and $s \xrightarrow{a,\phi}$ whenever there is an s' such that $s \xrightarrow{a,\phi} s'$.

In many occasions it will be of convenience to distinguish an initial location $s_0 \in \mathcal{S}$. In this case we call the structure (TA, s_0), a *rooted timed automaton.* □

$s \xrightarrow{a,\phi} s'$ intuitively means that when the system is in location s it could change to location s' by performing an action a only in those moments in which the clock constraint ϕ, called *guard*, is satisfied. Immediately afterwards, all clocks in $\kappa(s')$ are reset. A timed automaton is allowed to idle in a location s only as long as the associated invariant $\iota(s)$ is satisfied.

Example 4.3. The switch of Example 3.2 can be described by the timed automaton $System \stackrel{\text{def}}{=} (\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \iota, \kappa)$ where

$$\begin{array}{llll} \mathcal{S} = \{s_0, s_1\} & s_0 \xrightarrow{on,\mathbf{tt}} s_1 & \kappa(s_0) = \varnothing & \iota(s_0) = \mathbf{tt} \\ \mathcal{A} = \{on, off\} & s_1 \xrightarrow{on,\mathbf{tt}} s_1 & \kappa(s_1) = \{x\} & \iota(s_1) = (x \leq 2) \\ \mathcal{C} = \{x\} & s_1 \xrightarrow{off,x=2} s_1 & & \end{array}$$

We assume s_0 to be the initial location. The timed automaton is depicted in Figure 4.1. There, we represent the locations by circles which are divided in two halves. Inside the upper half we enumerate the clocks to be reset according to κ. In the lower half, we write the invariant of the location assigned according to ι. Edges are represented by arrows labelled with the corresponding action and clock constraint. To avoid messing the picture we do not write the constraint $\mathbf{tt}$, the empty set, and the braces enclosing sets. □

Timed automata were introduced by Alur and Dill (1990, 1994) although the concept of invariant was added later in the so called timed safety automata (Henzinger et al., 1994; Nicollin et al., 1992, 1993). The model we have defined is a slight variation of the timed safety automata model. We defined the clock resettings to be associated to locations instead of edges. The main reason to do so is that it allows a simpler treatment when using timed automata as the underlying semantic model of a real-time process algebra (see Chapter 5). Our approach allows to have a prefixing operation (like in ♧) and a separate operation

representing the clock resetting and yet having a straightforward semantics. Compare with the process algebra of (Yi et al., 1994; Pettersson, 1999) in which the prefix operation is integrated with both the clock resetting and the guard.

Nonetheless, this variation does neither restrict nor add expressiveness. Our timed automata can be translated into timed safety automata by "pulling" the clock resettings out of the target location into the arrow, that is, an edge $s \xrightarrow{a,\phi} s'$ will be translated into $s \xrightarrow{a,\phi,\kappa(s')} s'$. Conversely, a timed automaton with resettings on the edges could be transformed by "pushing" the clock resetting into the target state, i.e., given an edge $s \xrightarrow{a,\phi,C} s'$ we define $s \xrightarrow{a,\phi} s'$ and $\kappa(s') \stackrel{\text{def}}{=} C$. In case many edges with different clock resettings go to the same location, this state is "split" into different locations, one for each different set of clocks.

Remark 4.4. We remark that transition systems are a subset of timed automata. In fact, every transition system can be translated into a timed automaton in the following way. Given the transition system $(\Sigma, \mathcal{A}, \longrightarrow)$, we define the timed automaton $(\Sigma, \mathcal{A}, \varnothing, \rightarrow, \iota, \kappa)$ where $\kappa(\sigma) = \varnothing$ and $\iota(\sigma) = \mathbf{tt}$ for all $\sigma \in \Sigma$, and $\sigma \xrightarrow{a,\mathbf{tt}} \sigma'$ whenever $\sigma \xrightarrow{a} \sigma'$. That is, a transition system is a timed automaton in which the occurrence of activity is not bound to any time.

Conversely, every timed automaton which is not bound to time can be straightforwardly translated into a transition system by removing all the "additional ingredients". Timed automata whose activity are time dependent cannot be translated into transition systems without losing information. □

In the same way we did for transition systems in Definition 2.7, we can characterise whether a timed automaton is finite or finitely branching. The way in which we define it is quite the same.

Definition 4.5. Let $TA = (\mathcal{S}, \mathcal{A}, \mathcal{C}, \rightarrow, \iota, \kappa)$ be a timed automaton. TA is *finitely branching* if for all $s \in \mathcal{S}$ the set

$$\{s \xrightarrow{a,\phi} s' \mid a \in \mathcal{A} \wedge \phi \in \Phi \wedge s' \in \mathcal{S}\}$$

is finite. We say that TA is *finite* if the set of locations $\mathcal{S}$ is also finite. □

It has been shown in (Alur et al., 1993; Alur and Dill, 1994) that reachability analysis is possible for finite timed automata, provided that the constants used in the clocks constraints are integers, or can be scaled to integers.

4.3 Semantics for Timed Automata

In Chapter 3 we introduced a concrete model to describe timed behaviour: the timed transition systems. In the previous section we defined timed automata to symbolically specify timed systems, but we only explained intuitively their behaviour. In the following we define the semantics of timed automata in terms of timed transition systems.

Definition 4.6. Let $TA = (\mathcal{S}, \mathcal{A}, \mathcal{C}, \rightarrow, \iota, \kappa)$ be a timed automaton. The *interpretation* of TA is given by the timed transition system $TTS(TA) = (\mathcal{S} \times \mathbf{V}, \mathcal{A} \times \mathbb{R}_{\geq 0}, \longrightarrow, \mathcal{U})$ where $\longrightarrow$ and $\mathcal{U}$ are defined as the least sets satisfying the following rules:

$$\textbf{Idle} \quad \frac{\models (v[\kappa(s) \hookleftarrow 0] + d)(\iota(s))}{\mathcal{U}_d(s, v)}$$

$$\textbf{Act} \quad \frac{s \xrightarrow{a,\phi} s' \qquad \models (v[\kappa(s) \hookleftarrow 0] + d)(\phi \wedge \iota(s))}{(s, v) \xrightarrow{a(d)} (s', (v[\kappa(s) \hookleftarrow 0] + d))}$$

The interpretation of the rooted timed automaton (TA, s_0) in the initial valuation $v_0 \in \mathbf{V}$ is given by the rooted timed transition system $(TTS(TA), (s_0, v_0))$. □

Since $\iota(s)$ is a past-closed constraint, it follows that $TTS(TA)$ satisfies axiom **Until** in Definition 3.1. Moreover, notice that if $(s, v) \xrightarrow{a(d)}$ then $\models (v[\kappa(s) \hookleftarrow 0] + d)(\iota(s))$ and so $\mathcal{U}_d(s, v)$, which implies that axiom **Delay** holds as well. Hence, $TTS(TA)$ is indeed a timed transition system.

Rule **Idle** states that the system is allowed to idle in a location s which was reached with clocks valuated according to v, if after resetting the clocks in $\kappa(s)$ and letting d units of time pass, the invariant $\iota(s)$ still holds. Rule **Act** explains the case in which a transition can take place. So, if a location s with an outgoing edge $s \xrightarrow{a,\phi} s'$ was reached when the clocks where valued according to v, action a can be executed after d units of time if, in addition to the invariant $\iota(s)$, the guard ϕ holds in valuation v after resetting the clocks in $\kappa(s)$ and idling for d units of time. Once the action is executed, the system changes its location to s' and hence it is reached with the clocks valued according to the valuation in which s was left, that is, $v[\kappa(s) \hookleftarrow 0] + d$.

Example 4.7. By Definition 4.6, the semantics of the timed automaton in Example 4.3 is the timed transition system $TTS(System) = (\mathcal{S} \times \mathbf{V}, \mathcal{A} \times \mathbb{R}_{\geq 0}, \longrightarrow, \mathcal{U})$ where

$$(s_0, (x := d')) \xrightarrow{on(d)} (s_1, (x := d' + d)) \iff d, d' \in \mathbb{R}_{\geq 0}$$
$$(s_1, (x := d')) \xrightarrow{on(d)} (s_1, (x := d)) \iff 0 \leq d \leq 2,\ d' \in \mathbb{R}_{\geq 0}$$
$$(s_1, (x := d)) \xrightarrow{off(2)} (s_0, (x := 2)) \iff d \in \mathbb{R}_{\geq 0}$$
$$\mathcal{U}_d(s_0, (x := d')) \iff d, d' \in \mathbb{R}_{\geq 0}$$
$$\mathcal{U}_d(s_1, (x := d')) \iff 0 \leq d \leq 2,\ d' \in \mathbb{R}_{\geq 0}$$

Choosing valuation $x := 0$ to be initial, we see that $(TTS(System), (s_0, (x := 0)))$ is a possible interpretation for the rooted timed automaton. □

4.4 Equivalences on Timed Automata

In the following we define several notions of equivalences for timed automata. In the first place, we extend timed bisimilarity to timed automata. This is rather straightforward since we do that on top of the semantics of timed automata in terms of timed transition systems.

Definition 4.8. Two locations s_1 and s_2 of a timed automaton TA are *timed bisimilar*, notation $s_1 \sim_t s_2$, if they are timed bisimilar in $TTS(TA)$ for all valuations, that is, if for every $v \in \mathbf{V}$, $(s_1, v) \sim_t (s_2, v)$.

Two rooted timed automata (TA_1, s_1) and (TA_2, s_2) are *timed bisimilar* if their interpretations are timed bisimilar for every initial valuation, that is, if $(TTS(TA_1), (s_1, v_0))$ and $(TTS(TA_2), (s_2, v_0))$ are timed bisimilar for every valuation $v_0 \in \mathbf{V}$. □

Since $\sim_t$ is an equivalence relation on the set of the actual states, it is immediate that $\sim_t$ is also an equivalence relation on the set of locations $\mathcal{S}$.

Timed bisimulation is our preferred equivalence since it precisely captures the timed and branching behaviour of a timed system. However, proving things to be timed bisimilar directly on the timed transition system can be quite difficult. In many cases, we do not need to resort to the definition of timed bisimulation to prove that two timed automata are timed bisimilar. We can state that without leaving the symbolic description by using stronger equivalences defined at this level.

In the following, we define three different notions of equivalences and we relate them to each other and to timed bisimulation. We show that timed bisimulation is the weakest relation and isomorphism is the strongest one. To start with, we define the notion of isomorphism on timed automata.

Definition 4.9. Let $TA = (\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \iota, \kappa)$ and $TA' = (\mathcal{S}', \mathcal{A}, \mathcal{C}, \longrightarrow', \iota', \kappa')$ be two timed automata. We say that TA and TA' are *isomorphic*, notation $TA \cong TA'$, if there is a bijective function $\mathfrak{I} : \mathcal{S} \to \mathcal{S}'$ such that

1. $s \xrightarrow{a,\phi} s' \iff \mathfrak{I}(s) \xrightarrow{a,\phi}{}' \mathfrak{I}(s')$,
2. $\iota(s) = \iota'(\mathfrak{I}(s))$, and
3. $\kappa(s) = \kappa'(\mathfrak{I}(s))$.

In this case we say that the function $\mathfrak{I}$ is an *isomorphism*. Two rooted timed automata (TA, s_0) and (TA', s_0') are *isomorphic* if in addition $\mathfrak{I}(s_0) = s_0'$. □

The definition of isomorphism could be extended by allowing a bijection on the clocks and another on the actions. For our purposes, the simpler Definition 4.9 is sufficient. Although isomorphism is a simple notion, it is certainly too strong. In the following we define a structural bisimulation which, although strong, it will be useful in many cases since it is defined in the style of bisimulation but directly on timed automata.

Definition 4.10. Let $(\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \iota, \kappa)$ be a timed automaton. A relation $R \subseteq \mathcal{S} \times \mathcal{S}$ is a *structural bisimulation* if R is symmetric and for all $a \in \mathcal{A}$ and $\phi \in \Phi$, whenever $\langle s_1, s_2 \rangle \in R$ the following transfer properties hold:

1. if $s_1 \xrightarrow{a,\phi} s_1'$, then there is a $s_2' \in \mathcal{S}$ such that $s_2 \xrightarrow{a,\phi} s_2'$ and $\langle s_1', s_2' \rangle \in R$;
2. $\iota(s_1) = \iota(s_2)$; and
3. $\kappa(s_1) = \kappa(s_2)$.

We say that two locations s_1 and s_2 are *structurally bisimilar*, and denote $s_1 \sim_s s_2$, if there exists a structural bisimulation R such that $\langle s_1, s_2 \rangle \in R$.

Two rooted timed automata (TA_1, s_1) and (TA_2, s_2) are *structurally bisimilar*, if their initial locations s_1 and s_2 are structurally bisimilar in the (disjoint) union of TA_1 and TA_2. □

The proof that $\sim_s$ is the largest structural bisimulation relation and that it is an equivalence relation follows exactly the same technique as the proof for standard bisimulation. Thus, we state the following theorem without proof.

Theorem 4.11.

1. *$\sim_s$ is a structural bisimulation relation, and*
2. *$\sim_s$ is an equivalence relation on the set $\mathcal{S}$ of locations.*

As for the other bisimulations, we can define the notion of structured bisimulation up to $\sim_s$.

Definition 4.12. Let $(\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \iota, \kappa)$ be a timed automaton. A relation $R \subseteq \mathcal{S} \times \mathcal{S}$ is a *structural bisimulation up to $\sim_s$* if R is symmetric and for all $a \in \mathcal{A}$ and $\phi \in \Phi$, whenever $\langle s_1, s_2 \rangle \in R$ the following transfer properties hold

1. $s_1 \xrightarrow{a,\phi} s_1'$, then there is a $s_2' \in \mathcal{S}$ such that $s_2 \xrightarrow{a,\phi} s_2'$ and $\langle s_1', s_2' \rangle \in (\sim_s \cup R)^*$;
2. $\iota(s_1) = \iota(s_2)$; and
3. $\kappa(s_1) = \kappa(s_2)$. □

The following theorem, which states that finding a structured bisimulation up to $\sim_s$ is enough to prove structured bisimilarity, can be proved in similar way to Theorem 3.6.

Theorem 4.13. *Let R be a structural bisimulation up to $\sim_s$. Then $R \subseteq (\sim_s \cup R)^* \subseteq \sim_s$.*

Structural bisimulation does not consider the actual behaviour of the automaton but rather its syntactic structure. Some simple examples which we would reasonably require to be equal but are not are given in Figure 4.2. Although yet too strong, structural

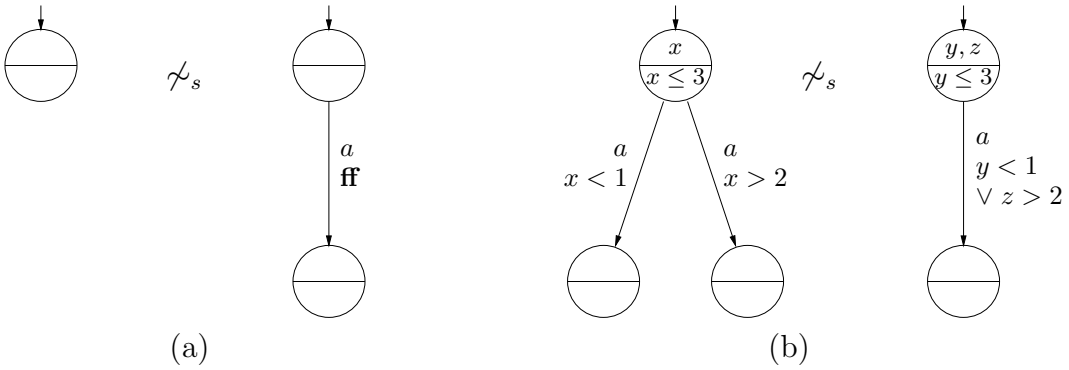


Figure 4.2: Two examples of non structurally bisimilar timed automata

bisimulation will turn out to be useful to prove the soundness of many axioms for the process algebra that we will introduce in the following chapters.

In the following, we define a symbolic bisimulation which equates timed automata like in Figure 4.2. In addition to the structure of timed automata, the symbolic bisimulation considers that, in some cases, clocks with different names may respond to the same functionality or value, and besides, it takes into account that invariants and guards can be equivalent according to an adequate renaming of clocks.

From this rough description, it becomes clear that special care should be taken with clocks. We will need the notion of free variables of a particular location. A free variable is a clock variable which may appear in a guard or invariant sometime in the future without being reset (i.e. before it appears in a clock resetting set).

Definition 4.14. Let $(\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \iota, \kappa)$ be a timed automaton. The set of *free variables* of a location $s \in \mathcal{S}$, notation $FV(s)$, is defined as the smallest set satisfying

$$FV(s) = \Big(\{x \mid s \xrightarrow{a,\phi} s' \wedge x \in var(\phi) \cup FV(s')\} \cup var(\iota(s))\Big) - \kappa(s)$$

□

Symbolic bisimulation considers clocks which are set at the same moment to be equivalent. As a consequence, a symbolic bisimulation relation contains triplets $\langle s_1, s_2, SR\rangle$, where s_1 and s_2 are locations, and SR is a component explaining how the clocks relate. We call SR a *synchronisation relation*. Concretely, SR contains pairs of sets of clocks $\langle C_1, C_2\rangle$. Each C_i contains clock that can be free in s_i or bound by $\kappa(s_i)$. The idea is that clocks in $C_1 \cup C_2$ are "synchronised" in the sense that they have been reset at the same moment. Therefore, in particular $\langle \kappa(s_1), \kappa(s_2)\rangle \in SR$.

Knowing that some clocks have been synchronised (i.e., reset at the same time) according to SR, a pair of (not necessarily equivalent) constraints may become equivalent. For instance, if SR contains the pairs $\langle\{x\},\{y\}\rangle$ and $\langle\{z\},\{x,w\}\rangle$ then $\phi_1 \equiv (x - z > 2)$ and $\phi_2 \equiv (y - x > 2) \wedge (x - w \leq 2)$ turn to be equivalent. This can be easily concluded if

we define two suitable *substitutions* that rename clocks in the same pair to the same clock variable. In the aforementioned example, define the substitutions ξ^1_{SR} and ξ^2_{SR} such that

$$\xi^1_{SR}(x) = \xi^2_{SR}(y) \stackrel{\text{def}}{=} \hat{x} \quad \text{and} \quad \xi^1_{SR}(z) = \xi^2_{SR}(x) = \xi^2_{SR}(w) \stackrel{\text{def}}{=} \hat{y}$$

for some clocks $\hat{x}, \hat{y} \in \mathcal{C}$, then we have

$$\xi^1_{SR}(\phi_1) \iff (\hat{x} - \hat{y} > 2) \iff (\hat{x} - \hat{y} > 2) \wedge (\hat{x} - \hat{x} \leq 2) \iff \xi^2_{SR}(\phi_2)$$

We formalise the definition of these substitutions in the following.

Definition 4.15. A *substitution* is a function $\xi : \mathcal{C} \to \mathcal{C}$ and it extends to clock constraints as expected.

Let $SR \subseteq \wp(\mathcal{C}) \times \wp(\mathcal{C})$ such that for all $\langle C_1, C_2\rangle, \langle C'_1, C'_2\rangle \in SR$, if $(C_1 \cap C'_1) \cup (C_2 \cap C'_2) \neq \varnothing$ then $\langle C_1, C_2\rangle = \langle C'_1, C'_2\rangle$. We say that ξ^1_{SR} and ξ^2_{SR} are the *left and right substitutions induced by SR* if they satisfy

$$\xi^i_{SR}(x) = \xi^j_{SR}(y) \quad \stackrel{\text{def}}{\iff} \quad \exists \langle C_1, C_2\rangle \in SR.\ x \in C_i \wedge y \in C_j$$

for all $x, y \in \mathcal{C}$, $i, j \in \{1, 2\}$. □

Notice that ξ^i_{SR} is well defined since elements in SR are required to be, roughly speaking, pairwise disjoint.

The definition of the symbolic bisimulation heavily relies on SR and the induced left and right substitutions. Thus, in a triplet $\langle s_1, s_2, SR\rangle$, the invariants of s_1 and s_2 have to be equivalent under the hypothesis that clocks are synchronised according to SR. Similarly, the simulation of the edges, i.e. the transfer properties that involve the edges, has to preserve the guards under SR. In this case it is also important that, once the target locations of the edges are reached —say, they are s'_1 and s'_2—, they have to be related in a new triplet $\langle s'_1, s'_2, SR'\rangle$ and in this case SR' has to preserve the old equivalence imposed by SR for the clocks that are still relevant. We call this last property *forward compatibility*.

Definition 4.16. Let $(\mathcal{S}, \mathcal{A}, \mathcal{C}, \rightarrow, \iota, \kappa)$ be a timed automaton. A *symbolic bisimulation*—indexbisimulation!symbolic is a relation $R \subseteq \mathcal{S} \times \mathcal{S} \times \wp(\wp(\mathcal{C}) \times \wp(\mathcal{C}))$ that, whenever $\langle s_1, s_2, SR\rangle \in R$, the following properties hold.

1. $\langle C_1, C_2\rangle, \langle C'_1, C'_2\rangle \in SR$ and $(C_1 \cap C'_1) \cup (C_2 \cap C'_2) \neq \varnothing$ then $\langle C_1, C_2\rangle = \langle C'_1, C'_2\rangle$.
2. For $i = 1, 2$, $\bigcup\{C_i \mid \langle C_1, C_2\rangle \in SR\} \supseteq FV(s_i) \cup \kappa(s_i)$.
3. $\langle \kappa(s_1), \kappa(s_2)\rangle \in SR$.
4. $\models \xi^1_{SR}(\iota(s_1)) \Leftrightarrow \xi^2_{SR}(\iota(s_2))$.
5. $s_1 \xrightarrow{a,\phi_1} s'_1$, then either
   (a) $\models \neg(\xi^1_{SR}(\iota(s_1) \wedge \phi_1))$, or

(b) there are $s_2^{(1)}, \ldots, s_2^{(n)} \in \mathcal{S}$, $n \geq 1$ such that

i. $s_2 \xrightarrow{a,\phi_2^{(i)}} s_2^{(i)}$ for all $i = 1, \ldots, n$;

ii. $\models \xi^1_{SR}(\iota(s_1) \wedge \phi_1) \Rightarrow \bigvee_i \xi^2_{SR}(\iota(s_2) \wedge \phi_2^{(i)})$; and

iii. for all i, there exists SR' such that $\langle s_1', s_2^{(i)}, SR' \rangle \in R$ and it is *forward compatible* with SR, that is,

$$\{\langle C_1 \cap FV(s_1'), C_2 \cap FV(s_2^{(i)}) \rangle \mid \langle C_1, C_2 \rangle \in SR'\} - \{\langle \varnothing, \varnothing \rangle\} = \{\langle C_1 \cap FV(s_1'), C_2 \cap FV(s_2^{(i)}) \rangle \mid \langle C_1, C_2 \rangle \in SR\} - \{\langle \varnothing, \varnothing \rangle\}.$$

6. $s_2 \xrightarrow{a,\phi_2} s_2'$, then either

(a) $\models \neg(\xi^2_{SR}(\iota(s_2) \wedge \phi_2))$, or

(b) there are $s_1^{(1)}, \ldots, s_1^{(n)} \in \mathcal{S}$, $n \geq 1$ such that

i. $s_1 \xrightarrow{a,\phi_1^{(i)}} s_1^{(i)}$ for all $i = 1, \ldots, n$;

ii. $\models \xi^2_{SR}(\iota(s_2) \wedge \phi_2) \Rightarrow \bigvee_i \xi^1_{SR}(\iota(s_1) \wedge \phi_1^{(i)})$; and

iii. for all i, there exists SR' such that $\langle s_1^{(i)}, s_2', SR' \rangle \in R$ and it is *forward compatible* with SR, that is,

$$\{\langle C_1 \cap FV(s_1^{(i)}), C_2 \cap FV(s_2') \rangle \mid \langle C_1, C_2 \rangle \in SR'\} - \{\langle \varnothing, \varnothing \rangle\} = \{\langle C_1 \cap FV(s_1^{(i)}), C_2 \cap FV(s_2') \rangle \mid \langle C_1, C_2 \rangle \in SR\} - \{\langle \varnothing, \varnothing \rangle\}.$$

We say that two locations s_1 and s_2 are *symbolically bisimilar*, and denote $s_1 \sim_{\&} s_2$, if there exists a symbolic bisimulation R such that $\langle s_1, s_2, SR \rangle \in R$ for some SR satisfying

(a) if $\langle \{x\} \cup C_1, C_2 \rangle \in SR$ and $x \in FV(s_1)$ then $C_1 \subseteq \{x\}$;

(b) if $\langle C_1, \{x\} \cup C_2 \rangle \in SR$ and $x \in FV(s_2)$ then $C_2 \subseteq \{x\}$; and

(c) if $\langle \{x\}, \{y\} \rangle \in SR$, $x \in FV(s_1)$, and $y \in FV(s_2)$ then $x = y$.

Two rooted timed automata (TA_1, s_1) and (TA_2, s_2) are *symbolically bisimilar*, if their initial locations s_1 and s_2 are symbolically bisimilar in the (disjoint) union of TA_1 and TA_2. □

The definition of symbolic bisimulation may look impressive, but it is not so hard to understand if one has in mind that the sets SR are intended to represent equivalence relations on clocks which are modified consistently as the simulation steps are taken. Notice that SR cannot simply be an equivalence relation since we need to differentiate whether a clock is relevant in the left or the right location. We will find out sometimes that clock x on the left should match clock y on the right and clock y on the left should be matched to some other clock on the right, perhaps, clock x. The example set SR we used before

Definition 4.15, contained pairs $\langle\{x\},\{y\}\rangle$ and $\langle\{z\},\{x,w\}\rangle$, where the x of the first pair is intended to be different to the one in the second pair.

Condition 1 requires that SR is indeed a set of "equivalence classes" by making sure that the elements in it are pairwise disjoint. In other words, one clock should be matched uniquely to the same set of clocks both in the left side and in the right side. Besides SR should contain the useful clocks, that is, clocks which are going to be used in the future starting from the current locations. The useful clocks are those that are set or free in the current location. This is stated by condition 2. Item 3 synchronises all the clocks that are set at the same moment by requiring that they share the same equivalence class.

The invariant of two related locations should be equivalent up to the synchronisation of clocks as is required in 4. The transfer property 5 states in which way an arrow outgoing from the left location should be simulated by the related location. Condition 5a checks whether the arrow is relevant or not, that is, whether or not it is guarded by a false statement. Otherwise case 5b should be considered. In fact, an arrow can be simulated by several arrows (item 5(b)i). At any moment the simulated arrow is enabled, it should hold that at least one of the simulating arrows should be enabled (item 5(b)ii). Besides, target locations should also be related together with a new set of equivalence classes SR', with the requirement that SR' should be forward compatible with SR, that is, it should synchronise the same relevant clocks as SR. Clocks which are not free in the new locations are not relevant anymore. This last condition is stated in item 5(b)iii. Finally item 6 states the symmetric transfer property.

Intuitively, two locations may be related by a symbolic bisimulation but they are not necessarily symbolically bisimilar. This has to do with the fact that a triplet $\langle s_1, s_2, SR\rangle$ "remembers" in SR how the clocks were related in the past (refer to the forward compatibility property in 5(b)iii). For two locations to be symbolically bisimilar, their free clock variables cannot be arbitrarily synchronised. Following the criterion of timed bisimulation in which two locations are timed bisimilar if they are related in every valuation (see Definition 4.8), we require that SR satisfies the following additional conditions. First, we require that free variables are not related to any other variable at the same side. This is stated by items (a) and (b). We also require that if two free variables of different sides are related they should have the same name (condition (c)).

As expected, $\sim_{\&}$ is an equivalence relation. In this case, the proof is not standard.

Theorem 4.17. *$\sim_{\&}$ is an equivalence relation on the set $\mathcal{S}$ of locations.*

Proof. To prove that $\sim_{\&}$ is an equivalence relation we need to prove that it is reflexive, symmetric, and transitive. Reflexivity follows from the fact that $\sim_s \subseteq \sim_{\&}$, which is proven below in Theorem 4.19. If R is a symbolic bisimulation, it is not difficult to prove that $\{\langle s_2, s_1, SR^{-1}\rangle \mid \langle s_1, s_2, SR\rangle \in R\}$ is also a symbolic bisimulation, which proves symmetry. Moreover, notice that this relation satisfy conditions (a), (b), and (c) in Definition 4.16 whenever the given relation R does. Transitivity is the most complicated to prove. We state that in a separate lemma which can be found in Appendix A.1. *Q.E.D.*

In the following we relate all notions of equivalence on timed automata we defined so

far. First we state that, given two isomorphic timed automata, any pair of locations related by the isomorphism are also structurally bisimilar.

Theorem 4.18. *Let $\mathfrak{I}$ be an isomorphism from TA to TA′. Then the locations s and $\mathfrak{I}(s)$ are structurally bisimilar in the (disjoint) union of TA and TA′.*

Proof. The fact that the relation $R \stackrel{\text{def}}{=} \{\langle s, \mathfrak{I}(s)\rangle \mid s \in \mathcal{S}\}$ is a structural bisimulation should be clear. *Q.E.D.*

In the next theorem we state that if two locations are structurally bisimilar, they are also symbolically bisimilar.

Theorem 4.19. *Let s_1 and s_2 be two locations in a timed automaton TA. If $s_1 \sim_s s_2$ then $s_1 \sim_{\&} s_2$.*

Proof. Let R_s be a structural bisimulation such that $\langle s_1, s_2\rangle \in R_s$. We define the relation $R_{\&}$ as the least relation satisfying the following rules.

$$\frac{\langle s_1, s_2\rangle \in R_s \qquad SR = \{\langle \kappa(s_1), \kappa(s_2)\rangle\} \cup \{\langle \{x\} - \kappa(s_1), \{x\} - \kappa(s_2)\rangle \mid x \in FV(s_1) \cup FV(s_2)\}}{\langle s_1, s_2, SR\rangle \in R_{\&}}$$

$$\frac{\langle s_1', s_2', SR'\rangle \in R_{\&} \qquad s_1' \xrightarrow{a,\phi} s_1'' \qquad s_2' \xrightarrow{a,\phi} s_2'' \qquad \langle s_1', s_2'\rangle \in R_s \qquad SR'' = \{\langle \kappa(s_1''), \kappa(s_2'')\rangle\} \cup \{\langle C_1 \cap FV(s_1''), C_2 \cap FV(s_2'')\rangle \mid \langle C_1, C_2\rangle \in SR'\}}{\langle s_1'', s_2'', SR''\rangle \in R_{\&}}$$

It is not difficult to check that $R_{\&}$ is a symbolic bisimulation. From this, and from the fact that SR in the first rule satisfies conditions (a)–(c) in Definition 4.16, the theorem follows. *Q.E.D.*

Similarly we state that if two locations are symbolically bisimilar, they are also timed bisimilar. The proof of the following theorem can be found in Appendix A.2.

Theorem 4.20. *Let s_1 and s_2 be two locations in a timed automaton TA. If $s_1 \sim_{\&} s_2$ then $s_1 \sim_t s_2$.*

We summarise the relation between the equivalences introduced before. We have that

1. two isomorphic (rooted) timed automata are also structurally bisimilar,
2. two structurally bisimilar timed automata are also symbolically bisimilar,
3. two symbolically bisimilar timed automata are also timed bisimilar, and
4. in none of the previous cases the converse holds in general.

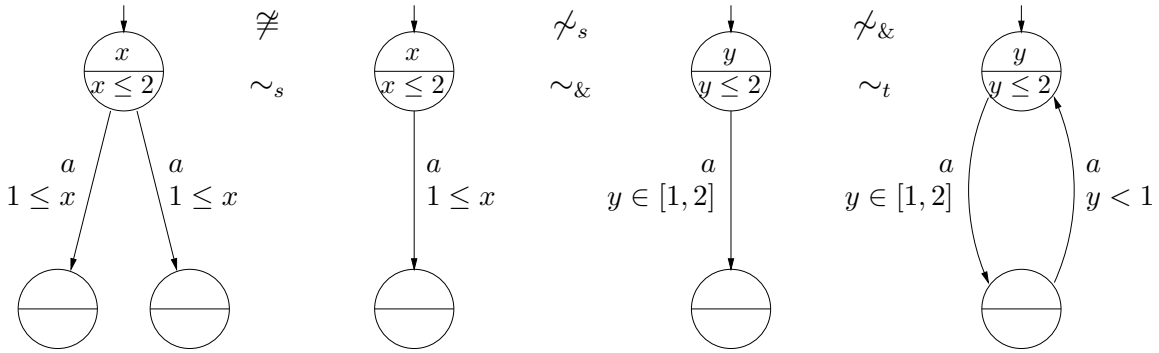


Figure 4.3: Example of different equivalences on timed automata

The three first statements are immediate consequences of Theorems 4.18, 4.19, and 4.20 respectively. The last statement follows from the examples depicted in Figure 4.3.

Remark 4.21. We mention that it is easy to show that structural bisimulation coincides with bisimulation on transition systems when timed automata are restricted to those representable by transition systems as explained by Remark 4.4. In fact it is not difficult to show that timed bisimulation also coincides with bisimulation under this condition, and consequently, also symbolic bisimulation. □

Chapter 5

♡ – Algebra for Real-Time Systems

As we have already discussed, timed automata give the adequate framework for the representation and analysis of real-time systems. Nevertheless, its syntax becomes unwieldy to specify realistic systems. Timed process algebras instead present a higher-level language that facilitates the hierarchical and structured description of real-time systems. To mention a few: ACP with integration (Baeten and Bergstra, 1991; Klusener, 1993), different versions of timed CCS (Yi, 1990, 1991; Moller and Tofts, 1990; Chen, 1993) and timed CSP (Davies et al., 1992), ATP (Nicollin and Sifakis, 1994), extensions of LOTOS with time (Bolognesi and Lucidi, 1992; Leduc and Léonard, 1994), and timed μCRL (Groote, 1997). Although relations of some of this process algebras with timed automata have been studied (see Section 5.8), none of them presents a complete and straight connection.

In this chapter, we introduce a process algebra to describe timed automata. We represent it by the symbol ♡ and pronounce it as *hearts* by close phonetics to the acronym *ARTS: algebra for real-time systems*. ♡ extends ♧ with three new operations. These operations correspond to the ingredients that make the difference between timed automata and transition systems, namely, clock resettings, invariants, and guards. In fact, ♡ is a language for timed automata. This can be easily seen in a simple example. Consider the timed automaton of Figure 5.1. In ♡ we can write

$$\{\!|x|\!\}\,(x \leq 2) \gg (x > 1) \mapsto a; \mathbf{0}$$

to describe the same behaviour. Like in ♧, $a; \mathbf{0}$ represents the process that executes action a and terminates. The timed automaton has some additional annotations that constrain the time in which a should be executed. ♡ provides the necessary operations to represent such a timing constraint. Operation $\{\!|x|\!\}\ \ldots$ represents the clock resetting in the initial

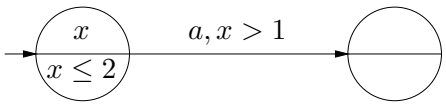


Figure 5.1: A simple timed automata

location, $\ldots(x \leq 2) \gg \ldots$ corresponds to the invariant, and the guard on the arrow is represented by $\ldots(x > 1) \mapsto \ldots$.

After discussing the syntax in the first section, we present an operational semantics that associates a timed automaton to every ♡ term. We also show that any timed automaton can be coded in terms of ♡, hence concluding that both the automata and algebraic approach are equally expressive. Finally, we give a direct semantics in terms of timed transition systems and prove that it is equivalent, or more precisely *timed bisimilar*, to the interpretation of the associated timed automaton.

5.1 Syntax

Like timed automata, ♡ uses clocks and constraints over clocks (see Section 4.1). A ♡ term is constructed using any of the operations of ♧ plus three new operations. The old operations define the same processes when reinterpreted in ♡. It is worth mentioning that ♧ operations do not impose any time restrictions. Nevertheless, their behaviour should be extended to take time into account. For instance, $a;p$ is a process that *at any moment* can perform action a and continue executing p.

One of the new operations is the *guarding operation*. Given a clock constraint ϕ, $\phi \mapsto p$ represents a process that can execute any action p can, but only at those moments ϕ holds. The *invariant operation* controls how long a process is allowed to idle. Thus, given a past closed constraint ψ, $\psi \gg p$ is a process that can idle as long as ψ holds or execute p before. In fact, p must be executed before ψ does not hold anymore. The last of the new operations is the *clock resetting*. $\{\!|C|\!\}\, p$ is a process that behaves like p but resets all the clocks in C before doing anything else. We will usually write $\{\!|x_1, \ldots, x_n|\!\}\, p$ instead of $\{\!|\{x_1, \ldots, x_n\}|\!\}\, p$.

Definition 5.1. Let $\mathcal{A}$ be a set of action names and let $\mathcal{C}$ be a set of clocks. The language ♡ is defined according to the following grammar

$$\begin{array}{rcl} p & ::= & \mathbf{0} \mid a;p \mid \phi \mapsto p \mid \psi \gg p \mid \{\!|C|\!\}\, p \mid p + p \\ & & \mid\ p \,|\!|_A\, p \mid p \,\llbracket_A\, p \mid p \,|_A\, p \mid p[f] \mid X \end{array}$$

where $a \in \mathcal{A}$ is an *action name*, $\phi \in \Phi$ is a *guarding constraint*, $C \subseteq \mathcal{C}$ is a *clock resetting* set, $\psi \in \overline{\Phi}$ is a *time invariant*, $A \subseteq \mathcal{A}$ is a *synchronisation set*, $f : \mathcal{A} \to \mathcal{A}$ is a *renaming function*, and X is a *process variable* belonging to the set $\mathcal{V}$ of process variables.

As for ♧, process variables are defined by *recursive equations* of the form $X = p$ where p is a ♡ term, and a set E of recursive equation defines a *recursive specification*. We occasionally consider a distinguished process variable in the recursive specification E called *root*. □

Definition 2.14 applies also for ♡. Thus, the notion of *guarded* and *regular* recursive specifications is defined for ♡ exactly as for ♧.

We will assume that the new operations $\phi \mapsto \_$, $\{\!|C|\!\}\, \_$, and $\psi \gg \_$ have the highest precedence together with prefixing and renaming. Notice that no confusion arises if we use

these operations (except renaming) without parenthesis. Thus $\{\!|x|\!\}\,(x < 5) \gg (x > 2) \mapsto a; X$ means $\{\!|x|\!\}\,(\,(x < 5) \gg (\,(x > 2) \mapsto (a; X)\,)\,)$.

Example 5.2. We enhance the ♧ specification of the switch given in Example 2.9 to meet the time requirements as in Examples 3.2 and 4.3. The ♡ specification of the switch is as follows. The arrival of people can happen at any time. Thus,

$$\mathit{Arrival} \;=\; \mathit{on}; \mathit{Arrival}$$

The switch can be turned on at any moment, however it turns itself off after exactly two minutes of idling.

$$\begin{aligned} \mathit{SwitchOff} &= \mathit{on}; \mathit{SwitchOn} \\ \mathit{SwitchOn} &= \{\!|x|\!\}\,(x \le 2) \gg (\;\mathit{on}; \mathit{SwitchOn} \\ &\qquad\qquad + (x = 2) \mapsto \mathit{off}; \mathit{SwitchOff}\;) \end{aligned}$$

The complete system is described by the composition of the switch and the arriving people which interact on action *on*.

$$\mathit{System} \;=\; \mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOff}$$

□

As the reader may notice at this point, the clock resetting operation is a *binder*, that is, $\{\!|C|\!\}\,p$ binds every free occurrence of clocks belonging to C in p. This immediately induces the concepts of free and bound (clock) variables. Intuitively a clock x is free in p if p has a subterm $\phi \mapsto q$ or $\phi \gg q$, with x occurring in ϕ, that does not appear in a context $\{\!|\ldots x \ldots|\!\}\,\ldots$. A clock x is bound in p if p has a subterm $\{\!|\ldots x \ldots|\!\}\,q$. We define formally only the set $\mathit{fv}(p)$ of free variables of the process p.

Definition 5.3. Let $p \in$ ♡. The set $\mathit{fv}(p)$ of *free (clock) variables* of p is defined as the least set satisfying the equations in Table 5.1. □

Thus, for instance, if $p \equiv \{\!|x|\!\}\,(y - x < 4) \gg a; \mathbf{0} + (x \ge 2) \mapsto b; \mathbf{0}$ then $\mathit{fv}(p) = \{x, y\}$; notice, however, that x appears also bound.

5.2 Semantics in Terms of Timed Automata

We define the semantics of ♡ in terms of timed automata. A timed automaton is associated to a ♡ term using structured operational semantics. In order to define the semantics of the parallel composition, we need to consider the auxiliary function $\overline{\mathsf{ck}}$. Given a process p, $\overline{\mathsf{ck}}(p)$ returns a process that behaves like p except that no clock is reset at the very beginning. It is defined recursively in Table 5.2. Notice that $\overline{\mathsf{ck}}(X)$ defines a new variable and, besides, it holds that $\overline{\mathsf{ck}}(\overline{\mathsf{ck}}(p)) = \overline{\mathsf{ck}}(p)$.

To give semantics to a ♡ term, we need to define the different parts of the timed automaton. We start by defining the clock setting function κ, the invariant ι and the set

$fv(\mathbf{0}) = \varnothing$		$fv(p \,\|\|_A\, q) = fv(p) \cup fv(q)$	
$fv(a;p) = fv(p)$		$fv(p \,\llbracket\!\rrbracket_A\, q) = fv(p) \cup fv(q)$	
$fv(\phi \mapsto p) = var(\phi) \cup fv(p)$		$fv(p \,\|_A\, q) = fv(p) \cup fv(q)$	
$fv(p+q) = fv(p) \cup fv(q)$		$fv(p[f]) = fv(p)$	
$fv(\{\!\|C|\!\}\, p) = fv(p) - C$		$fv(X) = fv(p)$	$(X = p)$
$fv(\psi \gg p) = var(\psi) \cup fv(p)$			

Table 5.1: Free and bound variables in ♡

$\overline{\mathsf{ck}}(\mathbf{0}) = \mathbf{0}$	$\overline{\mathsf{ck}}(a;p) = a;p$	$\overline{\mathsf{ck}}(\{\!\|C|\!\}\, p) = \overline{\mathsf{ck}}(p)$
$\overline{\mathsf{ck}}(\mathsf{op}(p)) = \mathsf{op}(\overline{\mathsf{ck}}(p))$	$(\mathsf{op} \in \{\phi \mapsto \_, \psi \gg \_, \_[f]\})$	
$\overline{\mathsf{ck}}(p \oplus q) = \overline{\mathsf{ck}}(p) \oplus \overline{\mathsf{ck}}(q)$	$(\oplus \in \{+, \|\|_A, \llbracket\!\rrbracket_A, \|_A\})$	
$\overline{\mathsf{ck}}(X) = X_{\overline{\mathsf{ck}}}$	(provided $X = p$ and $X_{\overline{\mathsf{ck}}} = \overline{\mathsf{ck}}(p)$)	

Table 5.2: Auxiliary clock removal operation

of edges $\rightarrowtail$ as the least relations satisfying the rules in Table 5.3. Notice that it can easily be deduced that

$$\kappa(\overline{\mathsf{ck}}(p)) = \varnothing \qquad \iota(\overline{\mathsf{ck}}(p)) = \iota(p) \qquad \frac{p \xrightarrow{a,\phi} p'}{\overline{\mathsf{ck}}(p) \xrightarrow{a,\phi} p'} \tag{5.1}$$

and $fv(p) \subseteq fv(\overline{\mathsf{ck}}(p)) \subseteq fv(p) \cup \kappa(p)$.

The rules capture the intuitive behaviour of ♡ terms. Notice that $\mathbf{0}$ cannot execute any action but it can idle indefinitely. The execution of a in process $a;p$ does not have any temporal restriction, neither in the invariant nor in its guard; thus, it can be postponed at will. $\phi \mapsto p$ can perform any action p can; however, they are restricted to the temporal boundaries imposed by ϕ. This can be observed from the fact that ϕ is "added" to the guard on the transition. Process $p + q$ can idle as long as at least one of p and q can. Moreover, $p + q$ can execute any action p or q can at the moment it is executed in the original process. Thus, since an action cannot be executed after the idling time is finished, we require that, for its execution, the corresponding invariant must also hold. In principle, processes $\{\!\|C|\!\}\, p$ and $\psi \gg p$ have the same outgoing edges as p since these operators only

$$
\begin{array}{lll}
\kappa(\mathbf{0}) & = & \varnothing \\
\kappa(a;p) & = & \varnothing \\
\kappa(\phi \mapsto p) & = & \kappa(p) \\
\kappa(p+q) & = & \kappa(p) \cup \kappa(q) \\
\kappa(\{\!|C|\!\}\, p) & = & C \cup \kappa(p) \\
\kappa(\psi \gg p) & = & \kappa(p)
\end{array}
\qquad
\begin{array}{lll}
\kappa(p \,||_A\, q) & = & \kappa(p) \cup \kappa(q) \\
\kappa(p \,\llcorner\!\!\lrcorner_A\, q) & = & \kappa(p) \cup \kappa(q) \\
\kappa(p \,|_A\, q) & = & \kappa(p) \cup \kappa(q) \\
\kappa(p[f]) & = & \kappa(p) \\
\kappa(X) & = & \kappa(p) \qquad (X = p)
\end{array}
$$

$$
\begin{array}{lll}
\iota(\mathbf{0}) & = & \mathbf{tt} \\
\iota(a;p) & = & \mathbf{tt} \\
\iota(\phi \mapsto p) & = & \iota(p) \\
\iota(p+q) & = & \iota(p) \vee \iota(q) \\
\iota(\{\!|C|\!\}\, p) & = & \iota(p) \\
\iota(\psi \gg p) & = & \psi \wedge \iota(p)
\end{array}
\qquad
\begin{array}{lll}
\iota(p \,||_A\, q) & = & \iota(p) \wedge \iota(q) \\
\iota(p \,\llcorner\!\!\lrcorner_A\, q) & = & \iota(p) \wedge \iota(q) \\
\iota(p \,|_A\, q) & = & \iota(p) \wedge \iota(q) \\
\iota(p[f]) & = & \iota(p) \\
\iota(X) & = & \iota(p) \qquad (X = p)
\end{array}
$$

$$a;p \xrightarrow{a,\mathbf{tt}} p$$

$$\frac{p \xrightarrow{a,\phi} p' \quad \iota(p) = \psi}{p + q \xrightarrow{a,\phi\wedge\psi} p'} \qquad \frac{p \xrightarrow{a,\phi} p' \quad \iota(p) = \psi}{q + p \xrightarrow{a,\phi\wedge\psi} p'}$$

$$\frac{p \xrightarrow{a,\phi'} p'}{\phi \mapsto p \xrightarrow{a,\phi\wedge\phi'} p'} \qquad \frac{p \xrightarrow{a,\phi} p'}{\{\!|C|\!\}\, p \xrightarrow{a,\phi} p'} \qquad \frac{p \xrightarrow{a,\phi} p'}{\psi \gg p \xrightarrow{a,\phi} p'}$$

$$\frac{p \xrightarrow{a,\phi} p'}{p \,||_A\, q \xrightarrow{a,\phi} p' \,||_A\, \overline{\mathsf{ck}}(q)}\ a \notin A \qquad \frac{p \xrightarrow{a,\phi} p'}{q \,||_A\, p \xrightarrow{a,\phi} \overline{\mathsf{ck}}(q) \,||_A\, p'}\ a \notin A$$

$$\frac{p \xrightarrow{a,\phi} p' \quad q \xrightarrow{a,\phi'} q'}{p \,||_A\, q \xrightarrow{a,\phi\wedge\phi'} p' \,||_A\, q'}\ a \in A$$

$$\frac{p \xrightarrow{a,\phi} p'}{p \,\llcorner\!\!\lrcorner_A\, q \xrightarrow{a,\phi} p' \,||_A\, \overline{\mathsf{ck}}(q)}\ a \notin A \qquad \frac{p \xrightarrow{a,\phi} p' \quad q \xrightarrow{a,\phi'} q'}{p \,|_A\, q \xrightarrow{a,\phi\wedge\phi'} p' \,||_A\, q'}\ a \in A$$

$$\frac{p \xrightarrow{a,\phi} p'}{p[f] \xrightarrow{f(a),\phi} p'[f]} \qquad \frac{p \xrightarrow{a,\phi} p'}{X \xrightarrow{a,\phi} p'}\ X = p$$

Table 5.3: SOS rules for ♡

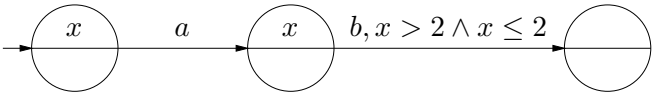
add information to the state. For $\{\!|C|\!\}\, p$, clocks in C are reset together with the clocks to be reset by p. The invariant of $\psi \gg p$ is restricted to satisfy ψ in addition to the invariant of p.

Since in $p \,||_A\, q$, p and q are running in parallel each one of them cannot idle longer than its invariant permits. Thus, the conjunction of these invariants determines the general invariant of $p \,||_A\, q$. Notice that, to execute a synchronisation action $a \in A$, a must be enabled in both p and q. That is why the conjunction of the independent guards is the new guard. Non-synchronisation actions proceed as in the original process though special care is needed for the process which remains idle. This is the situation in which the auxiliary operation $\overline{\mathsf{ck}}$ comes into play. If we admitted an edge $p \,||_A\, q \xrightarrow{a,\phi} p' \,||_A\, q$ instead of $p \,||_A\, q \xrightarrow{a,\phi} p' \,||_A\, \overline{\mathsf{ck}}(q)$, the clocks of q, which were reset as soon as $p \,||_A\, q$ was reached, would be reset again when $p' \,||_A\, q$ is reached after performing action a. This last situation does not reflect the intuitive behaviour since time for the process q would seem not to have progressed.

However, the semantics is not yet well defined. In fact, not every process can have a straightforward timed automaton as a semantic interpretation. To do so, clock names must be considered with care. Consider, for instance, the process

$$p_1 \equiv \{\!|x|\!\}\, a; (x > 2) \mapsto \{\!|x|\!\}\, (x \leq 2) \mapsto b; \mathbf{0} \tag{5.2}$$

The second occurrence of x is intended to be bound to the outermost clock setting as shown by the grey arrow. Using the rules in Table 5.3, the following timed automaton would be obtained.

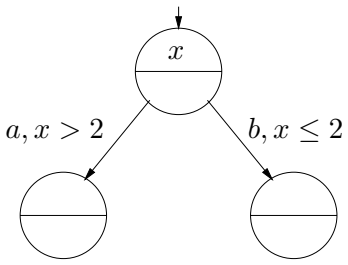


In this sense, x would be captured by the innermost clock setting as shown by the black arrow in (5.2). Therefore, we consider that clocks are different if they are set at different locations, although they may have the same name. A similar problem occurs with invariants instead of guards.

Unintended binding of clock names, which we simply call *clock capture*, may also occur in contexts with summations and parallel composition. Consider the process

$$p_2 \equiv \{\!|x|\!\}\, (x > 2) \mapsto a; \mathbf{0} + (x \leq 2) \mapsto b; \mathbf{0}$$

Naively, the following timed automaton could be wrongly interpreted for p_2.



This timed automaton, however, is the interpretation of process $p_3 \equiv \{\!|x|\!\}\,((x > 2) \mapsto a;\mathbf{0} + (x \leq 2) \mapsto b;\mathbf{0})$. According to our intentions, this should be different from p_2. Notice that the rightmost x is free in p_2 but bound in p_3.

Similarly, an attempt to define the timed automaton for process

$$p_4 \equiv \{\!|x|\!\}\,(x > 2) \mapsto a;\mathbf{0} \;||_A\; (x \leq 2) \mapsto b;\mathbf{0}$$

would induce a capture of the rightmost clock x.

We give a dynamic characterisation of processes which do not have conflict of variables. We do so in two steps.

Definition 5.4. A process $p \in \heartsuit$ has a *local conflict of (clock) variables* if $\mathsf{cv}(p)$ can be proven using the rules in Table 5.4. Otherwise, we say that p is *locally free of conflict* and this is denoted by $\neg\mathsf{cv}(p)$. □

We call it "local" since the predicate does not evaluate future states; notice that it is never the case that $\mathsf{cv}(a;p)$, regardless whether p has conflict or not. Since we are going to define the idea of conflict-freedom dynamically, predicate cv suffices. As a matter of fact, to consider a process conflict-free, we check that none of its derivatives has a local conflict.

$$\frac{var(\phi) \cap \kappa(p) \neq \varnothing}{\mathsf{cv}(\phi \mapsto p)} \qquad \frac{var(\psi) \cap \kappa(p) \neq \varnothing}{\mathsf{cv}(\psi \gg p)}$$

$$\frac{\mathsf{cv}(p) \quad \mathsf{op} \in \{\phi \mapsto \_, \{\!|C|\!\}\,\_, \psi \gg \_, \_[f]\}}{\mathsf{cv}(\mathsf{op}(p))} \qquad \frac{\mathsf{cv}(p) \quad X = p}{\mathsf{cv}(X)}$$

$$\frac{\kappa(p) \cap fv(q) \neq \varnothing \quad \oplus \in \{+, ||_A, \between_A, |_A\}}{\mathsf{cv}(p \oplus q) \quad \mathsf{cv}(q \oplus p)} \qquad \frac{\mathsf{cv}(p) \quad \oplus \in \{+, ||_A, \between_A, |_A\}}{\mathsf{cv}(p \oplus q) \quad \mathsf{cv}(q \oplus p)}$$

Table 5.4: Conflict of clock variables in $\heartsuit$

Definition 5.5. We say that $p \in \heartsuit$ is *conflict-free* (or *does not have conflict of variables*) if for every sequence

$$p \equiv p_0 \xrightarrow{a_1,\phi_1} p_1 \xrightarrow{a_2,\phi_2} p_2 \cdots p_{n-1} \xrightarrow{a_n,\phi_n} p_n,$$

$\neg \mathsf{cv}(p_i)$ for all $i \in \{0, \ldots, n\}$. Otherwise, we say that p *has conflict of variables*. We say that a recursive specification is *conflict-free* (or *does not have conflict of variables*) if every process variable it defines is conflict-free. □

To give some examples, notice that none of the above defined processes p_1, p_2 and p_4 is conflict-free. In particular, notice that $\neg \mathsf{cv}(p_1)$, but $p_1 \xrightarrow{a,\mathbf{tt}} (x > 2) \mapsto \{\!|x|\!\}\, (x \leq 2) \mapsto b; \mathbf{0} \equiv p_1'$ and $\mathsf{cv}(p_1')$.

We are now ready to define the semantics of a ♡ term.

Definition 5.6. Let $\heartsuit^{\mathsf{cf}} \stackrel{\text{def}}{=} \{p \in \heartsuit \mid p \text{ is conflict-free } \}$. The semantics of ♡ is defined according to the timed automaton $TA(\heartsuit^{\mathsf{cf}}) = (\heartsuit^{\mathsf{cf}}, \mathcal{A}, \mathcal{C}, \longrightarrow, \iota, \kappa)$ where $\mathcal{A}$ and $\mathcal{C}$ are as in Definition 5.1 and $\longrightarrow$, ι, and κ are the least relations satisfying the rules in Table 5.3. The semantics of a process $p \in \heartsuit$ is defined only if it is conflict-free. In that case it is given by the rooted timed automaton $(TA(\heartsuit^{\mathsf{cf}}), p)$. □

Example 5.7. It is not difficult to check that the timed automaton defined by the process *System* in Example 5.2 is the one depicted in Figure 4.1. □

5.3 Timed Automata up to α-conversion

The intention of Definition 5.6 is to give a straight semantics to ♡. However, this is at the cost of excluding some terms, namely those that have conflict of variables. In this section we enlarge the class of terms to which semantics can be given. Such a class includes all processes defined by guarded recursion.

Capture of variables is a well known problem in languages with variables. It can be solved by considering terms modulo α-congruence. This is the solution proposed in the λ-calculus (Barendregt, 1984; Stoughton, 1988), and adopted by the π-calculus (Milner et al., 1992; Milner, 1993) in the context of a process calculus with variables. To define α-congruence we first need to introduce the notion of substitution in ♡.

Notation. Let $C, C' \subseteq \mathcal{C}$. We use the notation $\xi_{\{C'/C\}}$ to represent a surjective (total) function in $C \to C'$. In particular, if it is a bijection, we denote it by $\xi_{[C'/C]}$, and if $C' = \{x\}$ we write $\xi_{\{x/C\}}$

Given a function $\xi : \mathcal{C} \to \mathcal{C}$, we define the function $\xi\, \xi_{\{C'/C\}}$ as follows

$$\xi\, \xi_{\{C'/C\}}(x) \stackrel{\text{def}}{=} \begin{cases} \xi_{\{C'/C\}}(x) & \text{if } x \in C \\ \xi(x) & \text{otherwise} \end{cases}$$

We denote by id the identity function, and we just write $\xi_{\{C'/C\}}$ (or $\xi_{[C'/C]}$, if it applies) instead of $id\, \xi_{\{C'/C\}}$. The overloading in the use of $\xi_{\{C'/C\}}$ and $\xi_{[C'/C]}$ should be harmless. □

$\xi(\mathbf{0}) = \mathbf{0}$	
$\xi(\phi \mapsto p) = \xi(\phi) \mapsto \xi(p)$	
$\xi(\phi \gg p) = \xi(\psi) \gg \xi(p)$	
$\xi(\{\!\{C\}\!\}\, p) = \{\!\{C'\}\!\}\, \xi\, \xi_{[C'/C]}(p)$	$(C' \cap \xi(fv(p) - C) = \varnothing)$
$\xi(\mathsf{op}(p)) = \mathsf{op}(\xi(p))$	$(\mathsf{op} \in \{a;\_, \_[f]\})$
$\xi(p \oplus q) = \xi(p) \oplus \xi(q)$	$(\oplus \in \{+, \|\|_A, \between_A, \|_A\})$
$\xi(X) = X_\xi$	(provided $X = p$ and $X_\xi = \xi(p)$)

Table 5.5: Substitution in ♡

Definition 5.8. As in Definition 4.15, a *substitution* is a function $\xi : \mathcal{C} \to \mathcal{C}$ that can be extended to constraints. We extend it to sets in the usual way: $\xi(C) = \{\xi(x) \mid x \in C\}$. We also extend it to ♡ terms recursively as stated in Table 5.5. □

Notice that the substitution on a process variable X is defined in terms of a new process variable X_ξ whose definition is like X's but with the clock variables substituted according to ξ.

A term α-converts to another if it has the same syntactic form with the exception that bound clocks might be renamed together with its binder.

Definition 5.9. We define the relation $\simeq_\alpha \subseteq ♡ \times ♡$ as the least relation satisfying rules in Table 5.6. We call $\simeq_\alpha$ *α-congruence*, and if $p \simeq_\alpha q$, we say that p and q are *α-congruent*, or that one can *α-convert* to the other. □

By induction on the depth of the proof tree of $\simeq_\alpha$, it can proved that it is an equivalence relation. The fact that for every context $\mathbf{C}[\_]$, $p \simeq_\alpha q$ implies $\mathbf{C}[p] \simeq_\alpha \mathbf{C}[q]$ should be clear from the definition. As a consequence, we can state:

Proposition 5.10. *$\simeq_\alpha$ is a congruence.*

The following proposition can be straightforwardly proved by induction on the depth of the proof tree.

Proposition 5.11. *$p \simeq_\alpha q$ implies $fv(p) = fv(q)$.*

Every ♡ process defined with process variables in a guarded recursive specification can be α-converted into a new one with no local conflict of variables. What we have just stated is a consequence of the following theorem.

$$\mathbf{0} \simeq_\alpha \mathbf{0} \qquad \frac{X \in \mathcal{V}}{X \simeq_\alpha X} \qquad \frac{p \simeq_\alpha q \quad X = p \quad Y = q}{X \simeq_\alpha Y}$$

$$\frac{\xi_{[C^\star/C]}(p) \simeq_\alpha \xi_{[C^\star/C']}(q) \quad C^\star \cap (\ (fv(p) - C) \cup (fv(q) - C')\ ) = \varnothing}{\{\!|C|\!\}\, p \simeq_\alpha \{\!|C'|\!\}\, q}$$

$$\frac{p \simeq_\alpha q \quad \mathsf{op} \in \{a;\_, \phi \mapsto \_, \psi \gg \_, \_[f]\}}{\mathsf{op}(p) \simeq_\alpha \mathsf{op}(q)} \qquad \frac{p \simeq_\alpha q \quad p' \simeq_\alpha q' \quad \oplus \in \{+, ||_A, \between_A, |_A\}}{p \oplus p' \simeq_\alpha q \oplus q'}$$

Table 5.6: α-conversion in ♡

Theorem 5.12. *Let $p \in$ ♡ be a guarded process. Then there is a $q \in$ ♡ such that $\neg\mathsf{cv}(q)$ and $p \simeq_\alpha q$.*

Proof sketch. More generally, using structural induction we prove that for every guarded p and every $C \subseteq \mathcal{C}$, there is a q such that $p \simeq_\alpha q$, $\neg\mathsf{cv}(q)$, and $\kappa(q) \cap C = \varnothing$.

The idea is that C will carry along the proof tree of $p \simeq_\alpha q$ the possible variables that may introduce conflict. As an example, we give the inductive case of $\phi \mapsto p$. By induction hypothesis there is a q such that $p \simeq_\alpha q$, $\neg\mathsf{cv}(q)$, and $\kappa(q) \cap C' = \varnothing$ for any given $C' \subseteq \mathcal{C}$. In particular, take $C' = C \cup var(\phi)$. Hence $\phi \mapsto p \simeq_\alpha \phi \mapsto q$, $\neg\mathsf{cv}(\phi \mapsto q)$ (since $\neg\mathsf{cv}(q)$ and $\kappa(q) \cap var(\phi) = \varnothing$), and $\kappa(q) \cap C = \varnothing$. *Q.E.D.*

The theorem does not hold in general. The unguardedly defined process X, with $X = \{\!|x|\!\}\, (a; \mathbf{0} + X) \,||_\varnothing\, (x \leq 1) \gg X$, cannot be α-converted into a process locally free of conflict since it would require an infinite proof tree to do so.

The solution that we propose to give semantics to any ♡ term is similar to the approach proposed in π-calculus and its derived calculi (such as the fusion calculus (Parrow and Victor, 1998; Victor, 1998)), although in our case, special care is needed. The new semantics will be defined by the previous rules plus a new one:

$$\textbf{alpha} \quad \frac{p \xrightarrow{a,\phi} p' \quad p' \simeq_\alpha q' \quad \neg\mathsf{cv}(p) \quad \neg\mathsf{cv}(q')}{p \xrightarrow{a,\phi} q'}$$

What rule **alpha** does is to keep the moves within processes locally free of conflict. So, as soon as a process p moves to a process p' that has local conflict of variables, rule **alpha** finds an α-congruent q' which does not, and uses it to simulate the move[1]. Thus, we also need to prune those edges whose target is a locally conflictive process since they take us to an undesired dead end. So, the new semantics defines a new edge $\longrightarrow'$ such that $p \xrightarrow{a,\phi}{}' q$ if and only if $p \xrightarrow{a,\phi} q$ and p and q are locally free of conflict.

[1] The majority of the papers discussing π-calculus or related calculi do not explicitly give a rule, but it is simply stated that the operational semantics is defined up to α-conversion. An exception is (Sangiorgi,

Since there are terms that cannot be appropriately α-converted into another term that is free of local conflict we need to rule them out.

Definition 5.13. A process $p \in \heartsuit$ is *α-conflict-free* if there is a $p_0 \in \heartsuit$ such that $p \simeq_\alpha p_0$, $\neg\mathsf{cv}(p_0)$, and for every $n > 0$, whenever

$$p_0 \xrightarrow{a_0,\phi_0} p_1 \xrightarrow{a_1,\phi_1} p_2 \cdots p_n \xrightarrow{a_n,\phi_n} p_{n+1},$$

and $\neg\mathsf{cv}(p_i)$, $0 < i \leq n$, then there is a $q \in \heartsuit$ such that $\neg\mathsf{cv}(q)$ and $p_{n+1} \simeq_\alpha q$. We remark that $\longrightarrow$ is calculated as the least relation satisfying rules in Table 5.3 and **alpha**. □

In other words, a term is α-conflict-free if, after been appropriately α-converted it never reaches a term that cannot be α-converted into a locally conflict-free process.

Definition 5.14. Let $\heartsuit^\alpha \stackrel{\text{def}}{=} \{p \in \heartsuit \mid \neg\mathsf{cv}(p)$ and p is α-conflict-free$\}$. The *semantics of $\heartsuit$ up to α-conversion* is defined by the timed automaton $TA(\heartsuit^\alpha) = (\heartsuit^\alpha, \mathcal{A}, \mathcal{C}, \longrightarrow', \iota, \kappa)$ where $\mathcal{A}$ and $\mathcal{C}$ are as in Definition 5.1, ι, and κ are defined by the least relations satisfying the rules in Table 5.3, and $\longrightarrow' \stackrel{\text{def}}{=} \longrightarrow \cap(\heartsuit^\alpha \times \mathcal{A} \times \Phi \times \heartsuit^\alpha)$ with $\longrightarrow$ been the least relation satisfying rules in Table 5.3 and rule **alpha** above.

The semantics up to α-conversion of a process $p \in \heartsuit$ is defined whenever p is α-conflict-free by any rooted timed automaton $(TA(\heartsuit^\alpha), q)$, provided $p \simeq_\alpha q \in \heartsuit^\alpha$. □

By Theorem 5.12 it follows that guarded processes are guaranteed to have semantics up to α-conversion.

Proposition 5.15. *If every process variable appearing in $p \in \heartsuit$ is defined using guarded recursion, then the semantics of p up to α-conversion is defined.*

At this point, it should be clear that α-congruence does not preserve structural bisimulation. (The distracted reader may check Figure 5.2.) This may rise the question whether Definition 5.14 is indeed a good definition. In the next theorem we state that α-congruence preserves symbolic bisimulation.

Theorem 5.16. *Let $p, q \in \heartsuit^\alpha$; so, none of them has local conflict of variables and moreover they are α-conflict-free. If $p \simeq_\alpha q$ then $p \sim_{\&} q$ where p and q should be interpreted here as states in $TA(\heartsuit^\alpha)$.*

The proof of this theorem as well as the one of Theorem 5.17 below can be found in Appendix B.2. As a corollary we have that the semantics up to α-conversion is well defined.

1996) that explicitly gives the rule

$$\frac{p \xrightarrow{a} p' \quad p \simeq_\alpha q}{q \xrightarrow{a} p'}.$$

Notice that we could not use a similar rule, since $p \simeq_\alpha q$ *does not* imply $\kappa(p) = \kappa(q)$, and as a consequence variables that should have been bound, suddenly would become free.

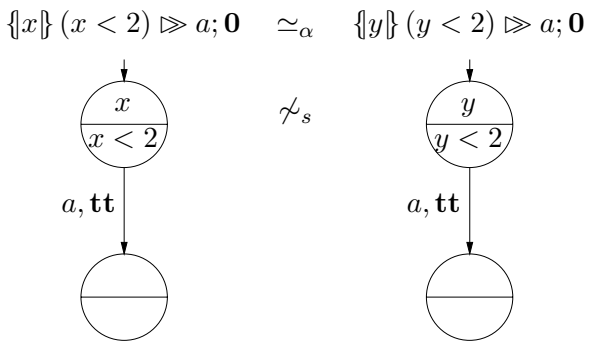


Figure 5.2: α-congruence does not preserve structural bisimulation

Corollary 5.16.1. *For every $p \in \heartsuit^{\alpha}$, the semantics of p up to α-conversion is unique up to symbolic bisimulation, i.e., if $p \simeq_{\alpha} q$, $p \simeq_{\alpha} q'$, and q and q' are locally free of conflict, then $(TA(\heartsuit^{\alpha}), q)$ and $(TA(\heartsuit^{\alpha}), q')$ are symbolically bisimilar.*

The following theorem states that the original semantics given in Definition 5.6 coincides, modulo symbolic bisimulation, with the semantics up to α-conversion.

Theorem 5.17. *Let $p \in \heartsuit^{\mathsf{cf}}$; so, p is conflict-free. Then, the rooted timed automata $(TA(\heartsuit^{\mathsf{cf}}), p)$ and $(TA(\heartsuit^{\alpha}), p)$ are symbolically bisimilar.*

5.4 Representability

♡ has the property to represent any timed automaton. That is, every timed automaton can be expressed by a ♡ process p such that the semantics of p is isomorphic to the original timed automaton. Moreover, such process p is conflict-free.

To represent the generality of timed automata we need to define a generalised summation. Let P be a denumerable and ordered set of ♡ processes having the form $\phi \mapsto a; p$. We define the process variables X^{Σ}_{P} as follows

$$X^{\Sigma}_{\varnothing} = \mathbf{0} \qquad X^{\Sigma}_{\{\phi \mapsto a;p\} \cup P} = \phi \mapsto a; p + X^{\Sigma}_{P}$$

The *generalised summation* $\sum P$ is defined by the process variable X^{Σ}_{P}. It should be clear that

$$\kappa(\textstyle\sum P) = \varnothing, \quad \iota(\sum P) = \mathbf{tt}, \quad \text{and} \quad \sum P \xrightarrow{a,\phi} p \quad \text{whenever } \phi \mapsto a; p \in P$$

If $P = \{\phi_i \mapsto a_i; p_i \mid i \in I\}$ for some index set I, we write $\sum_{i \in I} \phi_i \mapsto a_i; p_i$. If $P = \{p_1, \ldots, p_n\}$ is finite, $\sum P$ is simply a shorthand notation for $p_1 + \cdots + p_n$.

Notice that the semantics of a term p is defined by the rooted timed automaton $(TA(\heartsuit^{\mathsf{cf}}), p)$. Since the set of locations is $\heartsuit^{\mathsf{cf}}$ the timed automaton includes much too

many locations. For that reason we consider only the reachable part. A state s is *(symbolically) reachable* if there is a sequence of edges from the initial state s_0 to s, i.e., there are $a_1, \ldots, a_n$, $\phi_1, \ldots, \phi_n$ and $s_1, \ldots, s_n$ ($n \geq 0$) such that $s_0 \xrightarrow{a_1,\phi_1} s_1 \cdots \xrightarrow{a_n,\phi_n} s_n = s$. The *reachable part* of a rooted timed automaton (TA, s_0) is the same rooted timed automaton restricted to the set of states that are reachable. Notice that we are considering a static view that differs from the usual notion of reachability in timed automata theory (compare to Alur et al., 1993).

We are now ready to state and prove our theorem.

Theorem 5.18. *For every rooted timed automaton (TA, s_0) with denumerable branching —i.e., for every location s the set $\{s \xrightarrow{a,\phi} s' \mid a \in \mathcal{A} \wedge \phi \in \Phi \wedge s' \in \mathcal{S}\}$ is denumerable— there is a conflict-free recursive specification E with root X_{s_0} such that the reachable part of (TA, s_0) and the reachable part of $(TA(\heartsuit^{\mathsf{cf}}), X_{s_0})$ are isomorphic.*

Proof. The proof consists of associating a process variable to each state s of TA and defining each one of them as the term that resets the clocks of $\kappa(s)$ and has an invariant $\iota(s)$ over the summation of the outgoing edges represented by prefixes with their respective guard. Thus, the isomorphism is given by the function that maps every state on to its corresponding variable.

Let $TA = (\mathcal{S}, \mathcal{A}, \mathcal{C}, \rightarrow', \iota', \kappa')$. For each state $s \in \mathcal{S}$ define a different variable X_s. Let $\mathcal{V}_\mathcal{S}$ be the set of such variables. Define the set of recursive specifications E_{TA} with root X_{s_0} and recursive equations

$$X_s = \{\!|\kappa'(s)|\!\}\, \iota'(s) \gg \left(\sum \{\phi \mapsto a; X_{s'} \mid s \xrightarrow{a,\phi}{}' s'\}\right).$$

By construction, it is evident that E is conflict-free. Define

$$TA(\heartsuit^{\mathsf{cf}}) \upharpoonright \mathcal{V}_\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{V}_\mathcal{S}, \mathcal{A}, \mathcal{C}, \rightarrow\!\upharpoonright \mathcal{V}_\mathcal{S}, \iota \upharpoonright \mathcal{V}_\mathcal{S}, \kappa \upharpoonright \mathcal{V}_\mathcal{S})$$

where:

$$\begin{aligned}
\rightarrow\!\upharpoonright \mathcal{V}_\mathcal{S} &\stackrel{\text{def}}{=} \rightarrow \cap (\mathcal{V}_\mathcal{S} \times \mathcal{A} \times \Phi(\mathcal{C}) \times \heartsuit^{\mathsf{cf}}) \\
\iota \upharpoonright \mathcal{V}_\mathcal{S} &\stackrel{\text{def}}{=} \iota \cap (\mathcal{V}_\mathcal{S} \times \overline{\Phi}) \\
\kappa \upharpoonright \mathcal{V}_\mathcal{S} &\stackrel{\text{def}}{=} \kappa \cap (\mathcal{V}_\mathcal{S} \times \wp(\mathcal{C}))
\end{aligned}$$

Clearly, $\mathfrak{I} : S \to \mathcal{V}_\mathcal{S}$ defined as $\mathfrak{I}(s) \stackrel{\text{def}}{=} X_s$ for all $s \in S$, is an isomorphism, which implies the theorem straightforwardly. *Q.E.D.*

Example 5.19. Take the automaton of Example 4.3. Using the technique in the proof we can construct the recursive specification

$$\begin{aligned}
X_{s_0} &= \{\!|\varnothing|\!\}\, \mathbf{tt} \gg \mathbf{tt} \mapsto \mathit{on}; X_{s_1} \\
X_{s_1} &= \{\!|x|\!\}\, (x \leq 2) \gg (\ \mathbf{tt} \mapsto \mathit{on}; X_{s_1} \\
&\qquad\qquad + (x = 2) \mapsto \mathit{off}; X_{s_0})
\end{aligned}$$

The reader is invited to check that $(TA(\heartsuit^{\mathsf{cf}}) \upharpoonright \{X_{s_0}, X_{s_1}\}, X_{s_0})$ is isomorphic to the automaton of Example 4.3. □

By combining Definition 5.14 and Theorem 5.18 we immediately have that any ♡ term which is α-conflict-free has a symbolically bisimilar term that is conflict-free.

Corollary 5.18.1. *For every α-conflict-free $p \in \heartsuit^{\alpha}$ there is a conflict-free $q \in \heartsuit^{\mathsf{cf}}$ such that $p \sim_{\&} q$.*

From Theorem 5.18 and its proof, we have the following corollary.

Corollary 5.18.2. *In the conditions of Theorem 5.18, the recursive specification E_{TA} resulting from the definition in the proof is guarded if the timed automaton TA is finitely branching, and it is regular if TA is finite.*

Reciprocally, any conflict-free guarded specification defines a finitely branching timed automaton. Similarly, any conflict-free regular specification defines a regular timed automaton. This should be straightforward for the reader learned in process algebras. In fact the condition of "conflict-free" can be dropped but its proof requires some algebraic manipulation. We refer to Chapter 7, Section 7.3.

5.5 Congruences

Assume that $\overline{\mathsf{ck}}$ is also an operation of the language. Under this condition, it is not difficult to observe that the rules given in Tables 5.3, B.2, and B.3, plus a suitable modification of rules for $\overline{\mathsf{ck}}$ (see rules (5.1), page 52) are all in *path* format. As a consequence, structural bisimulation is a congruence for ♡ operations, according to (Baeten and Verhoef, 1993). Since in addition none of the rules have look-ahead, by (Rensink, 1997), structural bisimulation is also a congruence for recursion (see Chapter 2, page 20).

Theorem 5.20. *Let p and q be two ♡ terms such that $p \sim_s q$. For any ♡ context $\mathbf{C}[\ ]$, it holds that $\mathbf{C}[p] \sim_s \mathbf{C}[q]$. Moreover, $\sim_s$ is also a congruence for recursion in ♡ (see Theorem 2.13 for an appropriate formulation).*

Symbolic bisimulation is also a congruence for ♡ operations. Its proof is much more involved, and since our interest is rather on timed bisimulation we will omit it.

Theorem 5.21. *Let $p, q \in \heartsuit^{\alpha}$ such that $p \sim_{\&} q$. For any ♡ context $\mathbf{C}[\ ]$ such that $\mathbf{C}[p]$ and $\mathbf{C}[q]$ are α-conflict-free, it holds that $\mathbf{C}[p] \sim_{\&} \mathbf{C}[q]$.*

As for timed automata, and like structural and symbolic bisimulation, timed bisimulation also extends naturally to ♡: $p \sim_t q$ if there are p' and q' such that they are α-conflict-free, $p \simeq_{\alpha} p'$, $q \simeq_{\alpha} q'$, and $(p', v) \sim_t (q', v)$ for all valuation v, where (p', v) and (q', v) are states in the timed transition system obtained from the semantics of the timed automaton $TA(\heartsuit^{\alpha})$ (or equivalently $TA(\heartsuit^{\mathsf{cf}})$). Timed bisimulation is also a congruence for ♡ operations. The proof of the following theorem is given in Appendix B.4.

Theorem 5.22. *Let p and q be two α-conflict-free ♡ terms such that $p \sim_t q$. Let $\mathbf{C}[\ ]$ be a ♡ context such that $\mathbf{C}[p]$ and $\mathbf{C}[q]$ are α-conflict-free. Then, it holds that $\mathbf{C}[p] \sim_t \mathbf{C}[q]$.*

$$\mathcal{U}_d\langle \mathbf{0}, v\rangle \qquad \mathcal{U}_d\langle a; p, v\rangle \qquad \frac{\mathcal{U}_d\langle p, v\rangle}{\mathcal{U}_d\langle \phi \mapsto p, v\rangle} \qquad \frac{\models (v+d)(\psi) \quad \mathcal{U}_d\langle p, v\rangle}{\mathcal{U}_d\langle \psi \gg p, v\rangle}$$

$$\frac{\mathcal{U}_d\langle p, v[C \hookleftarrow 0]\rangle}{\mathcal{U}_d\langle \{\!|C|\!\}\, p, v\rangle} \qquad \frac{\mathcal{U}_d\langle p, v\rangle}{\mathcal{U}_d\langle p+q, v\rangle} \qquad \frac{\mathcal{U}_d\langle p, v\rangle}{\mathcal{U}_d\langle q+p, v\rangle} \qquad \frac{\mathcal{U}_d\langle p, v\rangle}{\mathcal{U}_d\langle X, v\rangle}\ X = p$$

$$\langle a; p, v\rangle \xrightarrow{a(d)} \langle p, v+d\rangle \qquad \frac{\models (v+d)(\phi) \quad \langle p, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle}{\langle \phi \mapsto p, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle}$$

$$\frac{\langle p, v[C \hookleftarrow 0]\rangle \xrightarrow{a(d)} \langle p', v'\rangle}{\langle \{\!|C|\!\}\, p, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle} \qquad \frac{\models (v+d)(\psi) \quad \langle p, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle}{\langle \psi \gg p, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle}$$

$$\frac{\langle p, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle}{\langle p+q, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle} \qquad \frac{\langle p, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle}{\langle q+p, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle} \qquad \frac{\langle p, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle}{\langle X, v\rangle \xrightarrow{a(d)} \langle p', v'\rangle}\ X = p$$

Table 5.7: An operational semantics for the basic language

5.6 An Operational Semantics for the Basic Language

In this section we give a semantics for ♡ directly in terms of timed transition systems. We show that it coincides (modulo timed bisimulation) with the semantics via the timed automaton.

Let $\heartsuit^{br}$ be the subset of ♡ terms formed by the basic operations $\mathbf{0}$, $a;\_$, $+\ \phi \mapsto \_$, $\psi \gg \_$, and $\{\!|C|\!\}\ \_$ as well as recursion. (The superindex *br* stands for basic terms with recursion.) A direct operational semantics for this sub-language in terms of timed transition system is given in Table 5.7.

Definition 5.23. Let $\mathbf{V}$ be the set of all clock valuations. The direct semantics of $\heartsuit^{br}$ is defined by the timed transition system $TTS(\heartsuit^{br}) = (\heartsuit^{br} \times \mathbf{V}, \mathcal{A} \times \mathbb{R}_{\geq 0}, \longrightarrow, \mathcal{U})$ where $\mathcal{A}$ is as in Definition 5.1 and $\longrightarrow$ and $\mathcal{U}$ are the least relations satisfying rules in Table 5.7. The direct semantic of each process $p \in \heartsuit^{br}$ in the initial valuation $v_0 \in \mathbf{V}$ is defined by the rooted timed transition system $(TTS(\heartsuit^{br}), \langle p, v_0\rangle)$. □

The direct semantics of $\heartsuit^{br}$ is well defined, that is, it satisfies axioms **Until** and **Delay** in Definition 3.1. This fact can be proven by straightforward induction on the depth of the proof tree.

Rules in Table 5.7 express the concrete behaviour of each term. In this case the execution of a transition or the idling time is made explicit instead of been coded into clock

constraints. Thus, for instance, process $\phi \mapsto p$ can actually perform any action a that p can perform at time d in the valuation v whenever the condition ϕ holds in v after d units of time has passed; or, for example, process $\psi \gg p$ can idle d units of times in a valuation v if p also can idle d units of time, and moreover, condition ψ holds in v after idling d units of time.

So far we stated two ways of interpreting terms in $\heartsuit^{br}$. We have just given a semantics in terms of timed transition systems. Alternatively, a timed transition system could be associated to every term p in $\heartsuit^{br}$ in two steps, namely, by obtaining the semantics in term of timed automata —either using Definition 5.6, or obtaining a timed automaton up to α-conversion with Definition 5.14— and then interpreting it in terms of a timed transition systems, according to Definition 4.6. The natural question is whether both semantics are equivalent. Theorem 5.24 states that both way of interpreting a process are equivalent up to timed bisimulation.

Theorem 5.24. *For every α-conflict-free $p \in \heartsuit^{br}$, and for every closed valuation v_0, $\langle p, v_0\rangle \sim_t (p', v_0)$, where $p' \simeq_\alpha p$ and $\neg \mathsf{cv}(p)$. Here, $\langle p, v_0\rangle$ is a state in the timed transition system $TTS(\heartsuit^{br})$ from the direct semantics, and (p', v_0) is a state in the "two steps" semantics $TTS(TA(\heartsuit^{\alpha}))$.*

Proof. We state that

$$R \stackrel{\text{def}}{=} \left\{ \langle \langle p, v_1\rangle, (p, v_2)\rangle \mid p \in \heartsuit^{br} \wedge \neg \mathsf{cv}(p) \wedge v_1 \restriction fv(p) = v_2 \restriction fv(p) \right\}^{\mathsf{s}},$$

with $v \restriction C \stackrel{\text{def}}{=} v \cap (C \times \mathbb{R}_{\geq 0})$, is a timed bisimulation up to $\sim_t$. This suffices to prove the theorem.

The proof makes use of the fact that $\simeq_\alpha$ preserves symbolic bisimulation in $TA(\heartsuit^{\alpha})$, as it is stated by Theorem 5.16, and that $\simeq_\alpha$ also preserves bisimulation in $TTS(\heartsuit^{br})$. This last fact is proved in (D'Argenio and Brinksma, 1996a). The complete proof of this theorem can be found in Appendix B.3. *Q.E.D.*

Like for timed automata, the notion of timed bisimulation can be extended to terms in $\heartsuit^{br}$ under the semantics given above. Thus, p and q are timed bisimilar, notation $p \sim_t q$, if for all $v_0 \in \mathbf{V}$, $\langle p, v_0\rangle \sim_t \langle q, v_0\rangle$.

An interesting fact is that, under this semantics, timed bisimulation is a congruence for *any* sequential $\heartsuit$ term, and the condition of α-conflict-free required by Theorem 5.22 can be dropped. This arises as a consequence of the generality of the semantics which do not impose any condition on the construction of the terms. (Compare Definitions 5.6, 5.14, and 5.23.) Thus, we can state the following theorem whose proof can be found in Appendix B.4.

Theorem 5.25. *Let $p, q \in \heartsuit^{br}$, such that $p \sim_t q$. Let $\mathbf{C}[\,]$ be a context built using operators $\mathbf{0}$, $a;\_$, $\phi \mapsto \_$, $\psi \gg \_$, $\{\!|C|\!\}\,\_$, $+$, and process variables. Then, it holds that $\mathbf{C}[p] \sim_t \mathbf{C}[q]$. Moreover, $\sim_t$ is also a congruence for recursion in $\heartsuit^{br}$ (see Theorem 2.13 for an appropriate formulation).*

5.7 Summary

In Figure 5.3, we summarise this chapter. Black arrows should be read as "can be interpreted in terms of". Thus, an α-conflict-free term p can be interpreted in terms of timed automata using the semantics up to α-conversion. If p is in addition conflict-free, we can directly use the semantics in terms of timed automata without any need to α-convert. One way or the other, the obtained automata are equivalent up to symbolic bisimulation. So, if we go further and give semantics to the obtained timed automata, the underlying timed transition systems are timed bisimilar. This is an immediate consequence of the fact that symbolic bisimilarity implies timed bisimilarity as it was stated in Chapter 4.

Alternatively, *any* sequential $\heartsuit$ term, i.e., any term not containing the parallel operator, has a direct semantics in terms of timed transition systems. This semantics is equivalent to previous ones up to timed bisimulation.

The grey arrow indicates that any (denumerably branching) timed automaton can be represented by a $\heartsuit$ term. Notice that such a term will always be a sequential and conflict-free $\heartsuit$ term, that is, a term belonging to the intersection of all the sub-languages we discussed along the chapter.

Unfortunately, although any timed transition system can be translated into a timed automaton by allowing uncountable branching, not every timed transition system can be translated into $\heartsuit$. To do so, uncountably branching summation should instead be allowed, but such a summation cannot be constructed using the language given above, as it is the case for a denumerable summation (see Section 5.4).

5.8 Related Work

During the last decade, many well-known process algebras have been extended with features to manipulate time (e.g. Davies et al., 1992; Yi, 1990, 1991; Moller and Tofts, 1990; Nicollin and Sifakis, 1994; Baeten and Bergstra, 1991; Klusener, 1991, 1993; Baeten and Bergstra, 1995; Bolognesi and Lucidi, 1992; Leduc and Léonard, 1994). However not all of them were studied in relation to timed automata. In the following we enumerate the works that relate a process algebra or a language to timed automata

- Nicollin et al. (1992, 1993) gave an interpretation of ATP (Nicollin and Sifakis, 1994) in terms of timed automata, considering a dense time domain. Yovine (1993) showed that such a translation preserves timed branching bisimulation. ATP is basically an extension of CCS (Milner, 1989) including a timeout operation, an execution delay or *watchdog* operation and the notion of urgent actions. Neither clocks nor time variables are considered in ATP. Basically the same study was done by Daws et al. (1994) for ET-LOTOS (Leduc and Léonard, 1994). In this case, also timed branching bisimulation is shown to be preserved. In neither of these works an inverse study was carried out, i.e., to express a timed automata in terms of the process algebra.

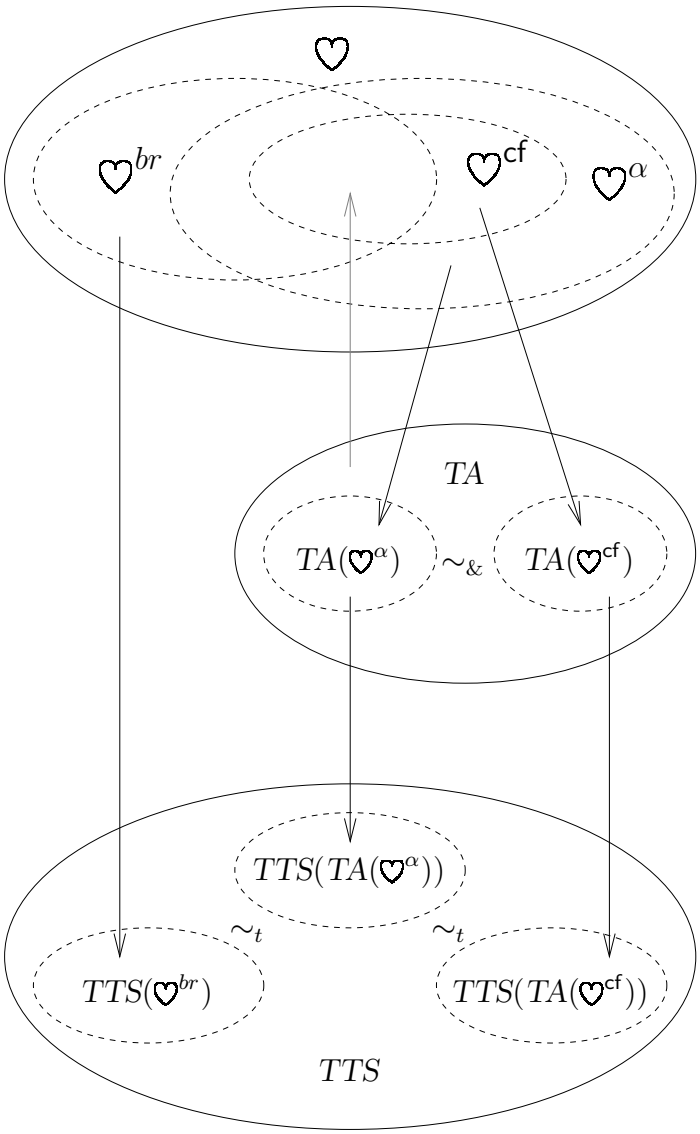


Figure 5.3: Summary of ♡ semantics

- Fokkink (1993, 1994a) sketched an interpretation of ACP with prefix integration (Klusener, 1991, 1993; Baeten and Bergstra, 1991) into timed automata. Moreover, the class of strongly regular processes and timed automata turn out to be equivalent when certain restrictions (namely, non-Zenoness and fairness) are imposed in the behaviour of the timed automata. Thus ACP with prefix integration is more expressive than timed automata. For instance, consider the (finite!) ACP process $\int_{v<1} a[v] \cdot \int_{w=v} b[w] \cdot \mathbf{0}$, that *records* in v the time when a was performed, and after v units of time executes b. In our language, an unguarded recursive expression would be needed to define this process if the time domain were denumerable. If instead the set of real numbers is considered, such a process cannot be expressed. However, if we had more expressive constraints by allowing arbitrary comparison between clocks, we could define $\{\!|x|\!\}\,(x < 1) \gg a; (\{\!|y|\!\}\,(2y \leq x) \gg (2y = x) \mapsto b; \mathbf{0})$. Such an extension would, of course, affect the tractability of the language.

- Lynch and Vaandrager (1996) introduced a language that explicitly manages clocks. Such a language has the same expressive power as timed automata with respect to (weak) timed trace equivalence.

- Alur and Henzinger (1994) studied the extension of programming languages with clock variables. They discuss their semantics in terms of the so called *real-time programs* which are easily translated into timed automata (see Henzinger et al., 1994).

- Yi et al. (1994) gave an algebra that represents timed automata without invariants. Later, Paul Pettersson (1999) extended it to include invariants as well. Basically, the algebra is a syntax for the timed automata including CCS parallel composition and restriction. The invariant operation is handled in the same way we do, but the prefixing operation integrates the guard and the clock resetting as well. It has the form $(\phi, a, C).p$ with $\phi \in \Phi$ and $C \subseteq \mathcal{C}$. It can be understood as our term $\phi \mapsto a; \{\!|C|\!\}\,p$. Thus, since terms with conditions in their first actions unavoidably become open terms, it is necessary to consider an initial valuation in its semantics for which $[\mathcal{C} \leftarrowtail 0]$ is taken.

- In a similar manner to (Nicollin et al., 1992, 1993; Daws et al., 1994), Bradley et al. (1995) gave a translation of AORTA (Bradley et al., 1994) —another CSP/LOTOS-like extension— into timed automata. The translation has been shown to preserve some appropriate equivalence. AORTA does not have the expressive power of timed automata, and hence an inverse translation is not given.

From the above mentioned works only (Lynch and Vaandrager, 1996; Alur and Henzinger, 1994; Yi et al., 1994; Pettersson, 1999) contain variables and constraints in the fashion of timed automata, just like ♡. From these works, (Alur and Henzinger, 1994) is less formal and the relation between the language and timed automata are discussed as recipe rather than as a formal proof system.

Chapter 6

Equational Theory for ♡

Equational theories for timed process algebras have been defined and studied in many occasions (see, e.g, Yi, 1991; Moller and Tofts, 1990; Baeten and Bergstra, 1991; Klusener, 1993; Davies et al., 1992; Fokkink and Klusener, 1995; Groote, 1997). Most of them axiomatise timed bisimulation. Yet, their concern is to study the theory of timed processes in general. The few that have been formally related to timed automata (see Section 5.8) only provide a semantic connection. To our knowledge no axiomatic theory of timed automata has been devised.

In the previous chapter we discussed ♡ as a formal description language whose semantics is defined in terms of timed automata. In this chapter we define an equational theory for ♡. It gives us an axiomatic framework to manipulate timed automata. Thus, large expressions can be reduced by using properties such as the expansion law, and redundant information can be eliminated by, for instance, using axioms that say how to reduce the number of clocks or how to find tighter constraints.

We state soundness and completeness of the axiom system with respect to symbolic bisimulation. We also introduce some useful axioms for timed bisimulation. We first discuss the core algebra with only the basic operations and then we extend it with the static operations. We shortly show that the discussed theories are conservative extensions of the theory for ♧.

6.1 Basic Axioms

In this first section we discuss the equational theory for the sublanguage consisting of the basic operations. So the signature of the language $\heartsuit^b$ we consider, includes only prefixing, nil, summation, guard, clock resetting, and the invariant operation.

The axioms for $\heartsuit^b$ are given in Table 6.1 and can be explained as follows. The choice is commutative (**A1**) and associative (**A2**). Axioms **A3** and **A3′** state a stronger property in the line of idempotency of +. In fact idempotency can be proved using any of these two axioms. **A4** states that **0** is the neutral element for + in the context of unbounded idling, that is when there is no invariant that restricts the progress of time. **Stp** says that a

A1	$p + q = q + p$	
A2	$(p + q) + r = p + (q + r)$	
A3	$\phi \mapsto p + \phi' \mapsto p = (\phi \vee \phi') \mapsto p$	
A3′	$\psi \gg p + \psi' \gg p = (\psi \vee \psi') \gg p$	
A4	$a; p + \mathbf{0} = a; p$	
Stp	$\mathbf{ff} \mapsto a; p = \mathbf{0}$	
L1	$\phi \mapsto p = \phi' \mapsto p$	if $\models \phi \Leftrightarrow \phi'$
L2	$\psi \gg p = \psi' \gg p$	if $\models \psi \Leftrightarrow \psi'$
G0	$\phi \mapsto \mathbf{0} = \mathbf{0}$	
G1	$\mathbf{tt} \mapsto p = p$	
G2	$\phi \mapsto (\phi' \mapsto p) = (\phi \wedge \phi') \mapsto p$	
G3	$\phi \mapsto (\psi \gg p) = \psi \gg (\phi \mapsto p)$	
G4	$\phi \mapsto (\{\!\{C\}\!\}\, p) = \{\!\{C\}\!\}\, (\phi \mapsto p)$	if $var(\phi) \cap C = \varnothing$
G5	$\phi \mapsto (p + q) = \phi \mapsto p + \phi \mapsto q$	
I1	$\mathbf{tt} \gg p = p$	
I2	$\psi \gg (\psi' \gg p) = (\psi \wedge \psi') \gg p$	
I3	$\psi \gg (\{\!\{C\}\!\}\, p) = \{\!\{C\}\!\}\, (\psi \gg p)$	if $var(\psi) \cap C = \varnothing$
I4	$\psi \gg p + \psi \gg q = \psi \gg (p + q)$	
I5	$\psi \gg (\phi \mapsto a; p) + \psi' \gg (\phi' \mapsto b; q)$ $= (\psi \vee \psi') \gg ((\psi \wedge \phi) \mapsto a; p + (\psi' \wedge \phi') \mapsto b; q)$	
CR1	$\{\!\{C\}\!\}\, p = p$	if $C \cap fv(p) = \varnothing$
CR2	$\{\!\{C \cup C' \cup \{x\}\}\!\}\, p = \{\!\{C \cup \{x\}\}\!\}\, \xi_{\{x/C'\}}(p)$	
CR3	$\{\!\{C\}\!\}\, \{\!\{C'\}\!\}\, p = \{\!\{C \cup C'\}\!\}\, p$	
CR4	$\{\!\{C\}\!\}\, p + \{\!\{C\}\!\}\, q = \{\!\{C\}\!\}\, (p + q)$	

D1	$\phi \mapsto a; (\{\!\{y\}\!\}\, p) = \phi \mapsto a; (\{\!\{y\}\!\}\, (x - y \Box d) \gg p)$	if $\models (\phi \Rightarrow (x \Box d))$ and $x \neq y$
D2	$\phi \mapsto a; p = \phi \mapsto a; ((x - y \Box d) \gg p)$	if $\models (\phi \Rightarrow (x - y \Box d))$ where $\Box \in \{\leq, <, \geq, >, =\}$

Table 6.1: Axioms for $\heartsuit^b$

prefixed process guarded by an unsatisfiable condition cannot proceed with its execution. **L1** and **L2** state that processes are equal up to equivalence of constraints. We will in general take these two axioms for granted and never refer to them in our calculations. The way in which guards can be simplified is explained by axioms **G0**–**G5**. Notice that they cannot be eliminated except in the case of **tt**. In particular, axioms **G3**, **G4** and **G5** say how to move invariants, clock resettings and summations out of the scope of a guard. Similarly, axioms **I1**–**I5** explain how to simplify the invariant operation. **I3** says how to take clocks resettings out of the scope of an invariant, while **I4** and **I5** move the invariant out of the scope of a summation. **CR1** and **CR2** eliminate redundant clocks. In particular, **CR2** implies that it is always possible to reduce the number of clocks to be reset to at most one per clock resetting operation. **CR3** gathers all the clocks resettings in only one operation and **CR4** moves clocks out of the scope of a summation. Finally, **D1** and **D2** state that the difference between clocks is invariant and thus it could be "transported" along the execution. In particular, **D1** explains how this difference is stated. Notice that axioms do not necessarily preserve free variables. For instance, **G1** (and **L1**) allows us to prove $(x \geq 0) \mapsto p = p$.

Definition 6.1. The equational theory for $\heartsuit^b$ is defined as follows. The signature contains the constant $\mathbf{0}$, the unary operators $a;\_$, $\phi \mapsto \_$, $\psi \gg \_$, and $\{\!|C|\!\}\,\_$, for every $a \in \mathcal{A}$, $\phi \in \Phi$, $\psi \in \overline{\Phi}$, and $C \subseteq \mathcal{C}$, and the binary operation $+$. The axiom system is given in Table 6.1. We identify with $sig(\heartsuit^b)$ and $ax(\heartsuit^b)$ the signature and the axiom system of $\heartsuit^b$. We also identify with $ax(\heartsuit^b) - \mathbf{D}$ the subset of axioms that does not contain axioms **D1** and **D2**. □

Using the axiom system, some interesting properties can be derived.

Property 6.2. The following equalities can be proved using the axiom system $ax(\heartsuit^b) - \mathbf{D}$.

1.	$p + p = p$	*4.*	$\mathbf{0} + \phi \mapsto a; p = \phi \mapsto a; p$
2.	$\phi \mapsto \phi' \mapsto p = \phi' \mapsto \phi \mapsto p$	*5.*	$\psi \gg p = \psi \gg \psi \mapsto p$
3.	$\psi \gg \psi' \gg p = \psi' \gg \psi \gg p$	*6.*	$\mathbf{ff} \gg \mathbf{0} + p = p$

For the last two properties induction is also necessary. In particular *5* also requires α-conversion.

Proof. As an example we prove property *5*. We use induction on the size of the term p. (Remember that we will omit to refer to axioms **L1** and **L2**.)

Case $\mathbf{0}$.

$$\psi \gg \mathbf{0} \overset{\text{G0}}{=} \psi \gg \psi \mapsto \mathbf{0}$$

Case $a; p$.

$$\psi \gg a; p \overset{\text{G1,A3}}{=} \psi \gg (\mathbf{tt} \mapsto a; p + \mathbf{tt} \mapsto a; p) \overset{\text{I4}}{=} \psi \gg \mathbf{tt} \mapsto a; p + \psi \gg \mathbf{tt} \mapsto a; p$$
$$\overset{\text{I5}}{=} \psi \gg (\psi \mapsto a; p + \psi \mapsto a; p) \overset{\text{A3}}{=} \psi \gg \psi \mapsto a; p$$

Case $\phi \mapsto p$.

$$\psi \gg\!\!\triangleright \phi \mapsto p \overset{\text{G3}}{=} \phi \mapsto \psi \gg\!\!\triangleright p \overset{\text{ind.}}{=} \phi \mapsto \psi \gg\!\!\triangleright \psi \mapsto p \overset{\text{G3}}{=} \psi \gg\!\!\triangleright \phi \mapsto \psi \mapsto p$$
$$\overset{\text{Prop. 2}}{=} \psi \gg\!\!\triangleright \psi \mapsto \phi \mapsto p$$

Case $\psi' \gg\!\!\triangleright p$.

$$\psi \gg\!\!\triangleright \psi' \gg\!\!\triangleright p \overset{\text{Prop. 3}}{=} \psi' \gg\!\!\triangleright \psi \gg\!\!\triangleright p \overset{\text{ind.}}{=} \psi' \gg\!\!\triangleright \psi \gg\!\!\triangleright \psi \mapsto p$$
$$\overset{\text{Prop. 3, G3}}{=} \psi \gg\!\!\triangleright \psi \mapsto \psi' \gg\!\!\triangleright p$$

Case $\{\!|C|\!\}\, p$. Assume $C' \cap var(\psi) = \varnothing$.

$$\psi \gg\!\!\triangleright \{\!|C|\!\}\, p \overset{(\simeq_\alpha)}{=} \psi \gg\!\!\triangleright \{\!|C'|\!\}\, \xi_{[C'/C]}p \overset{\text{I3}}{=} \{\!|C'|\!\}\, \psi \gg\!\!\triangleright \xi_{[C'/C]}p$$
$$\overset{\text{ind.}}{=} \{\!|C'|\!\}\, \psi \gg\!\!\triangleright \psi \mapsto \xi_{[C'/C]}p \overset{\text{I3,G4}}{=} \psi \gg\!\!\triangleright \psi \mapsto \{\!|C'|\!\}\, \xi_{[C'/C]}p \overset{(\simeq_\alpha)}{=} \psi \gg\!\!\triangleright \psi \mapsto \{\!|C|\!\}\, p$$

Case $p + q$.

$$\psi \gg\!\!\triangleright (p + q) \overset{\text{I4}}{=} \psi \gg\!\!\triangleright p + \psi \gg\!\!\triangleright q \overset{\text{ind.}}{=} \psi \gg\!\!\triangleright \psi \mapsto p + \psi \gg\!\!\triangleright \psi \mapsto q$$
$$\overset{\text{I4,G5}}{=} \psi \gg\!\!\triangleright \psi \mapsto (p + q)$$

Q.E.D.

Notice that $\mathbf{ff} \gg\!\!\triangleright \mathbf{0}$ is the neutral element with respect to summation in ♡, and not $\mathbf{0}$ as in ♧. In fact, this seems reasonable if we agree on the intuition that the neutral element should be the "least" expressive unit. Summation has the intuition of "adding" new behaviour and as a consequence the neutral element is the one that can be added to any behaviour without altering it. In fact $\mathbf{0}$ has certain behaviour in itself: it can idle infinitely long. Instead $\mathbf{ff} \gg\!\!\triangleright \mathbf{0}$ is the immediate time deadlocking process; it cannot do anything, including idling.

In the following we prove that the axiom system $ax(\heartsuit^b)-\mathbf{D}$, together with α-congruence, is sound and complete for symbolic bisimulation.

Theorem 6.3. *Let p and q be α-conflict-free ♡ terms. If $p = q$ is proved by means of equational reasoning using $ax(\heartsuit^b) - \mathbf{D}$ and α-conversion, then $p \sim_{\&} q$.*

Proof. The case of α-conversion is Theorem 5.16. For every axiom $p = q$ in $ax(\heartsuit^b) - \mathbf{D}$ assume $\neg\mathsf{cv}(p)$ and $\neg\mathsf{cv}(q)$ after appropriate α-conversion. For every axiom, with the exception of **CR2**, we define the relation

$$R = \{\langle p, q, SR\rangle \mid \neg\mathsf{cv}(p) \wedge \neg\mathsf{cv}(q) \wedge \langle \kappa(p), \kappa(q)\rangle \in SR$$
$$\wedge\, (\langle \{x\}, \{x\}\rangle \in SR - \{\langle \kappa(p), \kappa(q)\rangle\} \iff x \in FV(p) \cup FV(q))\}$$
$$\cup\, Id_{\&}$$

with $Id_{\&}$ being the identity relation for symbolic bisimulation defined as follows

$$\langle p, p, SR\rangle \in Id_{\&} \stackrel{\text{def}}{\Longleftrightarrow} \left\{ \begin{array}{lll} \neg \mathsf{cv}(p) & \wedge & \langle \kappa(p), \kappa(p)\rangle \in SR \\ \wedge & (\langle C, C'\rangle \in SR \Rightarrow C = C') \\ \wedge & \{C \mid \langle C, C'\rangle \in SR\} \supseteq \kappa(p) \cup FV(p) \end{array} \right.$$

In every case, R is a symbolic bisimulation.

Observe that **CR2** has a flavour of α-conversion. In fact, we could define a weaker notion of α-congruence, say $\cong_\alpha$, just like $\simeq_\alpha$ in Definition 5.9, but adding the following rule

$$\frac{\xi_{\{C^\star/C\}}(p) \cong_\alpha \xi_{\{C^\star/C'\}}(q) \quad C^\star \cap (\ (fv(p) - C) \cup (fv(q) - C')\ ) = \varnothing}{\{\!|C|\!\}\, p \cong_\alpha \{\!|C'|\!\}\, q}$$

in which we consider a (surjective) simultaneous substitution $\xi_{\{C^\star/C\}}$ instead of the bijective case $\xi_{[C^\star/C]}$ Notice that $\{\!|C \cup C' \cup \{x\}|\!\}\, p \cong_\alpha \{\!|C \cup \{x\}|\!\}\, \xi_{\{x/C'\}}(p)$. So, proving that $\cong_\alpha$ preserves $\sim_{\&}$ implies that **CR2** preserves $\sim_{\&}$. The proof that $\cong_\alpha$ preserves $\sim_{\&}$ follows tightly the proof of Theorem 5.16 (see Appendix B.2). *Q.E.D.*

Axioms **D1** and **D2** do not preserve $\sim_{\&}$. To make it clear, consider the following equality (remember that $\models y \geq 0 \Leftrightarrow \mathbf{tt}$):

$$\begin{array}{l} (x > 1) \mapsto a; (x < 1) \mapsto b; \mathbf{0} \stackrel{\text{CR1,L1}}{=} (x > 1) \mapsto a; \{\!|y|\!\}\, (y \geq 0 \wedge x < 1) \mapsto b; \mathbf{0} \\ \stackrel{\text{D1}}{=} (x > 1) \mapsto a; \{\!|y|\!\}\, (x - y > 1 \wedge y \geq 0 \wedge x < 1) \mapsto b; \mathbf{0} \\ \stackrel{\text{L1}}{=} (x > 1) \mapsto a; \{\!|y|\!\}\, \mathbf{ff} \mapsto b; \mathbf{0} \stackrel{\text{CR1,Stp}}{=} (x > 1) \mapsto a; \mathbf{0} \end{array}$$

It should be clear that $(x > 1) \mapsto a; (x < 1) \mapsto b; \mathbf{0} \not\sim_{\&} (x > 1) \mapsto a; \mathbf{0}$. Nevertheless, $ax(\heartsuit^b)$ is sound for timed bisimulation.

Theorem 6.4. *Let p and q be α-conflict-free $\heartsuit$ terms. If $p = q$ is proved by means of equational reasoning using $ax(\heartsuit^b)$ and α-conversion, then $p \sim_t q$.*

Proof. Since $\sim_t \subseteq \sim_{\&}$ and using the previous theorem, it is only necessary to check that **D1** and **D2** preserve $\sim_t$. In this case it is easier to use the direct semantics in terms of timed transition systems (see Definition 5.23)[1]. For **D1**, we define

$$\begin{array}{rcl} R_1 & \stackrel{\text{def}}{=} & \{\langle\langle \phi \mapsto a; (\{\!|y|\!\}\, p), v\rangle, \langle \phi \mapsto a; (\{\!|y|\!\}\, (x - y\Box d) \gg p), v\rangle\rangle \mid v \in \mathbf{V}\} \\ & \cup & \{\langle\langle \{\!|y|\!\}\, p, v\rangle, \langle \{\!|y|\!\}\, (x - y\Box d) \gg p, v\rangle\rangle \mid v \in \mathbf{V} \wedge \models v(\phi)\} \\ & \cup & Id \end{array}$$

[1] In (D'Argenio and Brinksma, 1996a,b) the complete set of axioms $ax(\heartsuit^b)$ has been proved to preserve timed bisimulation using the direct semantics.

For **D2**, we define

$$\begin{aligned} R_2 \stackrel{\text{def}}{=} & \{\langle\langle \phi \mapsto a; p, v\rangle, \langle \phi \mapsto a; ((x - y\Box d) \gg p), v\rangle\rangle \mid v \in \mathbf{V}\} \\ \cup & \{\langle\langle p, v\rangle, \langle (x - y\Box d) \gg p, v\rangle\rangle \mid v \in \mathbf{V} \wedge \models v(\phi)\} \\ \cup & \mathit{Id} \end{aligned}$$

Both relations can be proved to be timed bisimulations. As a consequence, the theorem follows. *Q.E.D.*

In fact this last theorem is also valid for the set $\heartsuit^{br}$ of arbitrary sequential and recursive ♡ terms. To prove it, all axioms should be checked using the direct semantics given in Section 5.6. The style of proving this is quite the same to the proofs above. The curious reader is referred to (D'Argenio and Brinksma, 1996a) where this proof can be found.

An interesting property that is derived from these axioms is that every term can be expressed in a normal form.

Definition 6.5. We define the set $B \subseteq \heartsuit$ of *basic terms* using an auxiliary set B'. We proceed inductively as follows:

- $\mathbf{0} \in B'$
- $p \in B$, $\phi \in \Phi$ and $a \in \mathcal{A} \quad \Longrightarrow \quad \phi \mapsto a; p \in B'$
- $p, q \in B' \quad \Longrightarrow \quad p + q \in B'$
- $p \in B'$, $\psi \in \overline{\Phi}$ and $x \in \mathcal{C} \quad \Longrightarrow \quad \{\!|x|\!\}\, \psi \gg p \in B$

B' is the set of all terms whose clock resettings and invariants are all within the scope of a prefix construction. Notice that a basic term has the general format (modulo **A1**, **A2**, **A3**, **A3′** and **A4**)

$$p = \{\!|x|\!\}\, \psi \gg \left(\sum_{i \in I} \phi_i \mapsto a_i; p_i \right)$$

where each p_i is a basic term and $\sum$ is as in Section 5.4.

We say that a basic term p is *non-redundant* if, for all $i \in I$, $\psi \wedge \phi_i$ is satisfiable, i.e. $\models \neg((\psi \wedge \phi_i) \Leftrightarrow \mathbf{ff})$, and moreover, each p_i is also non-redundant. □

In fact the existence of non-redundant basic terms plays an important role in the proof of completeness of $ax(\heartsuit^b) - \mathbf{D}$. The following theorem can be proven by structural induction using the axioms. Its complete proof is given in Appendix C.1.

Theorem 6.6. *For every term $p \in \heartsuit^b$ there is a non-redundant basic term q such that $p = q$ can be proven by means of the axiom system $ax(\heartsuit^b) - \mathbf{D}$ and α-conversion.*

We dedicate the rest of the section to prove completeness of $ax(\heartsuit^b) - \mathbf{D}$ with respect to $\sim_{\&}$. To do so, some auxiliary definitions and propositions will be necessary.

Definition 6.7. Let $p \equiv \{\!|x|\!\}\,\psi \gg\!\!> p'$ and $q \equiv \{\!|x|\!\}\,\psi \gg\!\!> q'$ be two basic terms. We say that p is a *summand* of q, notation $p \leq q$, if $q = \{\!|x|\!\}\,\psi \gg\!\!> (q' + p')$ can be proved in $ax(\heartsuit^b) - \mathbf{D}$. We extend $\leq$ to any term by defining $p^\star \leq q^\star$ whenever $p^\star = p \leq q = q^\star$ where $=$ is the equality induced by $ax(\heartsuit^b) - \mathbf{D}$ together with α-conversion. □

Proposition 6.8. *$\leq$ as defined above, is a partial order on $\heartsuit^b$.*

Proof. Since $=$ is an equivalence relation by definition, we only need to restrict to the defining case. We only show the case of transitivity which is the less straightforward. Suppose $p \leq q$ and $q \leq r$, p and q are as in Definition 6.7, and $r \equiv \{\!|x|\!\}\,\psi \gg\!\!> r'$. Then

$$
\begin{aligned}
\{\!|x|\!\}\,\psi \gg\!\!> r' &\overset{(\leq)}{=} \{\!|x|\!\}\,\psi \gg\!\!> (r' + q') \\
&\overset{\text{CR4,I4}}{=} (\{\!|x|\!\}\,\psi \gg\!\!> r') + (\{\!|x|\!\}\,\psi \gg\!\!> q') \\
&\overset{(\leq)}{=} (\{\!|x|\!\}\,\psi \gg\!\!> r') + (\{\!|x|\!\}\,\psi \gg\!\!> (q' + p')) \\
&\overset{\text{CR4,I4}}{=} (\{\!|x|\!\}\,\psi \gg\!\!> r') + (\{\!|x|\!\}\,\psi \gg\!\!> q') + (\{\!|x|\!\}\,\psi \gg\!\!> p') \\
&\overset{\text{CR4,I4,}(\leq)}{=} (\{\!|x|\!\}\,\psi \gg\!\!> r') + (\{\!|x|\!\}\,\psi \gg\!\!> p') \\
&\overset{\text{CR4,I4}}{=} \{\!|x|\!\}\,\psi \gg\!\!> (r' + p')
\end{aligned}
$$

Q.E.D.

Proposition 6.9. *Let $p \equiv \{\!|x|\!\}\,\psi \gg\!\!> p'$ be a basic term.*

1. *If $\phi \mapsto a; p''$ is a summand of p' then $p \xrightarrow{a,\phi}{}' p^\star$ for some $p^\star$ such that $p^\star = p''$.*
2. *Conversely, if $p \xrightarrow{a,\phi}{}' p^\star$, there is a summand $\phi \mapsto a; p''$ of p' such that $p^\star = p''$.*

Here, equality means that it can be derived from $ax(\heartsuit^b) - \mathbf{D}$ together with α-conversion.

Proof. In fact α-conversion suffices. It is immediate that, if $p \equiv \{\!|x|\!\}\,\psi \gg\!\!> \big(\sum_{i\in I} \phi_i \mapsto a_i; p_i\big)$, $p \xrightarrow{a,\phi}{}' p^\star$ if and only if $a \equiv a_j$, $\phi \equiv \phi_j$, $p^\star \simeq_\alpha p_j$, for some $j \in I$. *Q.E.D.*

We are now able to prove completeness.

Theorem 6.10. *The axiom system $ax(\heartsuit^b) - \mathbf{D}$ together with α-conversion is complete for the algebra $(\heartsuit^b, sig(\heartsuit^b), \sim_{\&})$.*

Proof. Let $p, q \in \heartsuit^b$ such that $p \sim_{\&} q$. Because of Theorem 6.6 and soundness, there are non-redundant basic terms $\hat{p}$ and $\hat{q}$ such that $\hat{p} \sim_{\&} p \sim_{\&} q \sim_{\&} \hat{q}$. Moreover, because of α-conversion, we can assume $\hat{p} \equiv \{\!|x|\!\}\,\psi \gg\!\!> p'$ and $\hat{q} \equiv \{\!|x|\!\}\,\psi' \gg\!\!> q'$. Since $\hat{p} \sim_{\&} \hat{q}$, $\models \xi^1_{SR}(\psi) \Leftrightarrow \xi^2_{SR}(\psi')$ for some adequate SR. But according to conditions (a)–(c) in Definition 4.16, we can guarantee that $\models \psi \Leftrightarrow \psi'$. So, by axiom **L2** and w.l.o.g., we can assume $\psi \equiv \psi'$.

We proceed by induction on the maximum number of nested prefixes in $\hat{p}$ and $\hat{q}$. If it is 0 then $\hat{p} = \{\!|x|\!\}\, \psi \gg \mathbf{0} = \hat{q}$. Otherwise, $\hat{p} = \{\!|x|\!\}\, \psi \gg p'$ and $\hat{q} = \{\!|x|\!\}\, \psi \gg q'$ with

$$p' = \left(\sum_{i\in I} \phi_i \mapsto a_i; p_i\right) \quad \text{and} \quad q' = \left(\sum_{j\in J} \phi_j \mapsto a_j; q_j\right).$$

Take a summand $\phi_k \mapsto a_k; p_k$ of p'. Then $\hat{p} \xrightarrow{a_k,\phi_k}{}' p^\star$ with $k \in I$ and $p_k = p^\star$ with $\psi \wedge \phi_k$ satisfiable. Since $\langle \hat{p}, \hat{q}, SR\rangle \in R$ for some symbolic bisimulation R with SR satisfying conditions (a)–(c) in Definition 4.16, only the transfer property 5b applies. Thus, there are $\{q_h^\star \mid h \in H\}$, such that

1. $\hat{q} \xrightarrow{a_k,\phi_h}{}' q_h^\star$, for all $h \in H$;
2. $\models \xi^1_{SR}(\psi \wedge \phi_k) \Rightarrow \bigvee_{h\in H} \xi^2_{SR}(\psi \wedge \phi_h)$; and
3. for all $h \in H$, there exists SR_h such that $\langle p^\star, q_h^\star, SR_h\rangle \in R$ and

$$\{\langle C_1 \cap FV(p^\star), C_2 \cap FV(q_h^\star)\rangle \mid \langle C_1, C_2\rangle \in SR_h\} - \{\langle \varnothing, \varnothing\rangle\} = \{\langle C_1 \cap FV(p^\star), C_2 \cap FV(q_h^\star)\rangle \mid \langle C_1, C_2\rangle \in SR\} - \{\langle \varnothing, \varnothing\rangle\}.$$

Notice that for all $h \in H$, SR_h still satisfies conditions (a)–(c) in Definition 4.16. As a consequence $p^\star \sim_{\&} q_h^\star$ for all $h \in H$. Moreover, because of the shape of SR, item 2 guarantees that

$$\models (\psi \wedge \phi_k) \Rightarrow \bigvee_{h\in H} (\psi \wedge \phi_h).$$

By Proposition 6.9, we can assume $H \subseteq J$ such that, for all $h \in H$, $a_k \equiv a_h$, $q_h^\star = q_h$, and $\phi_h \mapsto a_h; q_h$ is a summand of q'. Since $p_k \sim_{\&} p^\star \sim_{\&} q_h^\star \sim_{\&} q_h$, by induction $p_k = q_h$ for all $h \in H$. Now, we can calculate,

$$\begin{aligned}
\{\!|x|\!\}\, \psi \gg \phi_k \mapsto a_k; p_k &= \{\!|x|\!\}\, \psi \gg (\psi \wedge \phi_k) \mapsto a_k; p_k \\
&\leq \{\!|x|\!\}\, \psi \gg \Big((\psi \wedge \phi_k) \mapsto a_k; p_k + \big(\bigvee_{h\in H} (\psi \wedge \phi_h)\big) \mapsto a_k; p_k\Big) \\
&= \{\!|x|\!\}\, \psi \gg \big((\psi \wedge \phi_k) \vee \bigvee_{h\in H} (\psi \wedge \phi_h)\big) \mapsto a_k; p_k \\
&= \{\!|x|\!\}\, \psi \gg \big(\psi \wedge \bigvee_{h\in H} \phi_h\big) \mapsto a_k; p_k \\
&= \{\!|x|\!\}\, \psi \gg \sum_{h\in H} \phi_h \mapsto a_k; p_k \\
&= \{\!|x|\!\}\, \psi \gg \sum_{h\in H} \phi_h \mapsto a_h; q_h \\
&\leq \hat{q}
\end{aligned}$$

Proceeding in a similar way for every summand of p', we conclude that $\hat{p} \leq \hat{q}$. By symmetry $\hat{q} \leq \hat{p}$ and, as a consequence $p = q$. *Q.E.D.*

6.2 Axioms for the Static Operators

The basic operations together with the basic axioms form the core of the algebraic theory of $\heartsuit$. In this section we extend the equational treatment to the rest of the operators and show how they can be decomposed and eliminated in favour of the basic operators. We also define the complete equational theory and show that it is sound for symbolic and timed bisimulation depending whether axioms **D1** and **D2** are considered. The theory is proved to be complete for the set $\heartsuit^c$ of closed $\heartsuit$ terms (i.e., the set of those terms not containing process variables) under symbolic bisimulation.

The new axioms are given in Table 6.2. Axioms **R1**–**R6** define the renaming operation in terms of the basic operations. Notice that if we consider the axioms as rewrite rules from left to right, the renaming operation is "pushed" inside the term while appropriately renaming actions according to f.

The other axioms are related to the parallel composition. **Mrg** decomposes the parallel composition in terms of the left and communication merge. Thus, if $p \,||_A\, q$, then either the p can try to do the first activity ($p \,\between_A\, q$), or q does ($q \,\between_A\, p$), or both synchronise in a communication action ($p \,|_A\, q$). Axioms **LM1**–**LM8** define the left merge depending on the structure not only of the left operand —as is traditionally the case, see Table 2.2 in Chapter 2—, but also of the right one. Notice that in order to move a prefix in the left operand out of the scope of the left merge, it is necessary, on the one hand, that the right operand does not reset any clock, and on the other, that the invariant is appropriately moved out of the parallel composition. This is controlled by the predicate TR. This predicate is related to the auxiliary set B' used to define the set of basic terms (see Definition 6.5). Moreover, it can easily be observed from axioms **TR1**–**TR4** that if a process p satisfies $\mathsf{TR}(p)$ then $\kappa(p) = \varnothing$. Thus, TR is also used to encode the appropriate resetting of clocks in a parallel composition, responding thus to the same functionality that $\overline{\mathsf{ck}}$ has in the operational semantics. (**TR** stands for "trivially rooted process".) Finally, axioms **CM0**–**CM8** define the communication merge depending on the form of both operands.

Definition 6.11. The equational theory for $\heartsuit$ is defined as follows. The signature $sig(\heartsuit)$ contains the constant $\mathbf{0}$, the unary operations $a;\_$, $\phi \mapsto \_$, $\psi \gg \_$, $\{\!|C|\!\}\_$, and $\_[f]$, and the binary operations $+$, $||_A$, $\between_A$, and $|_A$. The axiom system $ax(\heartsuit)$ is given by axioms in Table 6.1 and Table 6.2. We identify with $ax(\heartsuit) - \mathbf{D}$ the subset of axioms which does not contain axioms **D1** and **D2**. □

The extended equational theory is sound and complete for symbolic bisimulation if the axioms **D1** and **D2** are excluded; if they are not, the complete theory is sound for timed bisimulation. We prove soundness in the next theorem.

Theorem 6.12. *Let p and q be α-conflict-free $\heartsuit$ terms. If $p = q$ is proved by means of equational reasoning using $ax(\heartsuit) - \mathbf{D}$ and α-conversion, then $p \sim_{\&} q$. If it is proved using $ax(\heartsuit)$ and α-conversion, then $p \sim_t q$.*

Proof. It is only necessary to check that axioms in Table 6.2 preserve $\sim_{\&}$. We show first

R1	$\mathbf{0}[f] = \mathbf{0}$	**R4**	$(\psi \gg p)[f] = \psi \gg (p[f])$
R2	$(a;p)[f] = f(a);(p[f])$	**R5**	$(\{\!|C|\!\}\, p)[f] = \{\!|C|\!\}\, (p[f])$
R3	$(\phi \mapsto p)[f] = \phi \mapsto (p[f])$	**R6**	$(p+q)[f] = p[f] + q[f]$

Mrg	$p \,\|\|_A\, q = p \between_A q + q \between_A p + p \mid_A q$	
LM1	$\mathbf{0} \between_A (\psi \gg q) = \psi \gg \mathbf{0}$	if $\mathsf{TR}(q)$
LM2	$a;p \between_A (\psi \gg q) = \psi \gg \mathbf{0}$	if $a \in A \wedge \mathsf{TR}(q)$
LM3	$a;p \between_A (\psi \gg q) = \psi \gg a;(p \,\|\|_A\, (\psi \gg q))$	if $a \notin A \wedge \mathsf{TR}(q)$
LM4	$(\phi \mapsto p) \between_A q = \phi \mapsto (p \between_A q)$	
LM5	$(\psi \gg p) \between_A q = \psi \gg (p \between_A q)$	
LM6	$(\{\!|C|\!\}\, p) \between_A q = \{\!|C|\!\}\, (p \between_A q)$	if $C \cap fv(q) = \varnothing$
LM7	$(p+q) \between_A r = p \between_A r + q \between_A r$	
LM8	$p \between_A \{\!|C|\!\}\, q = \{\!|C|\!\}\, (p \between_A q)$	if $C \cap fv(p) = \varnothing$
CM0	$p \mid_A q = q \mid_A p$	
CM1	$\mathbf{0} \mid_A \mathbf{0} = \mathbf{0}$	
CM2	$\mathbf{0} \mid_A a;p = \mathbf{0}$	
CM3	$a;p \mid_A a;q = a;(p \,\|\|_A\, q)$	if $a \in A$
CM4	$a;p \mid_A b;q = \mathbf{0}$	if $a \neq b \vee a \notin A$
CM5	$(\phi \mapsto p) \mid_A q = \phi \mapsto (p \mid_A q)$	
CM6	$(\psi \gg p) \mid_A q = \psi \gg (p \mid_A q)$	
CM7	$(\{\!|C|\!\}\, p) \mid_A q = \{\!|C|\!\}\, (p \mid_A q)$	if $C \cap fv(q) = \varnothing$
CM8	$(p+q) \mid_A r = p \mid_A r + q \mid_A r$	

TR1 $\mathsf{TR}(\mathbf{0})$ **TR2** $\mathsf{TR}(a;p)$ **TR3** $\dfrac{\mathsf{TR}(p)}{\mathsf{TR}(\phi \mapsto p)}$ **TR4** $\dfrac{\mathsf{TR}(p) \quad \mathsf{TR}(q)}{\mathsf{TR}(p+q)}$

Table 6.2: Axioms for the static operators

that they are even stronger and preserve $\sim_s$. In fact, it is straightforward to check that

$$R_s = \{\langle p, q\rangle \mid p = q \text{ is an axiom in Table 6.2 with } p \text{ and } q \text{ being conflict-free}\}$$

is a structural bisimulation up to $\sim_s$, knowing that $p \,||_A\, q \sim_s q \,||_A\, p$ which is easy to prove.

Notice that this is not enough to prove that the new axioms preserve $\sim_{\&}$ over the larger class of α-conflict-free terms. In any case, the same relation R in the proof of Theorem 6.3 applies to all the axioms except for **Mrg** and **CM1** for which we should consider the relation $R' = R \cup R^c$ with R^c defined as follows

$$\langle p \,||_A\, q, q \,||_A\, p, SR\rangle \in R^c \stackrel{\text{def}}{\iff} \left\{ \begin{array}{l} \neg\mathsf{cv}(p \,||_A\, q) \;\wedge\; \langle \kappa(p \,||_A\, q), \kappa(q \,||_A\, p)\rangle \in SR \\ \wedge\; (\langle C, C'\rangle \in SR \Rightarrow C = C') \\ \wedge\; \{C \mid \langle C, C'\rangle \in SR\} \supseteq \kappa(p \,||_A\, q) \cup FV(p \,||_A\, q) \end{array} \right.$$

Q.E.D.

In the following we show completeness of $ax(\heartsuit) - \mathbf{D}$. To do so we first prove that the static operators can be eliminated in favour of the basic operations, that is, for every term in $\heartsuit^c$ there is a term in $\heartsuit^b$ that can be proved equal using the axioms.

Theorem 6.13. *For every term p in $\heartsuit^c$, there is a $q \in \heartsuit^b$ (not containing $||_A$, $\between_A$, $|_A$, and $\_[f]$) such that $p = q$ can be proved using $ax(\heartsuit) - \mathbf{D}$ and α-conversion.*

Proof sketch. Consider axioms in Table 6.2 from left to right as rewrite rules modulo axioms in Table 6.1 and **CM0**. It is simple to prove that the normal form is a term $q' \in \heartsuit^b$. The theorem follows by Theorem 6.6. *Q.E.D.*

Considering Theorem 6.10, completeness follows as an immediate corollary of Theorem 6.13.

Theorem 6.14. *The axiom system $ax(\heartsuit) - \mathbf{D}$ together with α-conversion is complete for the algebra $(\heartsuit^c, sig(\heartsuit), \sim_{\&})$.*

Not surprisingly, from the axioms we can prove an expansion theorem.

Theorem 6.15. *Let $p, q \in \heartsuit$ such that $p = \{\!|C|\!\}\,\psi \gg p'$ and $q = \{\!|C'|\!\}\,\psi' \gg q'$ with $p' = \sum_i \phi_i \mapsto a_i; p_i$ and $q' = \sum_j \phi'_j \mapsto b_j; q_j$. Suppose $p \,||_A\, q$ is α-conflict-free and in particular local conflict-free. From the axiom system $ax(\heartsuit) - \mathbf{D}$ we can derive*

$$\begin{array}{rcl} p \,||_A\, q & = & \{\!|C \cup C'|\!\}\,(\psi \wedge \psi') \gg \Big(\displaystyle\sum_{a_i \notin A} \phi_i \mapsto a_i; (p_i \,||_A\, q') \\ & & \quad + \displaystyle\sum_{b_j \notin A} \phi'_j \mapsto b_j; (p' \,||_A\, q_j) \\ & & \quad + \displaystyle\sum_{a_i = b_j \in A} (\phi_i \wedge \phi'_j) \mapsto a_i; (p_i \,||_A\, q_j) \Big) \end{array}$$

Moreover, if $p[f]$ is α-conflict-free and in particular local conflict-free, the following equality can also be derived,

$$p[f] \;=\; \{\!|C|\!\}\,\psi \gg \big(\textstyle\sum_i \phi_i \mapsto f(a_i); (p_i[f]) \big)$$

To illustrate calculation using the equational theory for ♡, we show that the switch modelled using ♡ in Example 5.2 is equal to the equation system that represents the timed automaton of the switch in Example 5.19.

Example 6.16. Recall that the system is described by

$$\mathit{System} \;=\; \mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOff}$$

where

$$\begin{aligned}
\mathit{Arrival} &= \mathit{on}; \mathit{Arrival} \\
\mathit{SwitchOff} &= \mathit{on}; \mathit{SwitchOn} \\
\mathit{SwitchOn} &= \{\!|x|\!\}\,(x \leq 2) \trianglerighteq (\; \mathit{on}; \mathit{SwitchOn} + (x = 2) \mapsto \mathit{off}; \mathit{SwitchOff}\;)
\end{aligned}$$

We use the axioms, the expansion law, and process variable folding and unfolding to calculate the following

$$\begin{aligned}
\mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOff} &= \mathit{on}; \mathit{Arrival} \,||_{\mathit{on}}\, \mathit{on}; \mathit{SwitchOn} \\
&= \{\!|\varnothing|\!\}\,\mathbf{tt} \trianglerighteq \mathbf{tt} \mapsto \mathit{on}; (\; \mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOn}\;)
\end{aligned}$$

$$\begin{aligned}
&\mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOn} \\
&\quad= \mathit{on}; \mathit{Arrival} \,||_{\mathit{on}}\, \{\!|x|\!\}\,(x \leq 2) \trianglerighteq (\; \mathit{on}; \mathit{SwitchOn} \\
&\qquad\qquad + (x = 2) \mapsto \mathit{off}; \mathit{SwitchOff}\;) \\
&\quad= \{\!|x|\!\}\,(x \leq 2) \trianglerighteq \big(\; \mathit{on}; (\mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOn}) \\
&\qquad\qquad + (x = 2) \mapsto \mathit{off}; (\mathit{on}; \mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOff})\;\big) \\
&\quad= \{\!|x|\!\}\,(x \leq 2) \trianglerighteq \big(\; \mathbf{tt} \mapsto \mathit{on}; (\mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOn}) \\
&\qquad\qquad + (x = 2) \mapsto \mathit{off}; (\mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOff})\;\big)
\end{aligned}$$

We recall that in Example 5.19 the obtained equation system was as follows.

$$\begin{aligned}
X_{s_0} &= \{\!|\varnothing|\!\}\,\mathbf{tt} \trianglerighteq \mathbf{tt} \mapsto \mathit{on}; X_{s_1} \\
X_{s_1} &= \{\!|x|\!\}\,(x \leq 2) \trianglerighteq (\; \mathbf{tt} \mapsto \mathit{on}; X_{s_1} + (x = 2) \mapsto \mathit{off}; X_{s_0})
\end{aligned}$$

We are close to the solution, but we are not yet in conditions to actually conclude that $\mathit{System} = X_{s_0}$. *(To be continued below)*

In fact, both pairs $\langle \mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOn}, \mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOff}\rangle$ and $\langle X_{s_0}, X_{s_1}\rangle$ solve the same system of equations, but we have no reason to say that both solutions are the same. For instance, in arithmetics, both 2 and $-\frac{17}{7}$ are solutions of the equation $x = 14x^2+7x-68$ and yet they are different. In process algebras there are ways to characterise those models in which recursive specifications, i.e. systems of recursive equations, have solutions, and moreover, if they are unique.

Definition 6.17. We define the following principles.

- The *Recursive Specification Principle (RSP)* is the assumption that every guarded recursive specification has at most one solution.
- The *Recursive Definition Principle (RDP)* is the assumption that every recursive specification has a solution.
- The *Restricted Recursive Definition Principle (RDP$^-$)* is the assumption that every guarded recursive specification has a solution. □

These principles were originally introduced in the context of ACP by Bergstra and Klop (1986). They also appear in (Baeten and Weijland, 1990; Baeten and Verhoef, 1995). Notice that a model that satisfies RDP also satisfies RDP$^-$. Moreover, if a model satisfies both RSP and RDP$^-$, then it is guaranteed that guarded recursive specifications has a unique solution.

Because of Proposition 5.15, every guarded recursive specification in $\heartsuit$ has a solution. In the case of the model $\heartsuit^{br}$ of sequential recursive terms every recursive specification has a solution, since they have a direct semantics in terms of timed transition systems and the condition of α-conflict-freedom can be dropped. This proves that $\heartsuit$ satisfies RDP$^-$ and, if we restrict to only sequential terms, it satisfies RDP.

The proof that $\heartsuit$ satisfies RSP is more involved and we will not give it here. We just mention that it is just a straightforward adaptation of the proof given for ACP (Baeten and Weijland, 1990). So, we formally state the following theorem.

Theorem 6.18. *The algebra $(\heartsuit^{br}, sig(\heartsuit^b), \sim_t)$ satisfies RSP and RDP. If $\heartsuit^{\alpha+}$ is the set of all α-conflict-free $\heartsuit$ terms, then the algebras $(\heartsuit^{\alpha+}, sig(\heartsuit^c), \sim_{\&})$ and $(\heartsuit^{\alpha+}, sig(\heartsuit^c), \sim_t)$ satisfy RSP and RDP$^-$.*

Now we are in conditions to conclude our example.

Example 6.16. *(Cont.)* Because RSP and RDP$^-$ are satisfied in $\heartsuit$, guarded recursive specifications have unique solutions. Therefore,

$$X_{s_0} = Arrival \,||_{on}\, SwitchOn = System \quad \text{and} \quad X_{s_1} = Arrival \,||_{on}\, SwitchOff$$

□

6.3 Conservative Extension

As we have mentioned before, $\heartsuit$ extends the syntax of $\clubsuit$. As a consequence $\clubsuit$ is a subset of $\heartsuit$. In the following we sketch the proof that both model and equational theory of $\heartsuit$ are conservative extensions of the model and equational theory of $\clubsuit$, respectively.

It is not difficult to see that for all $p \in \clubsuit^c$, $\kappa(p) = \varnothing$, $\iota(p) = \mathbf{tt}$ and $p \xrightarrow{a,\mathbf{tt}} p' \iff p \xrightarrow{a} p'$, just as we observed in Remark 4.4. As a consequence, in the context of $\clubsuit$, bisimulation coincides with structural, symbolic and timed bisimulation as we mentioned

in Remark 4.21. From this, we can conclude that $(\heartsuit^c, sig(\heartsuit^c), \sim_{\&})$ and $(\heartsuit^c, sig(\heartsuit^c), \sim_t)$ are model conservative extensions of $(\clubsuit^c, sig(\clubsuit^c), \sim)$.

In addition, notice that every axiom in $ax(\clubsuit)$ can be proved from the set of axioms $ax(\heartsuit) - \mathbf{D}$ except axiom **A4** (of course, under the condition that every process is a ♣ term). $p + \mathbf{0} = p$ does not hold in general in ♡ as we already discussed. Let $ax(\clubsuit)^-$ be the same as $ax(\clubsuit)$ except that axiom $p + \mathbf{0} = p$ is replaced by $a; p + \mathbf{0} = a; p$. Now, every axiom in $ax(\clubsuit)^-$ can be proved in $ax(\heartsuit) - \mathbf{D}$. $ax(\clubsuit)^-$ can still be proved sound and complete for $(\clubsuit^c, sig(\clubsuit^c), \sim)$.

Since $ax(\heartsuit) - \mathbf{D}$ is sound for $(\heartsuit^c, sig(\heartsuit^c), \sim_{\&})$ and $ax(\heartsuit)$ is sound for $(\heartsuit^c, sig(\heartsuit^c), \sim_t)$, the above discussed put us in the conditions of (D'Argenio and Verhoef, 1997, see also Section 2.3, page 25) from which we conclude

Theorem 6.19. *The equational theories of* ♡ *either with or without axioms* **D1** *and* **D2** *given in Definition 6.11 are conservative extensions of the equational theory* $ax(\clubsuit)^-$ *defined above.*

Chapter 7

Further Examples using ♡

In this chapter we give several examples using ♡. The first two sections are concerned with the specification of real-time systems. In the first section, we show how to define some basic operations that allow to model typical processes such as timeouts or deadlines. In the second section, we specify and analyse a railroad crossing (Leveson and Stolzy, 1987; Alur and Dill, 1994).

The third section discusses more theoretic aspects rather than the study of real-time systems. We show that regular processes —i.e., processes containing process variables defined in regular recursive specifications— are as expressive as *finite* timed automata. This connection is evident between ♧ and labelled transition systems since each ♧ process represents a state in the transition system. Such a direct correspondence does not exist in ♡. Only (α-) conflict free processes are locations for the timed automaton (see Definitions 5.6 and 5.14). So, the problem is focussed on processes that have conflict of variables. For instance, to define the semantics of process X, where

$$X = (x < 3) \rhd\!\!\rhd \{\!|x|\!\} (x < 2) \rhd\!\!\rhd a; X,$$

we need to resort to semantics up to α-conversion which would induce a (probably) infinite timed automata. Yet, by appropriate manipulations, we can obtain a finite timed automaton that shows an equivalent (symbolically bisimilar) behaviour. (See Figure 7.1.) We show that the solution is general, that is, every process with variables defined in a regular

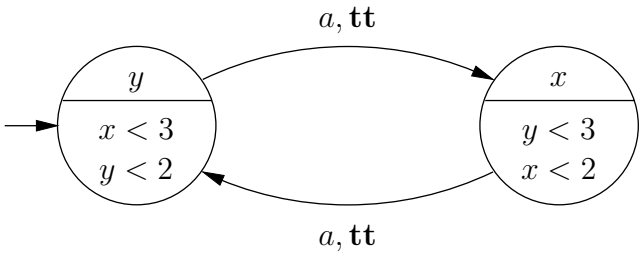


Figure 7.1: A finite timed automaton for $X = (x < 3) \rhd\!\!\rhd \{\!|x|\!\} (x < 2) \rhd\!\!\rhd a; X$

recursive specification is equivalent to a conflict-free process whose direct semantics is a finite timed automaton. To elaborate this result, we only resort to algebraic and syntactic manipulation.

7.1 Derived Operations

In this first section we define some operations that facilitate the specifications of real-time systems.

Let us consider a particular action $\tau \in \mathcal{A}$. We will require that for any possible parallel composition $p \,||_A\, q$, $p \,\lfloor\!\lfloor_A\, q$, or $p \,|_A\, q$, $\tau \notin A$, that is, τ is an action that cannot synchronise. In fact, τ plays the role of an *internal* action that cannot be observed by the environment and hence it cannot be subject to synchronisation. We will not consider, however, any special semantics for τ such as weak or branching bisimulation (Milner, 1989; Glabbeek and Weijland, 1996).

An important notion in the specification of systems is that of *abstraction*. Given a component one would like to abstract from or *hide* those activities that are not relevant for the environment and only concern to the internal behaviour of such a component. Thus, if such a component is represented by a process p, and the set of actions A is intended to be relevant only for p, the operation that abstracts in p all activities in A has to rename actions in A into the internal action τ. So, given a ♡ process p and a set of actions $A \subseteq \mathcal{A}$, the operation of hiding is defined by

$$\mathsf{hide}\, A \,\mathsf{in}\, p \stackrel{\text{def}}{=} p[f_A] \qquad \text{where} \quad f_A(a) \stackrel{\text{def}}{=} \begin{cases} \tau & \text{if } a \in A \\ a & \text{otherwise} \end{cases}$$

Thus $\mathsf{hide}\, A \,\mathsf{in}\, p$ is a process that behaves like p except that all those actions that belong to the hiding set A are renamed into τ.

In our setting, operations that involve timing aspects are more interesting. For instance, an important function is the *delaying* of the execution of a given process. The operation $\mathsf{delay}_d(p)$ idles for at least d units of time and then starts the execution of p. If x is a fresh clock (or at least not appearing free in p), operation delay_d can be defined as follows:

$$\mathsf{delay}_d(p) \stackrel{\text{def}}{=} \{\!|x|\!\}\, (x > d) \mapsto \tau; p$$

The exact moment at which p is able to start is unknown. We only know that it would start after delaying for at least d time units. Thus, we would probably be interested in an operation that delays an exact amount of time. We can define this operation as follows:

$$\mathsf{delay}^{=}_d(p) \stackrel{\text{def}}{=} \{\!|x|\!\}\, (x \leq d) \gg (x \geq d) \mapsto \tau; p$$

where x is fresh.

A more compact definition of this last functionality is given by the operation wait defined as follows:

$$\mathsf{wait}_d(p) \stackrel{\text{def}}{=} \{\!|x|\!\}\, (x \geq d) \mapsto p$$

However, wait is not quite equivalent to $\mathsf{delay}^{=}$ due to the absence of the τ prefixing. In this last case one must realise that any possible clock resetting of p (i.e., the reset of clocks in $\kappa(p)$) will be carried out before actually waiting. Therefore, it is convenient to use wait whenever $\kappa(p) = \varnothing$.

Another interesting operation is to force a process to start its execution *before* a certain time limit. Thus, $\mathsf{before}_d(p)$ forces p to start its execution before d time units have passed. If x is fresh, this operation can be defined by

$$\mathsf{before}_d(p) \stackrel{\text{def}}{=} \{\!|x|\!\}\,(x < d) \gg p$$

Notice that the definition of before does not have the internal action τ as a prefix of p as is the case in the delay operation. Putting an internal action in the definition of the before operation would not actually force the start of the execution of p but rather delay it.

We can generalise the before operation to an operation that forces to start the execution of process p in a given bounded interval. Let us call this operation between. The process $\mathsf{between}_{(d,d')}(p)$ starts the execution of p sometime after waiting d time units and before d' time units have passed. We define it as follows

$$\mathsf{between}_{(d,d')}(p) \stackrel{\text{def}}{=} \{\!|x|\!\}\,(x < d') \gg (x > d) \mapsto p$$

We can easily generalise this operation to closed intervals or combinations of closed and open in the obvious way. Observe that, in particular, $\mathsf{between}_{[d,d')}(p) = \mathsf{before}_{d'}(\mathsf{wait}_d(p))$.

A particular case of between is the operation that forces the execution of p at an exact time d. We call this operation urgent_d since it makes the execution of a given process to become urgent at exactly time d:

$$\mathsf{urgent}_d(p) \stackrel{\text{def}}{=} \mathsf{between}_{[d,d]}(p) = \{\!|x|\!\}\,(x \leq d) \gg (x \geq d) \mapsto p$$

By using the before and urgent operations we can define a *timeout*. Process $p\,\mathsf{to}_d\,q$ waits for the execution of p to start within d units of time; if p does not start quickly enough, then q immediately starts its execution at time d. We define to_d as follows

$$p\,\mathsf{to}_d\,q \stackrel{\text{def}}{=} \mathsf{before}_d(p) + \mathsf{urgent}_d(q)$$

In many cases, it is interesting to describe that a certain complex activity can be carried out within a given *deadline*. Consider a distinguished action *done* that indicates the end of the activity that has to meet the deadline. Thus, $\mathsf{deadline}_d(p)$ is a process that behaves like p except that every activity performed before p executes action *done* occurs within d time units. Since action *done* is relevant only within the scope of the deadline, it must be hidden from the environment. We define this operation as follows.

$$\mathsf{deadline}_d(p) \stackrel{\text{def}}{=} \mathsf{hide}\ \mathit{done}\ \mathsf{in}\ (p \,||_{\mathit{done}}\, \mathsf{before}_d(\mathit{done}; \mathbf{0}))$$

Notice that if p does not perform action *done* within d time units, $\mathsf{deadline}_d(p)$ reaches a time deadlock. Otherwise, it continues with the normal execution of p. At this point it is expected that p does not executes *done* for a second time.

We will use the operations introduced above in the following examples.

7.2 A Railroad Crossing

As an example of specification and analysis using ♡, we consider an automatic controller that opens and closes a gate at a railroad crossing. This example has been used in many papers as a simple benchmarking exercise (see, e.g., Leveson and Stolzy, 1987; Alur and Dill, 1994; Pettersson, 1999). We give a first approach to the specification of the system. This specification has traditionally been adopted to prove the safety property that there is no train in the crossing while the gate is open. We show that such approach is not completely correct in the sense that the closing of the gate is abnormally influenced by external factors. In the second part of this section, we give an alternative model which does not suffer from this drawback.

A first approach Three components constitute the system: a *Train*, a *Gate*, and a *Controller*. The *Train* communicates with the *Controller* to inform first that it approaches the crossing and then that it exits it. It does so by means of actions *appr* and *exit*, respectively. The *Controller* communicates with the *Gate* to command its closing and opening by using actions *lower* and *raise*, respectively. The *Train* and the *Gate* do not communicate with each other. The architecture of the system is then modelled by

$$RRCrossing \;=\; (Train \,||_{\varnothing}\, Gate) \,||_{A}\, Controller$$

where $A = \{appr, exit, lower, raise\}$.

Each component can be described as follows. The *Train* indicates to the *Controller* that it *appr*oaches at least 2 minutes before it enters the crossing, which is represented by action *in*. After leaving the crossing (action *out*), the *Train* informs the *Controller* that it *exit*ed. Furthermore, we know that the whole process, from signaling the approach until exiting, takes place within 5 time units. Therefore, the *Train* can be described as follows

$$Train \;=\; appr; \mathsf{deadline}_5(\mathsf{wait}_2(in; out; exit; done; Train))$$

The *Controller* waits until a *Train* signals its *appr*oach. After exactly 1 time unit, it indicates to *lower* the *Gate*. When the *Train exit*s the crossing, the *Controller* is informed so, and within 1 time unit it orders to *raise* the *Gate*.

$$Controller \;=\; appr; \mathsf{urgent}_1(lower; exit; \mathsf{before}_1(raise; Controller))$$

The *Gate* is a simple device that receives the indication to close the access from the road by means of action *lower*. After receiving this order, the *Gate* takes at most 1 time unit to close *down* the barrier. Once closed, the *Gate* waits the order of raising (action *raise*) and then it opens by lifting *up* the barrier. The response time to open the gate is between 1 and 2 time units. We describe the *Gate* as follows.

$$Gate \;=\; lower; \mathsf{before}_1(down; raise; \mathsf{between}_{(1,2)}(up; Gate))$$

By using the definitions given in the previous section we can rewrite the recursive equations given above as follows.

$$\begin{aligned}
\mathit{Train} &= \mathit{appr}; (\ (\ \{\!|x|\!\}\,(x \geq 2) \mapsto \mathit{in}; \mathit{out}; \mathit{exit}; \mathit{done}; \mathit{Train}) \\
&\qquad\qquad ||_{\mathit{done}}\ (\{\!|x'|\!\}\,(x' < 5) \gg \mathit{done}; \mathbf{0}) \\
&\qquad\qquad)[\mathit{done}/\tau] \\
\mathit{Controller} &= \mathit{appr}; \{\!|y|\!\}\,(y \leq 1) \gg (y \geq 1) \mapsto \mathit{lower}; \mathit{exit}; \\
&\qquad\qquad \{\!|y|\!\}\,(y < 1) \gg \mathit{raise}; \mathit{Controller} \\
\mathit{Gate} &= \mathit{lower}; \{\!|z|\!\}\,(z < 1) \gg \mathit{down}; \mathit{raise}; \\
&\qquad\qquad \{\!|z|\!\}\,(z < 2) \gg (z > 1) \mapsto \mathit{up}; \mathit{Gate}
\end{aligned}$$

We pay particular attention to the definition of *Train*. Using the axioms as well as the expansion law, it can be shown that

$$\begin{aligned}
\mathit{Train} = \ & \mathit{appr}; \{\!|x|\!\}\,(x < 5) \gg (x \geq 2) \mapsto \mathit{in}; \\
& (x < 5) \gg \mathit{out}; \\
& (x < 5) \gg \mathit{exit}; \\
& (x < 5) \gg \tau; \\
& (\ (\mathit{Train}\ ||_{\mathit{done}}\ \mathbf{0})[\mathit{done}/\tau]\ )
\end{aligned}$$

By RSP and RDP$^-$, we have that $\mathit{Train} = \mathit{Train}'$, where

$$\begin{aligned}
\mathit{Train}' = \ & \mathit{appr}; \{\!|x|\!\}\,(x < 5) \gg (x \geq 2) \mapsto \mathit{in}; \\
& (x < 5) \gg \mathit{out}; \\
& (x < 5) \gg \mathit{exit}; \\
& (x < 5) \gg \tau; \\
& \mathit{Train}'
\end{aligned}$$

As a consequence

$$\mathit{RRCrossing} = (\mathit{Train}' \,||_{\varnothing}\, \mathit{Gate}) \,||_{A}\, \mathit{Controller}$$

Consider the following shorthand notation:

$$\begin{aligned}
T_0 &\equiv \mathit{Train}' = \mathit{appr}; \{\!|x|\!\}\,T_1 & \qquad T_3 &\equiv (x < 5) \gg \mathit{exit}; T_4 \\
T_1 &\equiv (x < 5) \gg (x \geq 2) \mapsto \mathit{in}; T_2 & \qquad T_4 &\equiv (x < 5) \gg \tau; T_0 \\
T_2 &\equiv (x < 5) \gg \mathit{out}; T_3 & &
\end{aligned}$$

$$\begin{aligned}
C_0 &\equiv \mathit{Controller} = \mathit{appr}; \{\!|y|\!\}\,C_1 & \qquad G_0 &\equiv \mathit{Gate} = \mathit{lower}; \{\!|z|\!\}\,G_1 \\
C_1 &\equiv (y \leq 1) \gg (y \geq 1) \mapsto \mathit{lower}; C_2 & \qquad G_1 &\equiv (z < 1) \gg \mathit{down}; G_2 \\
C_2 &\equiv \mathit{exit}; \{\!|y|\!\}\,C_3 & \qquad G_2 &\equiv \mathit{raise}; \{\!|z|\!\}\,G_3 \\
C_3 &\equiv (y < 1) \gg \mathit{raise}; C_0 & \qquad G_3 &\equiv (z < 2) \gg (z > 1) \mapsto \mathit{up}; G_0
\end{aligned}$$

Using the expansion law we calculate the following.

$$(T_0 \,||_{\varnothing}\, G_0) \,||_A\, C_0 = \mathit{appr}; \{\!|x, y|\!\}\, ((T_1 \,||_{\varnothing}\, G_0) \,||_A\, C_1)$$

$$(T_1 \,||_{\varnothing}\, G_0) \,||_A\, C_1 = (x < 5 \wedge y \leq 1) \gg (\ (x \geq 2) \mapsto \mathit{in}; ((T_2 \,||_{\varnothing}\, G_0) \,||_A\, C_1) + (y \geq 1) \mapsto \mathit{lower}; \{\!|z|\!\}\, ((T_1 \,||_{\varnothing}\, G_1) \,||_A\, C_2)\ )$$

$$(T_1 \,||_{\varnothing}\, G_1) \,||_A\, C_2 = (x < 5 \wedge z < 1) \gg (\ (x \geq 2) \mapsto \mathit{in}; ((T_2 \,||_{\varnothing}\, G_1) \,||_A\, C_2) + \mathit{down}; ((T_1 \,||_{\varnothing}\, G_2) \,||_A\, C_2)\ )$$

$$(T_1 \,||_{\varnothing}\, G_2) \,||_A\, C_2 = (x < 5) \gg (x \geq 2) \mapsto \mathit{in}; ((T_2 \,||_{\varnothing}\, G_2) \,||_A\, C_2)$$

$$(T_2 \,||_{\varnothing}\, G_2) \,||_A\, C_2 = (x < 5) \gg \mathit{out}; ((T_3 \,||_{\varnothing}\, G_2) \,||_A\, C_2)$$

$$(T_3 \,||_{\varnothing}\, G_2) \,||_A\, C_2 = (x < 5) \gg \mathit{exit}; \{\!|y|\!\}\, ((T_4 \,||_{\varnothing}\, G_2) \,||_A\, C_3)$$

$$(T_4 \,||_{\varnothing}\, G_2) \,||_A\, C_3 = (x < 5 \wedge y < 1) \gg (\ \tau; ((T_0 \,||_{\varnothing}\, G_2) \,||_A\, C_3) + \mathit{raise}; \{\!|z|\!\}\, ((T_4 \,||_{\varnothing}\, G_3) \,||_A\, C_0)\ )$$

$$(T_0 \,||_{\varnothing}\, G_2) \,||_A\, C_3 = (y < 1) \gg \mathit{raise}; \{\!|z|\!\}\, ((T_0 \,||_{\varnothing}\, G_3) \,||_A\, C_0)$$

$$(T_4 \,||_{\varnothing}\, G_3) \,||_A\, C_0 = (x < 5 \wedge z < 2) \gg (\ \tau; ((T_0 \,||_{\varnothing}\, G_3) \,||_A\, C_0) + (z > 1) \mapsto \mathit{up}; ((T_4 \,||_{\varnothing}\, G_0) \,||_A\, C_0)\ )$$

$$(T_0 \,||_{\varnothing}\, G_3) \,||_A\, C_0 = (z < 2) \gg (\ \mathit{appr}; \{\!|x, y|\!\}\, ((T_1 \,||_{\varnothing}\, G_3) \,||_A\, C_1) + (z > 1) \mapsto \mathit{up}; ((T_0 \,||_{\varnothing}\, G_0) \,||_A\, C_0)\ )$$

$$(T_4 \,||_{\varnothing}\, G_0) \,||_A\, C_0 = (x < 5) \gg \tau; ((T_0 \,||_{\varnothing}\, G_0) \,||_A\, C_0)$$

$$(T_1 \,||_{\varnothing}\, G_3) \,||_A\, C_1 = (x < 5 \wedge y \leq 1 \wedge z < 2) \gg (\ (x \geq 2) \mapsto \mathit{in}; ((T_2 \,||_{\varnothing}\, G_3) \,||_A\, C_1) + (z > 1) \mapsto \mathit{up}; ((T_1 \,||_{\varnothing}\, G_0) \,||_A\, C_1)\ )$$

Using the axioms **D1**, **D2**, **CR2**, **L1**, **L2**, **Stp**, **A4**, and Property 6.2, we can eliminate summands that will never be reached, thus obtaining the following system of equations (we indicate with a (*) those equations that differ from the previous ones):

$$(T_0 \,||_{\varnothing}\, G_0) \,||_A\, C_0 = \mathit{appr}; \{\!|x|\!\}\, ((T_1 \,||_{\varnothing}\, G_0) \,||_A\, \xi_{[x/y]} C_1) \qquad (*)$$

$$(T_1 \,||_{\varnothing}\, G_0) \,||_A\, \xi_{[x/y]} C_1 = (x \leq 1) \gg (x = 1) \mapsto \mathit{lower}; \{\!|z|\!\}\, (x - z = 1) \gg ((T_1 \,||_{\varnothing}\, G_1) \,||_A\, C_2) \qquad (*)$$

$$(x - z = 1) \gg ((T_1 \,||_{\varnothing}\, G_1) \,||_A\, C_2) = (x - z = 1 \wedge z < 1) \gg \mathit{down}; ((T_1 \,||_{\varnothing}\, G_2) \,||_A\, C_2) \qquad (*)$$

$$(T_1 \,||_{\varnothing}\, G_2) \,||_A\, C_2 = (x < 5) \gg (x \geq 2) \mapsto \mathit{in}; ((T_2 \,||_{\varnothing}\, G_2) \,||_A\, C_2)$$

$$(T_2 \,||_{\varnothing}\, G_2) \,||_A\, C_2 = (x < 5) \gg \mathit{out}; ((T_3 \,||_{\varnothing}\, G_2) \,||_A\, C_2)$$

$$(T_3 \,||_{\varnothing}\, G_2) \,||_A\, C_2 = (x < 5) \gg \mathit{exit}; \{\!|y|\!\}\, ((T_4 \,||_{\varnothing}\, G_2) \,||_A\, C_3)$$

$$(T_4 \,||_{\varnothing}\, G_2) \,||_A\, C_3 = (x < 5 \wedge y < 1) \gg (\ \tau; ((T_0 \,||_{\varnothing}\, G_2) \,||_A\, C_3) + \mathit{raise}; \{\!|z|\!\}\, ((T_4 \,||_{\varnothing}\, G_3) \,||_A\, C_0)\ )$$

$$(T_0 \,||_{\varnothing}\, G_2) \,||_A\, C_3 = (y < 1) \gg \mathit{raise}; \{\!|z|\!\}\, ((T_0 \,||_{\varnothing}\, G_3) \,||_A\, C_0)$$

$$(T_4 \,||_{\varnothing}\, G_3) \,||_A\, C_0 = (x < 5 \wedge z < 2) \gg (\ \tau; ((T_0 \,||_{\varnothing}\, G_3) \,||_A\, C_0) + (z > 1) \mapsto \mathit{up}; ((T_4 \,||_{\varnothing}\, G_0) \,||_A\, C_0)\ )$$

$$(T_0 \,||_{\varnothing}\, G_3) \,||_A\, C_0 = (z < 2) \gg (\ \mathit{appr}; \{\!|x|\!\}\, (z - x \geq 0) \gg ((T_1 \,||_{\varnothing}\, G_3) \,||_A\, \xi_{[x/y]} C_1) + (z > 1) \mapsto \mathit{up}; ((T_0 \,||_{\varnothing}\, G_0) \,||_A\, C_0)\ ) \qquad (*)$$

$$(T_4 \,||_{\varnothing}\, G_0) \,||_A\, C_0 = (x < 5) \gg \tau; ((T_0 \,||_{\varnothing}\, G_0) \,||_A\, C_0)$$

$$(z - x \geq 0) \gg ((T_1 \,||_{\varnothing}\, G_3) \,||_A\, \xi_{[x/y]} C_1) = (z - x \geq 0 \wedge x \leq 1 \wedge z < 2) \gg (z > 1) \mapsto \mathit{up}; ((T_1 \,||_{\varnothing}\, G_0) \,||_A\, \xi_{[x/y]} C_1) \qquad (*)$$

As a consequence, by RSP and RDP$^-$, $\mathit{Spec}_0 = \mathit{RRCrossing}$ where Spec_0 is defined by the following equations.

$$\mathit{Spec}_0 = \mathit{appr}; \{\!|x|\!\}\, \mathit{Spec}_1$$

$$\mathit{Spec}_1 = (x \leq 1) \gg (x = 1) \mapsto \mathit{lower}; \{\!|z|\!\}\, \mathit{Spec}_2$$

$$\mathit{Spec}_2 = (x - 1 = z < 1) \gg \mathit{down}; \mathit{Spec}_3$$

$$\mathit{Spec}_3 = (x < 5) \gg (x \geq 2) \mapsto \mathit{in}; \mathit{Spec}_4$$

$$\mathit{Spec}_4 = (x < 5) \gg \mathit{out}; \mathit{Spec}_5$$

$$\mathit{Spec}_5 = (x < 5) \gg \mathit{exit}; \{\!|y|\!\}\, \mathit{Spec}_6$$

$$\mathit{Spec}_6 = (x < 5 \wedge y < 1) \gg (\ \tau; \mathit{Spec}_7 + \mathit{raise}; \{\!|z|\!\}\, \mathit{Spec}_8\ )$$

$$\mathit{Spec}_7 = (y < 1) \gg \mathit{raise}; \{\!|z|\!\}\, \mathit{Spec}_9$$

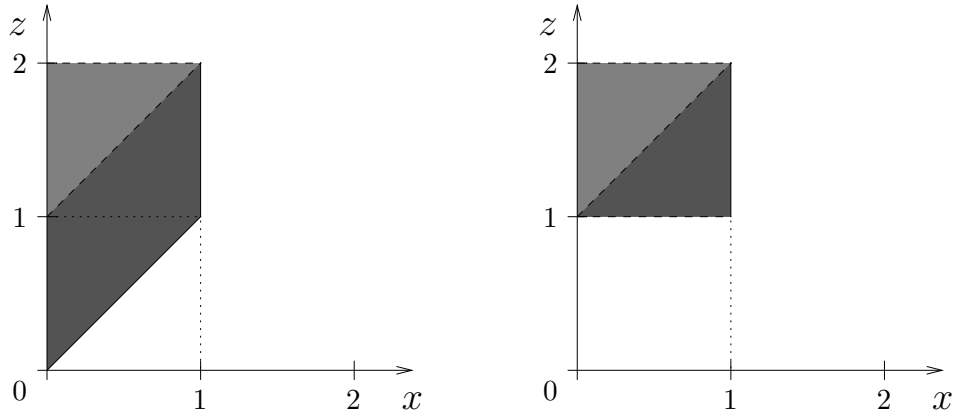
$$\mathit{Spec}_8 = (x < 5 \wedge z < 2) \gg (\ \tau; \mathit{Spec}_9 + (z > 1) \mapsto \mathit{up}; \mathit{Spec}_{10}\ )$$

$$\mathit{Spec}_9 = (z < 2) \gg (\ \mathit{appr}; \{\!|x|\!\}\, \mathit{Spec}_{11} + (z > 1) \mapsto \mathit{up}; \mathit{Spec}_0\ )$$

$$\mathit{Spec}_{10} = (x < 5) \gg \tau; \mathit{Spec}_0$$

$$\mathit{Spec}_{11} = (x \leq 1 \wedge x \leq z < 2) \gg (z > 1) \mapsto \mathit{up}; \mathit{Spec}_1$$

Some observations are in order. First, notice that the *Gate* closes and opens for each train. In particular, this can be observed in process Spec_9 in which action *appr* can take place and move to Spec_{11} that can perform only action *up*. The second observation is less obvious but refers to the same part of the specification. As we mentioned before, process $\mathit{Spec}_{11} = (z - x \geq 0) \gg ((T_1 \,||_{\varnothing}\, G_3) \,||_A\, \xi_{[x/y]} C_1)$ can only execute action *up*, which is an action that is independently executed by the *Gate* (in particular, process G_3). However, the execution of action *up* is not only controlled by clock z (which is local to process *Gate*) but also by x (which controls the activity of *Train*$'$ and *Controller*, but not of *Gate*).



(a) At process $Spec_{11}$ (b) *up* is enabled

Figure 7.2: On the enabling of action *up*

Figure 7.2(a) depicts the zone in which the invariant of $Spec_{11}$ holds, that is, the part of the plane in which the pair (x, z) takes those values that satisfy the invariant $(x \leq 1 \wedge x \leq z < 2)$. Figure 7.2(b) depicts the zone in which action *up* is enabled, that is, the zone in which the invariant and the guard are satisfied at the same time: $(x \leq 1 \wedge x \leq z < 2) \wedge (z > 1)$ (see Property 6.2.5). Notice that if $z - x < 1$, then we are in the shaded part of both zones in Figure 7.2. Under these conditions, the eventual execution of action *up* only depends on x since it is the constraint $x \leq 1$ the one that impose the deadline. This is an undesirable situation since the opening of the barrier, which is an activity private to the *Gate*, depends on the need of the *Controller* to synchronise on the signal *lower*, this being the reason why the constraint $x \leq 1$ appears.

An improved specification Although the *Train* is part of the system, it is not a component that can be easily subjected to modifications due to its more "autonomous" behaviour. Instead, the *Controller* is an electronic or software-driven device and the *Gate* is a simple machinery that combines some electronic part and a mechanic component that lifts the barrier. Therefore, these two components are those we can afford to change. Nevertheless, it is supposed that the timing aspects cannot be varied in the sense that faster response may not be guaranteed by the machinery. So, we proceed by adequately accepting signals in different parts of the *Controller* and the *Gate*. This activity is usually done by means of hardware interruptions, and should be already present in the system.

The first problem was that the *Gate* would close and open for each *Train* that passes instead of smartly staying closed if two *Train*s pass the crossing in a sufficiently short period. Part of the problem is located in the *Controller* that may give the signal to *raise* the *Gate* even if a *Train* is intending to indicate that it is approaching. To avoid this situation we allow the *Controller* to be preempted by the *appr*oaching signal before the

action *raise* takes place.

$$\begin{aligned} \mathit{NewController} &= \mathit{appr}; \mathsf{urgent}_1(\mathit{lower}; \mathit{ClosingSignaled}) \\ \mathit{ClosingSignaled} &= \mathit{exit}; \mathsf{before}_1(\ \mathit{raise}; \mathit{NewController} \\ &\qquad + \mathit{appr}; \mathit{ClosingSignaled}\ ) \end{aligned}$$

The second change we make also corrects the second problem in which the opening of the *Gate* (action *up*) is undesirably influenced by the *Controller*. We proceed by overriding the opening of the *Gate* if the signal "*lower*" is sent before the rising of the *Gate* is completed (action *up*).

$$\begin{aligned} \mathit{NewGate} &= \mathit{lower}; \mathit{GateClosing} \\ \mathit{GateClosing} &= \mathsf{before}_1(\mathit{down}; \mathit{raise}; \mathsf{between}_{(1,2)}(\ \mathit{up}; \mathit{NewGate} \\ &\qquad + \mathit{lower}; \mathit{GateClosing}\ )\ ) \end{aligned}$$

In the same way we proceeded before, we can prove that

$$\begin{aligned} &(\mathit{Train} \,||_{\varnothing}\, \mathit{NewGate}) \,||_{A}\, \mathit{NewController} \\ &\qquad = \quad (\mathit{Train}' \,||_{\varnothing}\, \mathit{NewGate}) \,||_{A}\, \mathit{NewController} \quad = \quad \mathit{NewSpec}_0 \end{aligned}$$

where $\mathit{NewSpec}_0$ is defined by the following recursive specification

$$\begin{aligned} \mathit{NewSpec}_0 &= \mathit{appr}; \{\!|x|\!\}\, \mathit{NewSpec}_1 \\ \mathit{NewSpec}_1 &= (x \leq 1) \gg (x = 1) \mapsto \mathit{lower}; \{\!|z|\!\}\, \mathit{NewSpec}_2 \\ \mathit{NewSpec}_2 &= (x - 1 = z < 1) \gg \mathit{down}; \mathit{NewSpec}_3 \\ \mathit{NewSpec}_3 &= (x < 5) \gg (x \geq 2) \mapsto \mathit{in}; \mathit{NewSpec}_4 \\ \mathit{NewSpec}_4 &= (x < 5) \gg \mathit{out}; \mathit{NewSpec}_5 \\ \mathit{NewSpec}_5 &= (x < 5) \gg \mathit{exit}; \{\!|y|\!\}\, \mathit{NewSpec}_6 \\ \mathit{NewSpec}_6 &= (x < 5 \wedge y < 1) \gg (\ \tau; \mathit{NewSpec}_7 + \mathit{raise}; \{\!|z|\!\}\, \mathit{NewSpec}_8\ ) \\ \mathit{NewSpec}_7 &= (y < 1) \gg (\ \mathit{appr}; \{\!|x|\!\}\, \mathit{NewSpec}_3 + \mathit{raise}; \{\!|z|\!\}\, \mathit{NewSpec}_9\ ) \\ \mathit{NewSpec}_8 &= (x < 5 \wedge z < 2) \gg (\ \tau; \mathit{NewSpec}_9 + (z > 1) \mapsto \mathit{up}; \mathit{NewSpec}_{10}\ ) \\ \mathit{NewSpec}_9 &= (z < 2) \gg (\ \mathit{appr}; \{\!|x|\!\}\, \mathit{NewSpec}_{11} + (z > 1) \mapsto \mathit{up}; \mathit{NewSpec}_0\ ) \\ \mathit{NewSpec}_{10} &= (x < 5) \gg \tau; \mathit{NewSpec}_0 \\ \mathit{NewSpec}_{11} &= (x \leq 1 \wedge x \leq z < 2) \gg (\ (x = 1) \mapsto \mathit{lower}; \{\!|z|\!\}\, \mathit{NewSpec}_2 \\ &\qquad + (z > 1) \mapsto \mathit{up}; \mathit{NewSpec}_1\ ) \end{aligned}$$

In particular, notice that in $\mathit{NewSpec}_{11}$, action *up* is not forced to occur within the constraint $x \leq 1$ anymore, since when clock x takes value 1 action *lower* presents an alternative execution step.

7.3 Regular Processes and their Timed Automata

In this section, we give a syntactic characterisation of *finite* timed automata. We show that regular processes are as expressive as finite timed automata. This connection is evident between ♧ and labelled transition systems. However, though the relation between ♡ and the timed automata model is not unexpected, its proof is rather far from being trivial. Basically, the problems appear because ♡ contains clock variables and the notion of binding. Let us consider some examples. For some expressions like

$$Z = \{\!|x|\!\}\,(x \leq 2) \gg (x \geq 1) \mapsto a; Z \qquad \Longrightarrow \qquad \begin{array}{c} x \\ \hline x \leq 2 \end{array} \quad a, x \geq 1$$

the respective timed automaton is straightforwardly obtained using the direct semantics. However, for some other expressions we need to resort to the semantics up to α-conversion, since an attempt to use the direct semantics would induce undesirable consequences. For instance, in

$$X = (x < 3) \gg \{\!|x|\!\}\,(x < 2) \gg a; X \qquad \not\Longrightarrow \qquad \begin{array}{c} x \\ \hline x < 3 \\ x < 2 \end{array} \quad a, \mathbf{tt} \tag{7.1}$$

a naive attempt to associate a timed automaton to X will derive in the one depicted on the right-hand side. However, this is not correct since the variable x is incorrectly bound. Less obvious is the following case, where $\phi(x, y)$ and $\phi'(x, y)$ are some constraints containing clock x and y.

$$Y = (\{\!|x|\!\}\,\phi(x, y) \mapsto a; Y) + (\{\!|y|\!\}\,\phi'(x, y) \mapsto b; Y)$$

$$\not\Longrightarrow \qquad a, \phi(x, y) \quad \begin{array}{c} x, y \\ \hline \mathbf{tt} \end{array} \quad b, \phi'(x, y) \tag{7.2}$$

In this expression, the x on the right-hand side of $+$ should not be bound to the resetting on the left-hand side, and symmetrically for y. Compare with the naively associated timed automaton that is depicted next to the equation.

Yet both processes X and Y are regular and therefore we expect them to represent finite timed automata. In the following, we show that for every regular recursive specification we can find an equivalent one that does not have conflict of variables and whose direct semantics defines a finite timed automaton. Moreover, we prove this by using only the axiom system and unfolding of recursive equations. This shows the power of our equational theory. It is also important to remark that all the proofs are basically algorithms, thus we

are also providing an effective method to translate one model into the other. A curious corollary of these proofs is that regular recursive specifications may need one clock less than timed automata in order to represent the same process.

Since we only deal with regular recursive specifications, in this section we restrict to sequential processes (i.e. the subset $\heartsuit^{br}$).

To proceed with the proof we use the following strategy. First, we rewrite the recursive specification into a new one such that each variable, according to the new definition, can execute at most one action and move to another variable. Afterwards, we translate this last specification into a recursive specification that trivially resembles a timed automaton.

We say that a guarded recursive specification E is in *one-step normal form* if for every $X = p \in E$, p is in the language defined by the following grammar,

$$p ::= \quad \mathbf{0} \quad | \quad a; X \quad | \quad \phi \mapsto p \quad | \quad p + p \quad | \quad \{\!|C|\!\}\, p \quad | \quad \psi \gg p \tag{7.3}$$

where $a \in \mathcal{A}$, $\phi \in \Phi$, $\psi \in \overline{\Phi}$, $C \subseteq \mathcal{C}$, and $X \in \mathcal{V}$.

We recall that according to Definition 2.14, a recursive specification E is strictly guarded if the right-hand side of every recursive equation in it is a guarded process, i.e., for all $X = p \in E$, every occurrence of a process variable in p appears in the context of a prefix. Moreover, we define E to be guarded if it can be rewritten into a strictly guarded recursive specification by properly unfolding the equations. If in addition E defines finitely many recursive equations, then E is regular.

The following proposition, which is standard for process algebra (see, e.g. Baeten and Weijland, 1990), gives an alternative characterisation of guarded recursive specifications.

Proposition 7.1. *Let E be a recursive specification with process variables in $\mathcal{V}$. Define the* unguarded variable dependency graph *G with nodes in $\mathcal{V}$ and edges $X \to Y$ if and only if Y is unguarded in $X = p \in E$. Then E is guarded if and only if $\to$ is* well founded*, i.e., there is no infinite chain $X_1 \to X_2 \to X_3 \to \ldots$ in G.*

The next lemma addresses the first part of our proof:

Lemma 7.2. *Any regular recursive specification E can be rewritten into a recursive specification $\overline{E}$ that is in one-step normal form such that the root variables of both specifications can be proved equal.*

Proof. The proof follows by adding new process variables, and folding and unfolding when needed. We proceed in two steps. First, we reduce the original specification E into a new specification $\overline{E}_0$ such that no prefixing occurs in the scope of another prefixing. That is done by creating new recursive equations. In the second step, $\overline{E}_0$ is put into one-step normal form by unfolding all the variables that occur unguarded.

Let $E_0 \equiv E$ be the given regular recursive specification. For $X = p \in E_0$, if p has an occurrence $a; q$ where q is not a process variable, we define $\mathit{size}(X)$ to be the number of symbols in p which are not process variables. Otherwise, $\mathit{size}(X) = 0$. Choose $X = p \in E_0$ such that $\mathit{size}(X) \geq \mathit{size}(Z)$ for all $Z = r \in E_0$, i.e., X is maximal according to size. If $\mathit{size}(X) > 0$, then p has an occurrence $a; q$ where q is not a process variable. In this case,

choose a fresh process variable Y and define p' to be p with the occurrence of $a;q$ replaced by $a;Y$. Define

$$E_1 \stackrel{\text{def}}{=} (E_0 - \{X = p\}) \cup \{X = p', Y = q\}$$

By RSP and RDP$^-$, process variables defined in E_0 represent the same processes as the variables with equal name defined in E_1. We repeat the construction until $size(X) = 0$ for every process variable X. The algorithm eventually terminates because the tuple

$$\langle \mathrm{MAX}(E_i), \#\{X = p \in E_i \mid size(X) = \mathrm{MAX}(E_i)\}\rangle,$$

with $\mathrm{MAX}(E_i) = \max\{size(X) \mid X = p \in E_i\}$, strictly decreases in each iteration i according to the lexicographical order.

Let $\overline{E}_0$ be the output of the previous algorithm. Clearly, $\overline{E}_0$ is also a regular specification. Notice that for every equation $X = q \in \overline{E}_0$, by construction, q is in the language defined by the grammar

$$p ::= \quad \mathbf{0} \quad | \quad a;X \quad | \quad \phi \mapsto p \quad | \quad p + p \quad | \quad \{\!|C|\!\}\, p \quad | \quad \psi \gg p \quad | \quad X$$

that is, q is "almost" (but not yet) in one-step normal form since it may have unguarded variables. Construct the unguarded variable dependency graph G_0 according to Proposition 7.1. Since $\overline{E}_0$ is regular, G_0 is acyclic and finite. Choose a leaf Y in G_0 with $Y = q \in \overline{E}_0$. For every $X = p \in \overline{E}_0$ such that $X \to Y$ define p' to be p with every unguarded occurrence of Y replaced by q. Define

$$\overline{E}_1 \stackrel{\text{def}}{=} (\overline{E}_0 - \bigcup\{X = p \mid X \to Y\}) \cup \bigcup\{X = p' \mid X \to Y\}$$

Notice that the unguarded dependency graph G_1 of $\overline{E}_1$ is the same as G_0 where all edges $X \to Y$ were removed. We repeat the process until the set of edges is empty. The algorithm eventually terminates since the number of edges is strictly decreasing in each iteration.

It is easy to check that the output of this last algorithm is a recursive specification $\overline{E}$ in one-step normal form. Moreover, since we proceeded by simply folding and unfolding recursive equations, the root variable X_0 of $\overline{E}$ can be proved equal to the root variable X_0 of E by using RSP and RDP$^-$. *Q.E.D.*

We say that a recursive specification E is in *TA-normal form* if it is regular and for every $X = p \in E$, p has the form

$$\{\!|x|\!\}\, \psi \gg \textstyle\sum_{i \in I} \phi_i \mapsto a_i; X_i \tag{7.4}$$

where I is a finite index set, $\psi \in \overline{\Phi}$, $x \in \mathcal{C}$, and for all $i \in I$, $a_i \in \mathcal{A}$, $\phi_i \in \Phi$, and $X_i \in \mathcal{V}$.

Notice that a recursive specification in TA-normal form represents a finite timed automaton. We can observe that each variable represents a location —hence the number of locations is finite— and for each of them we have defined exactly one resetting and one

invariant. Moreover, the summation defines the outgoing edges labelled with the respective guard and action. (See the proof of Theorem 5.18.)

By the shape of process (7.4), it should be clear that a recursive specification in TA-normal form does not have conflict of variables.

In the following, we state and prove the main theorem of this section.

Theorem 7.3. *Any regular specification can be rewritten into TA-normal form by using folding, unfolding, renaming of clock variables and the set of axioms* $ax(\heartsuit^b) - \mathbf{D}$*, such that their respective root variables are proved equal using RSP and* RDP^-*.*

Proof. Let E be a regular specification with root X_0. Because of Lemma 7.2, we may assume that E is already in one-step normal form. Let E' be the recursive specification defined as follows. For every process variable X defined in E with free variables $fv(X) = \{x_1, \ldots, x_n\}$, define a process variable $X_{x_1,\ldots,x_n}$. For every $X = p \in E$ define $X_{x_1,\ldots,x_n} = p' \in E'$ where p' is the same as p but with all process variables subindexed with their respective free clock variables. So, E' is the same as E with the names of the process variables changed.

We will need to consider renaming of clock variables. We do that in the expected way. In particular, the case of process variables reduces to an appropriate renaming of the subindices. Thus, if $\{y/x_i\}$ is the substitution that replacees x_i by y, then

$$\{y/x_i\}(X_{x_1,\ldots,x_i,\ldots,x_n}) \stackrel{\text{def}}{=} X_{x_1,\ldots,y,\ldots,x_n} \tag{7.5}$$

Let $\mathcal{C}$ be the set of all clocks occurring in E. We will consider a new clock variable $\overline{x} \notin \mathcal{C}$. Let $\mathcal{V}$ be the original set of process variables defined in E. We define

$$\mathcal{V}' \stackrel{\text{def}}{=} \{X_{x_1,\ldots,x_n} \mid X \in \mathcal{V} \wedge n = \#fv(X) \wedge \forall i \in \{1,\ldots,n\}.\, x_i \in \mathcal{C} \cup \{\overline{x}\}\}$$

From now on, we let $\overline{X}, \overline{X}_i, \ldots$ range over $\mathcal{V}'$. The necessity of the new clock $\overline{x}$ will be clarified later.

For each $X_{x_1,\ldots,x_n} = p \in E'$ we show that p can be proved equal to a term $\{\!|\overline{x}|\!\}\, \psi \gg \sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i$. We proceed by induction according to the grammar in (7.3). We calculate the most significant cases. They are summation and clock resetting.

Case $p + q$. By induction hypothesis, assume $p = \{\!|\overline{x}|\!\}\, \psi \gg \sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i$ and $q = \{\!|\overline{x}|\!\}\, \psi' \gg \sum_{j\in J} \phi'_j \mapsto b_j; \overline{Y}_j$. If $I \neq \varnothing$ and $J \neq \varnothing$, then

$$\begin{aligned}
p + q &\stackrel{\text{ind.}}{=} \left(\{\!|\overline{x}|\!\}\, \psi \gg \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i\right) + \left(\{\!|\overline{x}|\!\}\, \psi' \gg \textstyle\sum_{j\in J} \phi'_j \mapsto b_j; \overline{Y}_j\right) \\
&\stackrel{\text{CR4,I4}}{=} \{\!|\overline{x}|\!\} \left(\left(\textstyle\sum_{i\in I} \psi \gg \phi_i \mapsto a_i; \overline{X}_i\right) + \left(\textstyle\sum_{j\in J} \psi' \gg \phi'_j \mapsto b_j; \overline{Y}_j\right)\right) \\
&\stackrel{\text{A1–A3}}{=} \{\!|\overline{x}|\!\} \Big(\left(\textstyle\sum_{i\in I} (\psi \gg \phi_i \mapsto a_i; \overline{X}_i) + (\psi' \gg \phi'_1 \mapsto b_j; \overline{Y}_1)\right) \\
&\qquad + \left(\textstyle\sum_{j\in J} (\psi' \gg \phi'_j \mapsto b_j; \overline{Y}_j) + (\psi \gg \phi_1 \mapsto a_i; \overline{X}_1)\right) \Big)
\end{aligned}$$

$$\begin{aligned}
&\overset{\text{I5,I4}}{=} \quad \{\!|\overline{x}|\!\}\,(\psi \vee \psi') \rrhd \Big(\ \Big(\textstyle\sum_{i\in I}(\psi \wedge \phi_i) \mapsto a_i; \overline{X}_i + (\psi' \wedge \phi'_1) \mapsto b_j; \overline{Y}_1\Big) \\
&\qquad\qquad + \Big(\textstyle\sum_{j\in J}(\psi' \wedge \phi'_j) \mapsto b_j; \overline{Y}_j + (\psi \wedge \phi_1) \mapsto a_i; \overline{X}_1\Big) \ \Big) \\
&\overset{\text{A1–A3}}{=} \quad \{\!|\overline{x}|\!\}\,(\psi \vee \psi') \rrhd \Big(\ \Big(\textstyle\sum_{i\in I}(\psi \wedge \phi_i) \mapsto a_i; \overline{X}_i\Big) \\
&\qquad\qquad + \Big(\textstyle\sum_{j\in J}(\psi' \wedge \phi'_j) \mapsto b_j; \overline{Y}_j\Big) \ \Big)
\end{aligned}$$

The other cases are similar, but we should take into account that $\{\!|\overline{x}|\!\}\,\psi \rrhd \mathbf{0} = \{\!|\overline{x}|\!\}\,\psi \rrhd \mathbf{ff} \mapsto a; \overline{Z}$ for some $a \in \mathcal{A}$ and $\overline{Z} \in \mathcal{V}'$.

Case $\{\!|C|\!\}\,p$. By induction hypothesis assume $p = \{\!|\overline{x}|\!\}\,\psi \rrhd \sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i$. Then

$$\begin{aligned}
\{\!|C|\!\}\,p \quad &\overset{\text{ind.}}{=} \quad \{\!|C|\!\}\,\{\!|\overline{x}|\!\}\,\psi \rrhd \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i \\
&\overset{\text{CR3}}{=} \quad \{\!|C \cup \{\overline{x}\}|\!\}\,\psi \rrhd \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i \\
&\overset{\text{CR2}}{=} \quad \{\!|\overline{x}|\!\}\,(\xi_{\{\overline{x}/C\}}(\psi)) \rrhd \textstyle\sum_{i\in I}(\xi_{\{\overline{x}/C\}}(\phi_i)) \mapsto a_i; \xi_{\{\overline{x}/C\}}(\overline{X}_i)
\end{aligned}$$

By replacing p by its equivalent TA-normal form in each equation $X_{x_1,\ldots,x_n} = p \in E'$, we obtain a new set of equations E'', each one with the form

$$X_{x_1,\ldots,x_n} = \{\!|\overline{x}|\!\}\,\psi' \rrhd \textstyle\sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i \tag{7.6}$$

However, the new set of equations is not a recursive specification since there can be some process variables $\overline{X}_i$ that have not yet been defined. They come from case $\{\!|C|\!\}\,p$, in particular, as a consequence of renaming clock variables when using axiom **CR2**.

The definition of an undefined variable $X_{y_1,\ldots,y_n}$ is given by appropriately renaming the free clocks in the recursive equation $X_{x_1,\ldots,x_n} = \{\!|\overline{x}|\!\}\,\psi' \rrhd \sum_{i\in I} \phi_i \mapsto a_i; \overline{X}_i \in E''$.

Notice that for each equation (7.6) in E'', we need $n+1$ clock variables: there are n free variables $x_1, \ldots, x_n$ and one clock resetting of $\overline{x}$. Suppose that one of $y_1, \ldots, y_n$ is $\overline{x}$. Then, choose $\overline{y} \in \mathcal{C} \cup \{\overline{x}\}$ to be a clock other than $\{y_1, \ldots, y_n\}$. If instead none of $y_1, \ldots, y_n$ is $\overline{x}$, we take $\overline{y} = \overline{x}$. Now, $X_{y_1,\ldots,y_n}$ can be defined as follows

$$X_{y_1,\ldots,y_n} = \{\!|\overline{y}|\!\}\ (\{\vec{y}/\vec{x}\}(\psi')) \rrhd \textstyle\sum_{i\in I}(\{\vec{y}/\vec{x}\}(\phi_i)) \mapsto a_i; \{\vec{y}/\vec{x}\}(\overline{X}_i) \tag{7.7}$$

where $\{\vec{y}/\vec{x}\}$ is the simultaneous substitution $\{y_1/x_1, \ldots, y_n/x_n, \overline{y}/\overline{x}\}$.

The recursive specification containing the recursive equations defined in (7.6) and (7.7) defines all variables in $\mathcal{V}'$, is equivalent to E and is in TA-normal form.

Moreover, every process variable X defined in E with $fv(X) = \{x_1, \ldots, x_n\}$, can be proved equal to the new defined variable $X_{x_1,\ldots,x_n}$ using RSP and RDP$^-$, from which equality of root variables follows. *Q.E.D.*

As an example of the proof of Theorem 7.3, we translate the recursive equation (7.2) into TA-normal form. First, we define the new equation

$$Y_{x,y} = (\{\!|x|\!\}\,\phi(x,y) \mapsto a; Y_{x,y}) + (\{\!|y|\!\}\,\phi'(x,y) \mapsto b; Y_{x,y})$$

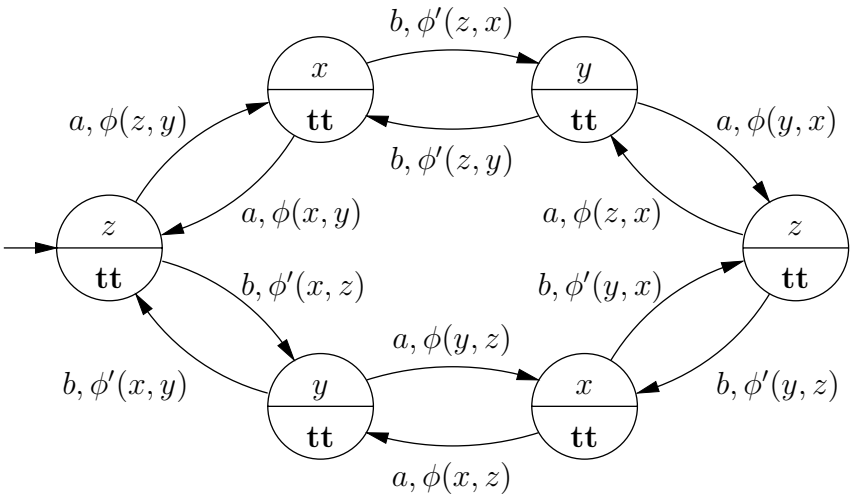


Figure 7.3: $Y = (\{\!|x|\!\}\, \phi(x, y) \mapsto a; Y) + (\{\!|y|\!\}\, \phi'(x, y) \mapsto b; Y)$

according to the first part of the proof. Consider a new clock z and $\mathcal{V}' = \{Y_{u,v} \mid u, v \in \{x, y, z\}\}$. Following the inductive calculation, we obtain

$$\begin{aligned} Y_{x,y} &= (\{\!|x|\!\}\, \phi(x, y) \mapsto (\{\!|z|\!\}\, \mathbf{tt} \gg \mathbf{tt} \mapsto a; Y_{x,y})) \\ &\quad +(\{\!|y|\!\}\, \phi'(x, y) \mapsto (\{\!|z|\!\}\, \mathbf{tt} \gg \mathbf{tt} \mapsto b; Y_{x,y})) \\ &= (\{\!|x|\!\}\, (\{\!|z|\!\}\, \mathbf{tt} \gg \phi(x, y) \mapsto a; Y_{x,y})) + (\{\!|y|\!\}\, (\{\!|z|\!\}\, \mathbf{tt} \gg \phi'(x, y) \mapsto b; Y_{x,y})) \\ &= \{\!|z|\!\}\, \mathbf{tt} \gg (\phi(z, y) \mapsto a; Y_{z,y} + \phi'(x, z) \mapsto b; Y_{x,z}) \end{aligned}$$

According to (7.7), we can calculate, for instance

$$Y_{z,x} = \{\!|y|\!\}\, \mathbf{tt} \gg (\phi(y, x) \mapsto a; Y_{y,x} + \phi'(z, y) \mapsto b; Y_{z,y})$$

In this case, variables $Y_{x,x}$, $Y_{y,y}$, and $Y_{z,z}$ are redundant since they are never be reached from the root. At this point, the reader should not find difficulties to check that the semantics of process $Y_{x,y} = Y$ is the timed automaton depicted in Figure 7.3.

Since every regular specification can be rewritten into TA-normal form according to Theorem 7.3, and every recursive specification in TA-normal form defines trivially a finite timed automaton as it was already observed, we obtain the following corollary.

Corollary 7.3.1. *Any process with variables defined in a regular recursive specification can be proved symbolically bisimilar to a finite timed automaton.*

The reason why the corollary states symbolic bisimulation is that in the proof of Theorem 7.3, we only use the set of axioms $ax(\heartsuit^b) - \mathbf{D}$ which is sound for $\sim_{\&}$.

As a second corollary we have that the same expressive power of the timed automata is preserved if we restrict to those automata that reset only one clock in each location, i.e., $\#\kappa(s) \leq 1$. So, by Theorem 5.18, Theorem 7.3 and the observation below equation (7.4), we have:

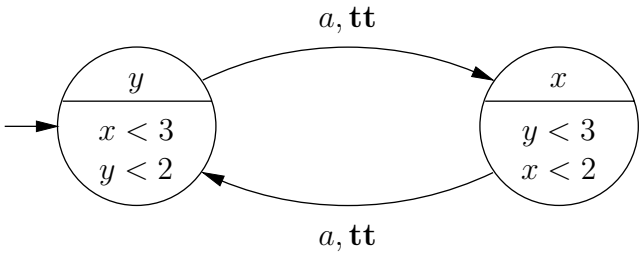


Figure 7.4: $X = (x < 3) \rhd\!\!\rhd \{\!|x|\!\} (x < 2) \rhd\!\!\rhd a; X$

Corollary 7.3.2. *Every finite timed automaton TA is symbolic bisimilar to some finite timed automaton TA′ such that at most one clock is reset in every location of TA′.*

A detail to remark is that regular specifications may need fewer clocks than finite timed automata in order to represent the same process. A clear example showing that fact is the expression (7.1) given above. The corresponding timed automaton is depicted in Figure 7.4. Notice that both clocks x and y are necessary.

However, we have the interesting result that in order to obtain a finite timed automaton from a regular recursive specification, we need to add at most one new clock. This follows from the observation that in Lemma 7.2 we do not modify the set $\mathcal{C}$, and to prove Theorem 7.3 we are only required to consider at most one additional clock. Therefore, we can state the following corollary.

Corollary 7.3.3. *Let E be a regular recursive specification with root X_0 and clocks in $\mathcal{C}$. Then, there exists a finite timed automaton TA with clocks in $\mathcal{C}'$ such that TA can be proved symbolically bisimilar to X_0 and $\#\mathcal{C}' \leq 1 + \#\mathcal{C}$.*

7.4 Concluding Remarks

In the several examples along this chapter, we used ♡ both as a specification language and as an algebraic framework that allows for equational reasoning. In the first direction, we discussed encodings —shorthand notations— of traditional and easy-to-understand real time operations such as timeouts, deadlines, and urgency. We used ♡ and these newly defined operations to give two simple and compact specifications of a railroad crossing system. We used algebraic reasoning as a way to reduce each model, given as parallel composition of the different components, into a single specification which, in one of the cases, served to expose some undesirable behaviour.

In a more involved fashion, algebraic manipulation was the centre of our methodology in Section 7.3. Thus, we have used the axiomatisation of ♡ not only as a way to study real-time systems, but also as a framework to study properties of timed automata.

Part Three

Stochastic Systems

Todos los dioses mostraron mohína de ver a la Fortuna, y algunos dieron señal de asco cuando ella, con chillido desentonado, hablando a tiento, dijo [a Júpiter]:

—[. . .] ¿qué se te antojó ahora de llamarme, habiendo tantos siglos que de mí no te acuerdas? Puede ser que se te haya olvidado a ti y a esotro vulgo de diosecillos lo que yo puedo, y que así he jugado contigo y con ellos como con los hombres.

Francisco de Quevedo Villegas, *La Fortuna con seso y la hora de todos*. Zaragoza, 1650.

Chapter 8

Probabilistic Transition Systems

We define probabilistic transition systems by adding a probabilistic transition to the transition system model. A probabilistic transition is a function that takes a state and returns a probability space whose sample space ranges over the set of states. In particular we consider a model in which probabilistic and non-deterministic transitions alternate along any possible execution. As a consequence, the set of states is partitioned in two subsets: one containing the probabilistic states, to which exactly one probabilistic transition is associated per state, and the other, the non-deterministic states, which have zero or more non-deterministic outgoing transitions.

The model we define can deal with any kind of probability space, including discrete, continuous, or even singular. It is basically a generalisation of models that only deal with discrete probabilities (e.g., Vardi, 1985; Pnueli and Zuck, 1993; Larsen and Skou, 1991; van Glabbeek et al., 1995; Hansson, 1994; Segala, 1995). This generalisation allows the representation of real-time systems in which time constraints are not necessarily deterministic but depend on some random factor.

We also define a notion of bisimulation on probabilistic transition systems. Probabilistic bisimulation was introduced by Larsen and Skou (1991) in a discrete setting. It can be seen as a bisimulation relation that preserves the probabilistic choices. In our case, we extend this original definition to deal with arbitrary probability spaces.

For this and subsequent chapters some basic notions on mathematical foundations of probability theory are necessary. For a brief introduction the reader is referred to Appendix D; for further reading, he/she is referred to (Shiryaev, 1996), for instance.

8.1 The Model

Probabilistic transition systems extend transition systems with probabilistic information. In our case, we accomplish it by distinguishing two kinds of transition relations: the already present labelled or non-deterministic transition relation and a new probabilistic one. A probabilistic transition is simply a relation that moves to a new state with a certain probability. Probabilistic and non-deterministic transitions alternate in the sense

that a probabilistic step is followed by a non-deterministic one, and vice versa.

Definition 8.1. Let $Prob(H)$ denote the set of probability spaces $(\Omega, \mathcal{F}, P)$ such that $\Omega \subseteq H$. A *probabilistic transition system* is a structure $PTS = (\Sigma, \Sigma', \mathcal{L}, T, \longrightarrow)$ where

1. Σ and Σ' are two disjoint sets of *states*. A state in Σ is said to be *probabilistic*. A state in Σ' is called *non-deterministic*.
2. $\mathcal{L}$ is a set of *labels*.
3. $T : \Sigma \to Prob(\Sigma')$ is a (total) function called *probabilistic transition relation*.
4. $\longrightarrow \subseteq \Sigma' \times \mathcal{L} \times \Sigma$ is the *labelled (or non-deterministic) transition relation*.

As usual, we use the shorthand notations $\sigma' \xrightarrow{\ell} \sigma$, $\sigma' \xrightarrow{\ell}$, and $\sigma' \not\xrightarrow{\ell}$.

As for transition systems, it will be convenient to distinguish an initial (probabilistic) state $\sigma_0 \in \Sigma$. In this case we call the structure (PTS, σ_0), a *rooted probabilistic transition system*. □

Since T is defined as a (total) function, each probabilistic state has exactly one outgoing transition. We could have more generally defined T to be a relation. However, T being a function is sufficient for our purposes. In the following we give a series of examples in order to understand the expressive power of probabilistic transition systems.

Example 8.2. The tossing of a coin can be modelled by the probabilistic transition system $(\Sigma, \Sigma', \mathcal{L}, T, \longrightarrow)$ where

$$\begin{array}{ll} \Sigma = \{\sigma_0\} & T(\sigma_0) = (\Sigma', \wp(\Sigma'), P) \\ \Sigma' = \{\sigma_h, \sigma_t\} & \sigma_h \xrightarrow{head} \sigma_0 \\ \mathcal{L} = \{head, tail\} & \sigma_t \xrightarrow{tail} \sigma_0 \end{array}$$

where P is the unique probability measure defined such that $P(\{\sigma_h\}) = P(\{\sigma_t\}) = \frac{1}{2}$. This probabilistic transition system is depicted in Figure 8.1. There, probabilistic transitions are represented by dotted arrows joined by an arc.

Non-determinism can be easily modelled. For instance, consider a player who tosses a coin but cheats whenever the contender does not pay attention and in such a case he will choose arbitrarily according to his convenience. Suppose that the probability that the contender gets distracted is $\frac{1}{7}$; then, the unfair player can be modelled as in Figure 8.2.

Moreover, we can model the probability of deadlock in a straightforward manner. For instance, Figure 8.3 represents the behaviour of an unconfident robot that is walking and continuously deciding to go to the left or to the right, but may hesitate and deadlocks with equal probability. □

Since our interest is to deal with time information using probabilistic transition systems, we borrow ideas from timed transition systems and label the non-deterministic transition with timed actions. Thus, the set of labels we will consider is $\mathcal{L} = \mathcal{A} \times \mathbb{R}_{\geq 0}$, where $\mathcal{A}$ is a

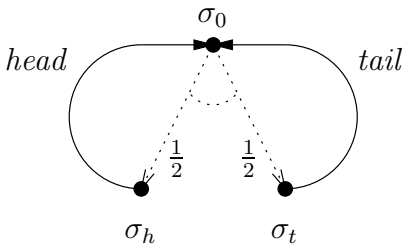


Figure 8.1: Tossing a coin

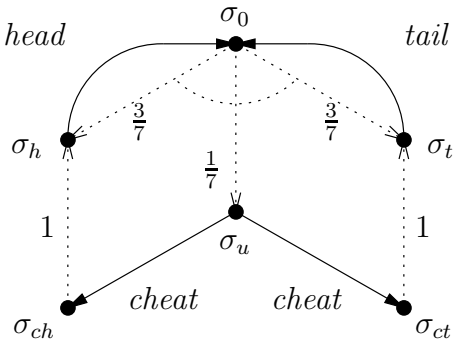


Figure 8.2: Tossing a coin and cheating

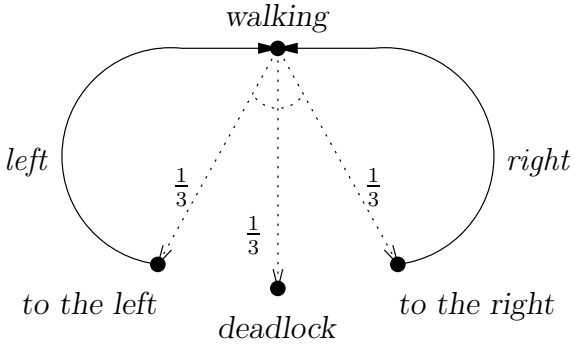


Figure 8.3: An unconfident robot

set of action names and $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers, which are intended to denote the (relative) time at which an action takes place. As before, we denote $a(d)$ whenever $(a, d) \in \mathcal{L}$ and it means "action a occurs right after the system has been idle for d time units". Note that probabilistic transitions do not take time.

Example 8.3. As an example, the following probabilistic transition system represents an alarm bell that repeatedly rings after idling randomly between 10.7 and 11.3 seconds according to a uniform distribution.

$$\begin{array}{lll} \Sigma = \{\sigma_0\} & T(\sigma_0) = \mathcal{R}(F_U) & d \xrightarrow{ring(d)} \sigma_0 \\ \Sigma' = \mathbb{R} & \mathcal{L} = \{ring\} \times \mathbb{R}_{\geq 0} & \text{with } d \in \mathbb{R}_{\geq 0} \end{array}$$

where

$$F_U(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t \leq 10.7 \\ \frac{t-10.7}{0.6} & \text{if } 10.7 < t \leq 11.3 \\ 1 & \text{if } 11.3 < t \end{cases}$$

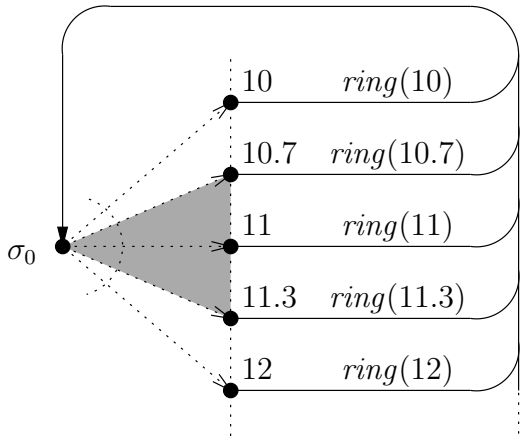


Figure 8.4: The alarm bell

and $\mathcal{R}(F_U)$ is the probability space $(\mathbb{R}, \mathcal{B}(\mathbb{R}), P)$ with $\mathcal{B}(\mathbb{R})$ being the Borel-algebra on the real line and P being the unique probability measure obtained from F_U (see Appendix D). Notice that, in this case, we have modelled a system with time information: $ring(d)$ means "the bell rings after the system has been idle for d seconds". Figure 8.4 gives a rough idea of this probabilistic transition system. The grey area indicates the states that are reached with non-zero probability. □

Example 8.4. We come back to our running example and get yet more specific about the system. Consider the switch with the time information as specified in Example 3.2. People can still arrive at any time but we can specify now that such arrival is a Poisson process occurring at a rate of one arrival each 30 minutes. Thus, the time difference between two persons turning on the light is a random variable with (negative) exponential distribution $F_e(t) = 1 - e^{-\frac{t}{30}}$. Moreover, the time at which the light turns itself off is a random variable with deterministic distribution $D_2(t) = \textbf{if } (t < 2) \textbf{ then } 0 \textbf{ else } 1$.

We can model the behaviour of the switch as a probabilistic transition system in the following way. Its ingredients are enumerated in Table 8.1, where $\mathcal{R}(F_e, D_2)$ is the probability space on the real plane with the unique probability measure obtained from F_e and D_2, and $Triv(d)$ is the trivial probability space on the sample space $\{d\}$.

We proceed to explain the system. Each time that somebody arrives and turns on the light, the system moves to the probabilistic state σ_{on}. At this point a probabilistic transition takes place and randomly determines the time at which the next arrival will occur and the time at which the light turns itself off. This is registered by a tuple (d, d') where d is the remaining time to the next arrival, and d' is the remaining time to switch off the light. Notice that $d' = 2$ with probability 1. Now two situations may occur: either the next arrival will happen soon enough, before the light turns itself off, (in which case $d \leq d'$), or the light will turn off before somebody arrives ($d' \leq d$). In the first case, the person who arrives will press the "on" button and the times will be set again. This is represented by transition $(d, d') \xrightarrow{on(d)} \sigma_{on}$. In the other case the light will turn itself off.

$\Sigma = \{\sigma_{on}\} \cup (\{\sigma_{off}\} \times \mathbb{R})$	$\Sigma' = \mathbb{R}^2 \cup \mathbb{R}$	$\mathcal{L} = \{on, off\} \times \mathbb{R}_{\geq 0}$
$T(\sigma_{on}) = \mathcal{R}(F_e, D_2)$	$(d, d') \xrightarrow{on(d)} \sigma_{on} \iff 0 \leq d \leq d'$	
$T(\sigma_{off}, d) = Triv(d)$	$(d, d') \xrightarrow{off(d')} (\sigma_{off}, d - d') \iff 0 \leq d' \leq d$	
	$d \xrightarrow{on(d)} \sigma_{on} \iff 0 \leq d$	

Table 8.1: A probabilistic transition system representing the automatic switch

This is recorded by $(d, d') \xrightarrow{off(d')} (\sigma_{off}, d - d')$; the value $d - d'$ is the remaining time until the next arrival. In state (σ_{off}, d) there is not that much to do, so the probabilistic transition is a trivial probability space with unique element d. At this point the system only waits until the next arrival in which the light is turned on: $d \xrightarrow{on(d)} \sigma_{on}$.

We might like to consider an initial state. In such a case we may think that originally the switch is off and the only thing that we have to do is to wait a random amount of time until the first arrival occurs. Thus, we extend Σ with a new state σ_{ini} and define $T(\sigma_{ini}) = \mathcal{R}(F_e)$. □

Probabilistic transition systems have been defined in many different flavours (Vardi, 1985; Pnueli and Zuck, 1993; Larsen and Skou, 1991; van Glabbeek et al., 1990, and many others). In particular, the definition of our model is inspired by works of Hans Hansson (1994), Ben Strulo (1993), and Roberto Segala (1995). We briefly discuss these works.

An alternating model is introduced in (Hansson and Jonsson, 1990; Hansson, 1991, 1994). Hansson defined an alternating model which only consider discrete probabilities (Hansson and Jonsson, 1990; Hansson, 1991, 1994). It is worth to mention that this model does not yet incorporate explicitly a probability space as a structure; instead, probabilistic transitions are arrows labeled with numbers in the interval $[0, 1]$ which adequately should sum up to 1. (Strulo, 1993) and (Harrison and Strulo, 1995) already consider both continuous and discrete probabilities in an alternating model. Like Hansson's, Strulo's model uses arrows labeled with numbers to represent probabilistic transitions. Although under discrete probabilities this is irrelevant, in the continuous case the model tends to be cumbersome and untractable. Segala defined probabilistic transition relations as explicit probability space structures (Segala, 1995; Segala and Lynch, 1995), but the model is not alternating: probabilistic and non-deterministic transitions are fused into one. Although the shape is already prepared for a straightforward extension to the general case, Segala still defines the model in the context of discrete probabilities[1].

[1]The clear reason for Roberto Segala to do so is that his work was one of the first oriented to the verification of randomized distributed real-time systems, and it is already complicated enough in the framework of discrete probabilities.

All these models include a notion of time. In all of them, timed transitions are represented loosely as in Wang Yi's model (Yi, 1990). As we showed, exactly the same model we defined above can be used to represent stochastic (timed) systems without introducing any additional machinery. The reader should understand however that our intention is to model soft real-time systems. As a consequence the concept of idling (and hence the predicate $\mathcal{U}_d$) is not interesting by itself any longer. In this context, the general assumption is that a process is allowed to idle in a given state as long as it still can execute a transition.

Example 8.5. We say that a probabilistic transition system is *discrete* if for all $\sigma \in \Sigma$, $T(\sigma)$ is a discrete probability space. We show that the subset of discrete probabilistic transition systems is as expressive as the simple probabilistic automata of (Segala, 1995; Segala and Lynch, 1995). A *(simple) probabilistic automaton* is a structure $(\Sigma, \mathcal{L}, T)$ where

1. Σ is a set of *states*,
2. $\mathcal{L}$ is a set of *labels*, and
3. $T \subseteq \Sigma \times \mathcal{L} \times Prob(\Sigma)$ is the *transition relation* such that, if $(\sigma, a, \mathcal{P}) \in T$, $\mathcal{P}$ is a discrete probability space.

Given a simple probabilistic automaton $(\Sigma, \mathcal{L}, T)$ we define the probabilistic transition system $(T, \Sigma, \mathcal{L}, T', \longrightarrow)$ where

- $\sigma \xrightarrow{a} (\sigma, a, \mathcal{P}) \stackrel{\text{def}}{\Longleftrightarrow} (\sigma, a, \mathcal{P}) \in T$, and
- $T'(\sigma, a, \mathcal{P}) \stackrel{\text{def}}{=} \mathcal{P}$, for all $(\sigma, a, \mathcal{P}) \in T$.

Notice that this translation is simply based on splitting the transition relation of the probabilistic automaton into the labelled transition followed by the probabilistic transition.

For the inverse translation, let $(\Sigma, \Sigma', \mathcal{L}, T, \longrightarrow)$ be a discrete probabilistic transition system. We translate it into the probabilistic automaton $(\Sigma', \mathcal{L}, T')$ where T' is defined by

$$(\sigma', a, \mathcal{P}) \in T' \stackrel{\text{def}}{\Longleftrightarrow} \exists \sigma \in \Sigma.\ \sigma' \xrightarrow{a} \sigma \wedge T(\sigma) = \mathcal{P}$$

In this case, the translation is based by gluing the non-deterministic transition with the probabilistic transition that succeeds it[2].

In the timed case, it can be shown that our model, when restricted to discrete probabilities, can be translated into probabilistic timed automata (Segala, 1995) but in general not in the other way around since this last model is much more expressive. □

[2]It can be proved that both translations preserve probabilistic bisimulation.

8.2 Probabilistic bisimulation

Probabilistic bisimulation was introduced by Larsen and Skou (1991) for the so-called reactive probabilistic transition systems. Such a definition was straightforwardly adapted to generative (Giacalone et al., 1990) and stratified models (van Glabbeek et al., 1990, 1995). In (Hansson, 1991), a bisimulation for discrete-timed processes with probabilities in an alternating style was introduced. Segala and Lynch (1995) extended the definition of (Larsen and Skou, 1991) to simple probabilistic automata. However, in all these works, definitions of bisimulation have been made in a discrete probabilistic setting. (Strulo, 1993; Harrison and Strulo, 1995) defined a probabilistic bisimulation in a continuous setting. In essence, our definition is the same. We also mention that a probabilistic bisimulation for the general case was introduced by de Vink and Rutten (1997) in a coalgebraic setting.

Definition 8.6. Let $(\Sigma, \Sigma', \mathcal{L}, T, \longrightarrow)$ be a probabilistic transition system. We define the function $\mu : \Sigma \times \wp(\Sigma') \to [0,1]$ by

$$\mu(\sigma, S) \stackrel{\text{def}}{=} \begin{cases} P(S \cap \Omega) & \text{if } S \cap \Omega \in \mathcal{F} \\ 0 & \text{otherwise} \end{cases}$$

provided that $T(\sigma) = (\Omega, \mathcal{F}, P)$.

Let R be an equivalence relation on $\Sigma \cup \Sigma'$ such that if $\langle \sigma_1, \sigma_2 \rangle \in R$ then either $\sigma_1, \sigma_2 \in \Sigma$ or $\sigma_1, \sigma_2 \in \Sigma'$. Let Σ'/R be the set of equivalence classes in Σ' induced by R. Then R is a *probabilistic bisimulation* if, whenever $\langle \sigma_1, \sigma_2 \rangle \in R$, for all $S \subseteq \Sigma'/R$ and $\ell \in \mathcal{L}$, the following transfer properties hold:

1. $\mu(\sigma_1, \bigcup S) = \mu(\sigma_2, \bigcup S)$, whenever $\sigma_1, \sigma_2 \in \Sigma$; and
2. $\sigma_1 \stackrel{\ell}{\longrightarrow} \sigma_1'$ implies $\sigma_2 \stackrel{\ell}{\longrightarrow} \sigma_2'$ and $\langle \sigma_1', \sigma_2' \rangle \in R$, for some $\sigma_2' \in \Sigma$, whenever $\sigma_1, \sigma_2 \in \Sigma'$.

Two states σ_1 and σ_2 are *probabilistically bisimilar*, notation $\sigma_1 \sim_p \sigma_2$, if there exists a probabilistic bisimulation R with $\langle \sigma_1, \sigma_2 \rangle \in R$.

Two rooted probabilistic transition systems (PTS_1, σ_1) and (PTS_2, σ_2) are *probabilistically bisimilar*, if their initial states σ_1 and σ_2 are probabilistically bisimilar in the (disjoint) union of PTS_1 and PTS_2. □

Although, the definition of probabilistic bisimulation coincides with the traditional definitions in the discrete case, e.g. (Larsen and Skou, 1991; Hansson and Jonsson, 1990; Segala and Lynch, 1995), we remark a necessary difference. In the discrete case, instead of property 1 above, it suffices to insist that $\mu(\sigma_1, S) = \mu(\sigma_2, S)$ where $S \in \Sigma'/R$, i.e., S is an equivalence class instead of a set of equivalence classes. In our case, this would have been too weak due to the allowance of, for instance, continuous distribution functions. This is shown in the following example.

Example 8.7. Let $PTS_i = (\{\sigma_i\}, \mathbb{R} \times \{i\}, \mathbb{R}, T_i, \longrightarrow_i)$, $i \in \{1, 2\}$, be two probabilistic transition systems, where $(d, i) \stackrel{d}{\longrightarrow}_i \sigma_i$, and $T_1(\sigma_1)$ and $T_2(\sigma_2)$ are the probability spaces

for a uniform distribution on $[0,1]$ and $[1,2]$, respectively. According to Definition 8.6, PTS_1 and PTS_2 are not bisimilar, since they do not agree in their probabilities. However, the weaker property of the discrete case would have induced that the relation

$$R = \{\langle \sigma_i, \sigma_j \rangle \mid i,j \in \{1,2\}\} \cup \{\langle (d,i),(d,j)\rangle \mid d \in \mathbb{R} \ \wedge \ i,j \in \{1,2\}\}$$

is a probabilistic bisimulation since the probability of a point in a continuous probability space is always 0. □

When proving bisimilarity, the case of $\mu(\sigma, \bigcup S) = 0$ will require special considerations on whether $(\bigcup S) \cap \Omega \in \mathcal{F}$ or not (where $(\Omega, \mathcal{F}, P) = T(\sigma)$). In the following we state that it suffices to consider only the cases where $\mu(\sigma, \bigcup S)$ is strictly greater than 0.

Proposition 8.8. *R is a probabilistic bisimulation if and only if R is an equivalence relation and whenever $\langle \sigma_1, \sigma_2 \rangle \in R$, for all $S \subseteq \Sigma'/R$ and $\ell \in \mathcal{L}$, then*

1. *$\mu(\sigma_1, \bigcup S) > 0$ implies $\mu(\sigma_1, \bigcup S) \leq \mu(\sigma_2, \bigcup S)$; and*
2. *$\sigma_1 \xrightarrow{\ell} \sigma_1'$ implies $\sigma_2 \xrightarrow{\ell} \sigma_2'$ and $\langle \sigma_1', \sigma_2' \rangle \in R$, for some $\sigma_2' \in \Sigma$.*

Proof. The "only if" is trivial. For the "if" it is sufficient to prove that the first transfer property in Definition 8.6 is implied by the definition above. First suppose $\mu(\sigma_1, \bigcup S) > 0$. Then $\mu(\sigma_1, \bigcup S) \leq \mu(\sigma_2, \bigcup S)$ and $\mu(\sigma_2, \bigcup S) > 0$. By symmetry of R, $\mu(\sigma_2, \bigcup S) \leq \mu(\sigma_1, \bigcup S)$ from which $\mu(\sigma_1, \bigcup S) = \mu(\sigma_2, \bigcup S)$. If $\mu(\sigma_1, \bigcup S) = 0$, by contradiction suppose $\mu(\sigma_1, \bigcup S) \neq \mu(\sigma_2, \bigcup S)$. Then $\mu(\sigma_2, \bigcup S) > 0$. Since R is symmetric, $\mu(\sigma_2, \bigcup S) \leq \mu(\sigma_1, \bigcup S)$ which contradicts our assumption. *Q.E.D.*

As a consequence of this result, from now on, we adopt this proposition as a definition of probabilistic bisimulation.

In the following, we state some properties of probabilistic bisimulation. Moreover, we prove that $\sim_p$ is the largest probabilistic bisimulation, and hence, that it is an equivalence relation.

Notice that the identity relation is a probabilistic bisimulation. However some classic results of non-probabilistic bisimulation do not hold in the probabilistic case. Suppose that R and R' are probabilistic bisimulations. In particular they are equivalence relations. It is not difficult to prove that $R \cup R'$ is not always an equivalence relation. As a consequence, $R \cup R'$ does not need to be a probabilistic bisimulation.

Nonetheless the transitive closure of the union of two probabilistic bisimulations is also a probabilistic bisimulation.

Theorem 8.9. *If R and R' are probabilistic bisimulations, then $(R \cup R')^*$ is also a probabilistic bisimulation.*

Proof. Let R and R' be probabilistic bisimulation. Since they are equivalence relations, $(R \cup R')^*$ is also an equivalence relation. Now, suppose that $\langle \sigma, \sigma' \rangle \in (R \cup R')^*$, then there are $\sigma_1, \ldots, \sigma_n$, $n \geq 0$, such that $\sigma R_0 \sigma_1 R_1 \ldots R_{n-1} \sigma_n R_n \sigma'$ with $R_i \in \{R, R'\}$ for all $i \leq n$.

Let $S \subseteq \Sigma'/(R \cup R')^*$. Since $R, R' \subseteq (R \cup R')^*$, there exists $S' \subseteq \Sigma'/R$ and $S'' \subseteq \Sigma'/R'$ such that $\bigcup S = \bigcup S' = \bigcup S''$. So, we have that

$$\mu(\sigma, \bigcup S) = \mu(\sigma_1, \bigcup S) = \cdots = \mu(\sigma_n, \bigcup S) = \mu(\sigma', \bigcup S)$$

which proves the first transfer property. The second transfer property follows straightforwardly. *Q.E.D.*

This last theorem is the key to prove that $\sim_p$ is the largest probabilistic bisimulation.

Corollary 8.9.1. *$\sim_p$ is the largest probabilistic bisimulation, which implies that it is an equivalence relation.*

Proof. Let $\Re$ be the set containing all probabilistic bisimulations. By definition, $\sim_p$ is the smallest set containing all relations in $\Re$. Taking into account Theorem 8.9, we can therefore calculate

$$\sim_p \quad = \quad \bigcup \Re \quad \subseteq \quad (\bigcup \Re)^* \quad \subseteq \quad \sim_p$$

As a consequence, $\sim_p = (\bigcup \Re)^*$ is a probabilistic bisimulation. *Q.E.D.*

As for bisimulation, we can define the notion of probabilistic bisimulation up to $\sim_p$ and show that finding a probabilistic bisimulation up to $\sim_p$ suffices to show probabilistic bisimilarity.

Definition 8.10. Let $(\Sigma, \Sigma', \mathcal{L}, T, \longrightarrow)$ be a probabilistic transition system. Let R be a symmetric relation on $\Sigma \cup \Sigma'$ such that if $\langle \sigma_1, \sigma_2 \rangle \in R$ then either $\sigma_1, \sigma_2 \in \Sigma$ or $\sigma_1, \sigma_2 \in \Sigma'$. R is a *probabilistic bisimulation up to* $\sim_p$ if, whenever $\langle \sigma_1, \sigma_2 \rangle \in R$, for all $S \subseteq \Sigma'/(\sim_p \cup R)^*$ and $\ell \in \mathcal{L}$, the following transfer properties hold:

1. $\mu(\sigma_1, \bigcup S) > 0$ implies $\mu(\sigma_1, \bigcup S) \leq \mu(\sigma_2, \bigcup S)$; and
2. $\sigma_1 \xrightarrow{\ell} \sigma_1'$ implies $\sigma_2 \xrightarrow{\ell} \sigma_2'$ and $\langle \sigma_1', \sigma_2' \rangle \in (\sim_p \cup R)^*$, for some $\sigma_2' \in \Sigma$. □

Theorem 8.11. *Let R be a probabilistic bisimulation up to $\sim_p$. Then $R \subseteq (\sim_p \cup R)^* \subseteq \sim_p$.*

Proof. We only need to prove that $(\sim_p \cup R)^*$ is a probabilistic bisimulation. It is not difficult to check that $(\sim_p \cup R)^*$ is an equivalence relation. So, it remains to check that the transfer properties hold. Since $(\sim_p \cup R)^* = \bigcup_{n \geq 0} (\sim_p \cup R)^n$ we prove that by induction on n.

The case for $n = 0$ is simply the identity relation, so it follows straightforwardly. For the inductive case, suppose $\langle \sigma_1, \sigma_2 \rangle \in \bigcup_{0 \leq h \leq n+1} (\sim_p \cup R)^h$. Then, there is a $\sigma \in \Sigma \cup \Sigma'$ such that $\langle \sigma_1, \sigma \rangle \in \bigcup_{0 \leq h \leq n} (\sim_p \cup R)^h$ and $\langle \sigma, \sigma_2 \rangle \in \sim_p \cup R$. Now, we have:

1. Let $S \subseteq \Sigma'/(\sim_p \cup R)^*$. Then $\mu(\sigma_1, \bigcup S) \leq \mu(\sigma, \bigcup S)$ by induction hypothesis. Besides, $\mu(\sigma, \bigcup S) \leq \mu(\sigma_2, \bigcup S)$ either because $\langle \sigma, \sigma_2 \rangle \in \sim_p$ and Definition 8.6, or because $\langle \sigma, \sigma_2 \rangle \in R$ and Definition 8.10. Therefore $\mu(\sigma_1, \bigcup S) \leq \mu(\sigma_2, \bigcup S)$.

2. Suppose $\sigma_1 \xrightarrow{\ell} \sigma_1'$. Then, $\sigma \xrightarrow{\ell} \sigma'$ and $\langle \sigma_1', \sigma' \rangle \in (\sim_p \cup R)^*$, for some $\sigma' \in \Sigma$, by induction hypothesis. Besides, $\sigma \xrightarrow{\ell} \sigma'$ implies $\sigma_2 \xrightarrow{\ell} \sigma_2'$ and either $\langle \sigma', \sigma_2' \rangle \in \sim_p$ or $\langle \sigma', \sigma_2' \rangle \in (\sim_p \cup R)^*$, for some $\sigma_2' \in \Sigma$, depending whether $\langle \sigma, \sigma_2 \rangle \in \sim_p$ or $\langle \sigma, \sigma_2 \rangle \in R$. Thus, $\sigma_2 \xrightarrow{\ell} \sigma_2'$ and $\langle \sigma_1', \sigma_2' \rangle \in (\sim_p \cup R)^*$, for some $\sigma_2' \in \Sigma$. *Q.E.D.*

8.3 Concluding Remarks

Although we cannot claim that this chapter adds new concepts, it certainly contributes original technical details. To our knowledge, only Strulo's probabilistic transition systems can deal with the generality of continuous distributions. However, his model is derived from a particular stochastic process algebra and it becomes cumbersome if it is considered in isolation or in other contexts. Less ambitious, Segala came up with a neat definition of probabilistic transition system in a discrete probabilistic setting. Having learned from both works, we could build a model (together with an equivalence relation) that works for any kind of probability space, and yet it provides a structured and elegant framework, in such a way that we can extend it with a notion of time in a very straightforward manner.

Chapter 9

Stochastic Automata

We have found in the probabilistic transition system model an adequate framework for the understanding of processes with stochastic behaviour. However, probabilistic transition systems become highly infinite as soon as we intend to model systems with stochastic time behaviour. This makes them too difficult to deal with. In this chapter, we introduce a new automaton model that allows us to represent processes with stochastic information in a symbolic way. We call it the stochastic automata model and it is inspired by the so-called *generalised semi-Markov processes (GSMPs)* (Whitt, 1980; Glynn, 1989, see also Cassandras 1993; Shedler 1993) and timed automata (Alur and Dill, 1994; Henzinger et al., 1994, see also Chapter 4). A stochastic automaton is an automaton extended with random clocks. A random clock is simply a random variable which can be set to some value according to a given probability distribution. Once set, clocks count down, and when they reach value zero, they may enable certain transitions in the automaton. We define the semantics of stochastic automata in terms of probabilistic transition systems. In fact, we define two different kinds of semantics: one when the stochastic automaton is regarded as a *closed* system, i.e., when the system is complete by itself and no external interaction is required, and the other when it is regarded as an *open* system, that is, a system that cooperates with the environment or is intended to be part of a larger system. Interpretation of stochastic automata as closed systems is adequate for the final analysis of the system, e.g. to study the performance or to verify the whole system. Instead, the interpretation as open systems is appropriate to study compositionality and to analyse how systems behave in contexts.

Compositionality is a major drawback in many existing models for performance analysis such as queuing networks, stochastic Petri nets, or GSMPs, specially, in non-Markovian models. Stochastic automata offer an appropriate framework to compose systems in a straightforward way. In fact, because of its simplicity, we use stochastic automata as the underlying semantics of a stochastic process algebra that allows for general distributions, as will be discussed in the next chapter.

Stochastic automata inherit the probabilistic bisimulation from the underlying semantics and, in doing so, it yields two different notions of equivalence: one when probabilistic bisimulation is considered with a closed system view, and the other, when it is considered

with an open system view. In addition, we define several equivalences for stochastic automata at the symbolic level. After studying the relation among all the equivalences, we show the connection between stochastic automata and GSMPs.

9.1 Clocks as Random Variables

As we did for timed systems, we will also use clocks to control and observe the passage of time on stochastic systems. But, in this context the time in which events occur is random, as we already noticed in Examples 8.3 and 8.4. For such a reason we choose clocks to be random variables by themselves. Thus, when a clock is set, it takes a random value whose probability depends on a distribution function. As time passes, clocks counts down synchronously, i.e., all do so at the same rate. When a clock reaches the value zero, we say that "the clock expires" and at this point it may enable different events. In this case we choose a different approach than with timed automata. Instead of checking whether a set of clocks satisfies a given constraint, we simply check whether all of them have expired. So, the occurrence of events is controlled by the expiration of sets of clocks. Of course, these clocks can be set at different moments along the execution of the system.

If $\mathcal{C}$ is a given set of *(random) clock variables* and $x \in \mathcal{C}$, we denote by F_x the distribution function of the clock variable x. We let $\mathbf{V}$ be the set of all valuations $v : \mathcal{C} \to \mathbb{R}$. Since clocks decrease as time evolves, we will use $v - d$ to denote the valuation which is observed d units of times after the valuation v was observed. So, if $d \in \mathbb{R}$, for all $x \in \mathcal{C}$, we define

$$(v - d)(x) \stackrel{\text{def}}{=} v(x) - d.$$

As we mentioned, clocks randomly take a value whenever they are set. Since many clocks can be set at the same instant, each one of them will probably take a different value. So, suppose $C \subseteq \mathcal{C}$ is the set of clocks that need to be set at a certain valuation v. For simplicity, we will assume that C can be ordered and we denote the ordered set by $\vec{C}$. If n is the cardinality of C, and $\vec{d} \in \mathbb{R}^n$ are the (randomly) chosen values for each clock in $\vec{C}$, then $v[\vec{C} \leftarrow\!\shortmid \vec{d}]$ denotes the new valuation after setting the clocks in C and is defined by:

$$v[\vec{C} \leftarrow\!\shortmid \vec{d}] \stackrel{\text{def}}{=} \begin{cases} \vec{d}(i) & \text{if } x = \vec{C}(i), \text{ for some } i \in \{1, \ldots, n\} \\ v(x) & \text{otherwise} \end{cases}$$

where $\vec{C}(i)$ and $\vec{d}(i)$ denote the ith element of $\vec{C}$ and $\vec{d}$, respectively.

9.2 The Model

Combining probabilities and time at a concrete semantic level results in a very complicated way to describe systems, as we experienced in Example 8.4. As for timed systems, we can resort to a symbolic way of specifying stochastic systems.

To do so we introduce the stochastic automata model. A stochastic automaton is basically a labelled transition system decorated with random clocks.

Definition 9.1. A *stochastic automaton* is a structure $SA = (\mathcal{S}, \mathcal{C}, \mathcal{A}, \rightarrow, \kappa)$ where:

- $\mathcal{S}$ is a set of *locations*.
- $\mathcal{C}$ is a set of *(random) clocks*. Each $x \in \mathcal{C}$ is a random variable with *distribution function* F_x.
- $\mathcal{A}$ is a set of *actions*.
- $\rightarrow \subseteq \mathcal{S} \times (\mathcal{A} \times \wp_{\text{fin}}(\mathcal{C})) \times \mathcal{S}$ is the set of *edges*.
- $\kappa : \mathcal{S} \to \wp_{\text{fin}}(\mathcal{C})$ is the *clock setting function*.

We denote $(s, a, C, s') \in \rightarrow$ by $s \xrightarrow{a,C} s'$ and we say that C is its *trigger set*. We also write $s \xrightarrow{a,C}$ whenever there is an s' such that $s \xrightarrow{a,C} s'$.

In many occasions it will be of convenience to distinguish an initial location $s_0 \in \mathcal{S}$. In this case we call the structure (SA, s_0) a *rooted stochastic automaton*. □

Intuitively, $s \xrightarrow{a,C} s'$ means that whenever the system is in location s, it can move to location s' by performing action a as soon as all the clocks in the trigger set C have expired. Immediately afterwards all clocks in $\kappa(s')$ are randomly set according to their probability distributions.

Example 9.2. The switch of Example 8.4 can be symbolically described by the stochastic automaton $System \stackrel{\text{def}}{=} (\mathcal{S}, \mathcal{A}, \mathcal{C}, \rightarrow, \kappa)$ where

$$\begin{array}{lll} \mathcal{S} = \{s_0, s_1, s_2\} & s_0 \xrightarrow{on,\{x\}} s_1 & \kappa(s_0) = \{x\} \\ \mathcal{A} = \{on, off\} & s_1 \xrightarrow{on,\{x\}} s_1 & \kappa(s_1) = \{x, y\} \\ \mathcal{C} = \{x, y\} & s_1 \xrightarrow{off,\{y\}} s_2 & \kappa(s_2) = \varnothing \\ & s_2 \xrightarrow{on,\{x\}} s_1 & \end{array}$$

with

$$F_x(t) = 1 - e^{-\frac{t}{30}} \qquad \text{and} \qquad F_y(t) = \begin{cases} 0 & \text{if } t < 2 \\ 1 & \text{otherwise} \end{cases}$$

Figure 9.1 depicts this stochastic automaton. Circles represent locations, variables enumerated in each location are the clocks that are to be set according to the function κ, and edges are represented by the arrows. The initial location, which has been chosen to be s_0, is represented by a small incoming arrow. □

Remark 9.3. Like timed automata, the stochastic automata model properly contains the transition system model. Given the transition system $(\Sigma, \mathcal{A}, \longrightarrow)$, we define the stochastic automaton $(\Sigma, \mathcal{A}, \varnothing, \rightarrow, \iota)$ where $\kappa(\sigma) = \varnothing$ for all $\sigma \in \Sigma$, and $\sigma \xrightarrow{a,\varnothing} \sigma'$ whenever $\sigma \xrightarrow{a} \sigma'$. Thus, a transition system is a stochastic automaton without any stochastic information. As a consequence, actions are always enabled and may occur at any time.

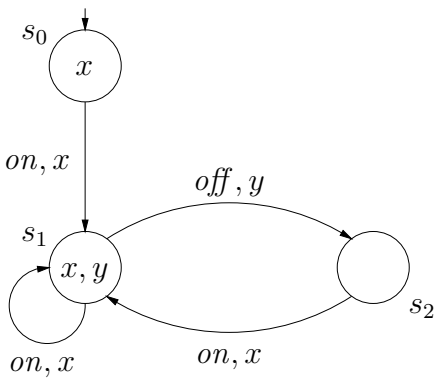


Figure 9.1: The switch

Conversely, every stochastic automaton which does not contain any activity which is stochastically dependent can be straightforwardly translated into a transition system by removing all the "additional ingredients". Other stochastic automata cannot be translated into transition systems without loosing information. □

We can also characterise whether a stochastic automaton is finite or finitely branching. The way in which we proceed is quite the same as for the other automata (see Definitions 2.7 and 4.5).

Definition 9.4. Let $SA = (\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \kappa)$ be a stochastic automaton. SA is *finitely branching* if for all $s \in \mathcal{S}$ the set

$$\{s \xrightarrow{a,C} s' \mid a \in \mathcal{A} \wedge C \in \mathcal{C} \wedge s' \in \mathcal{S}\}$$

is finite. We say that SA is *finite* if moreover it satisfies that the set of locations $\mathcal{S}$ is finite. □

9.3 Semantics of Stochastic Automata

We define the semantics of stochastic automata in terms of probabilistic transition systems. In fact, we define two different kinds of semantics: one when the stochastic automaton is regarded as a *closed* system, i.e., when the system is complete by itself and no external interaction is required, and the other when it is regarded as an *open* system, that is, a system that cooperates with the environment or is intended to be part of a larger system.

The closed system view. In the following we define the semantics of stochastic automata as *closed systems*. In this kind of system one not only models the components of the intended system but also the environment with which it interacts. In this way, an action of the whole system can take place as soon as it becomes enabled since there is no external agent that may delay its execution. That is, closed systems have the *maximal*

progress property. We refer to this interpretation as the *closed system behaviour* or *closed behaviour* for short.

Given a stochastic automaton $(\mathcal{S}, \mathcal{C}, \mathcal{A}, \rightarrow, \kappa)$, its closed behaviour is defined by a probabilistic transition system. We identify its states by the location in which the system is plus the values of the clocks at the current time. Thus the set of possible states is given by all the pairs of locations and valuations, i.e., the set $\mathcal{S} \times \mathbf{V}$. Since we have to differentiate between probabilistic and non-deterministic states, we enclose probabilistic states between parenthesis and non-deterministic states between square brackets. Thus $(\mathcal{S} \times \mathbf{V})$ is the set of probabilistic states with elements ranging over $(s, v), (s_i, v_i), \ldots$, and $[\mathcal{S} \times \mathbf{V}]$ is the set of non-deterministic states with elements ranging over $[s, v], [s_i, v_i], \ldots$.

In order to define the probabilistic transition we resort to some particular notations (which are formally defined in Appendix D). We denote by $\mathcal{R}(F_1, \ldots, F_n)$ the *unique* probability space induced by the distribution functions $F_1, \ldots, F_n$ in the n-dimensional *Borel space*.

Let $s \in \mathcal{S}$ be a location and $v \in \mathbf{V}$ be a valuation, and suppose $\#\kappa(s) = n$. We define the function $\mathcal{D}_v^s : \mathbb{R}^n \to [\{s\} \times \mathbf{V}]$ by

$$\mathcal{D}_v^s(\vec{d}) \stackrel{\text{def}}{=} \left[s, v[\overrightarrow{\kappa(s)} \hookleftarrow \vec{d}]\right]$$

for all $\vec{d} \in \mathbb{R}^n$. Notice that $\mathcal{D}_v^s$ is injective.

The function $\mathcal{D}_v^s$, which we call *decoration*, can be lifted to probability spaces on a real hyperspace in a way that the resulting probability space $\mathcal{D}_v^s(\mathcal{R}(F_1, \ldots, F_n))$ and the original $\mathcal{R}(F_1, \ldots, F_n)$ are isomorphic.

Definition 9.5. Let $SA = (\mathcal{S}, \mathcal{C}, \mathcal{A}, \rightarrow, \kappa)$ be a stochastic automaton. The *closed (system) behaviour* of SA is defined by the probabilistic transition system

$$PTS_c(SA) \stackrel{\text{def}}{=} (\ (\mathcal{S} \times \mathbf{V}), [\mathcal{S} \times \mathbf{V}], \mathcal{A} \times \mathbb{R}_{\geq 0}, T, \longrightarrow\ )$$

with T and $\longrightarrow$ defined by the following rules:

$$\textbf{Prob} \quad \frac{\overrightarrow{\kappa(s)} = (x_1, \ldots, x_n)}{T\,(s, v) \;=\; \mathcal{D}_v^s(\mathcal{R}(F_{x_1}, \ldots, F_{x_n}))}$$

$$\textbf{Closed} \quad \frac{\begin{array}{c} s \xrightarrow{a,C} s' \quad d \in \mathbb{R}_{\geq 0} \quad \forall x \in C.\ (v-d)(x) \leq 0 \\ \forall d' \in [0, d).\ \forall s \xrightarrow{b,C'} .\ \exists y \in C'.\ (v-d')(y) > 0 \end{array}}{[s, v] \xrightarrow{a(d)} (s', (v-d))}$$

We say that an edge $s \xrightarrow{a,C} s'$ is *enabled* in a valuation v if it induces a non-deterministic transition outgoing from $[s, v]$. In particular, note that $s \xrightarrow{a,\varnothing} s'$ is enabled for any valuation v.

The closed behaviour of the rooted stochastic automaton (SA, s_0) in the initial valuation $v_0 \in \mathbf{V}$ is the rooted probabilistic transition system $(PTS_c(SA), (s_0, v_0))$. □

Notice that, according to Definition 9.5, for each location s and valuation v there is exactly one probabilistic transition. So, for any stochastic automaton SA, $PTS_c(SA)$ is indeed a probabilistic transition system.

Rule **Prob** considers the setting of the clocks. Since the values of the clocks are assigned randomly, a probabilistic transition corresponds to this step. Notice that this definition relies on the definition of $\mathcal{D}_v^s$ on probability spaces. Rule **Closed** explains the case of triggering an edge. So, for the occurrence of an action a at time d according to an edge $s \xrightarrow{a,C} s'$, we check whether all the clocks in the trigger set C have already expired at time d. This part is considered by the satisfaction of

$$\forall x \in C.\ (v - d)(x) \leq 0.$$

Moreover, it should be the case that no edge was enabled before. That is, any edge must have an active (i.e. positive) clock at any valuation "previous" to $v - d$. In this way, the edge is forced to occur as soon as it becomes enabled. So, the maximal progress is checked by

$$\forall d' \in [0, d).\ \forall s \xrightarrow{b,C'} .\ \exists y \in C'.\ (v - d')(y) > 0.$$

In order to compare with timed automata, we may say that the first constraint corresponds to the guard of the edge $s \xrightarrow{a,C} s'$, and the second one corresponds to the invariant of location s.

Example 9.6. Following Definition 9.5, we can calculate that the closed behaviour of the stochastic automaton in Example 9.2 is given by

$$PTS_c(System) = (\ (\mathcal{S} \times \mathbf{V})\,, [\mathcal{S} \times \mathbf{V}]\,, \mathcal{A} \times \mathbb{R}_{\geq 0}, T, \longrightarrow\ )$$

where

$$T\,(s_0, v) = \mathcal{D}_v^{s_0}(\mathcal{R}(F_x))$$

$$T\,(s_1, v) = \mathcal{D}_v^{s_1}(\mathcal{R}(F_x, F_y))$$

$$T\,(s_2, v) = \mathcal{D}_v^{s_2}()$$

$$[s_0, (x := d, y := d')] \xrightarrow{on(d)} (s_1, (x := 0, y := d' - d)) \iff 0 \leq d$$

$$[s_1, (x := d, y := d')] \xrightarrow{on(d)} (s_1, (x := 0, y := d' - d)) \iff 0 \leq d \leq d'$$

$$[s_1, (x := d, y := d')] \xrightarrow{off(d')} (s_2, (x := d - d', y := 0)) \iff 0 \leq d' \leq d$$

$$[s_2, (x := d, y := d')] \xrightarrow{on(d)} (s_1, (x := 0, y := d' - d)) \iff 0 \leq d$$

Choosing v_0 with $v_0(x) = v_0(y) = 0$ as the initial valuation, the rooted probabilistic transition system $(PTS_c(System), (s_0, v_0))$ is the closed behaviour of the rooted stochastic automaton. □

The open system view. In this subsection, we define the behaviour of a stochastic automaton as an open system. An *open system* is a system that interacts with its environment. The environment can be a user or another system. Basically, an open system is a component of a larger system. When a stochastic automaton describes an open system, the semantics given in Definition 9.5 does not suffice: in an open system, an action that is enabled may not be executed until the environment is also ready to execute such an action. Therefore, an activity may not take place as soon as it is enabled. This kind of behaviour is appropriate to study compositionality and to analyse how the system behaves in contexts. In fact, it turns out that probabilistic bisimulation is not a congruence for some basic operations on stochastic automata, such as parallel composition. This has to do with the so-called race condition on the branches of stochastic automata. Fastest branches (i.e. branches which are enabled) may be disallowed or slowed down when the system is embedded in some context, and therefore, slower branches, which could not be executed in isolation, may become enabled in the composed stochastic automaton. For a discussion of this phenomenon, we refer to Examples 10.22 and 10.23.

In the following we define the *open system behaviour*. The open behaviour is defined as the closed behaviour in which the maximal progress condition is dropped. As a consequence, non-deterministic transitions in the open behaviour represent the fact that an edge is potentially executable at any time after it becomes enabled.

Definition 9.7. Let $SA = (\mathcal{S}, \mathcal{C}, \mathcal{A}, \rightarrow, \kappa)$ be a stochastic automaton. The *open (system) behaviour* of SA is defined by the probabilistic transition system

$$PTS_o(SA) \stackrel{\text{def}}{=} (\, (\mathcal{S} \times \mathbf{V}) \,, [\mathcal{S} \times \mathbf{V}] \,, \mathcal{A} \times \mathbb{R}_{\geq 0}, T, \longrightarrow \,)$$

where T is defined by rule **Prob** as in Definition 9.5 and $\longrightarrow$ is defined as follows.

$$\textbf{Open} \quad \frac{s \xrightarrow{a,C} s' \quad d \in \mathbb{R}_{\geq 0} \quad \forall x \in C.\ (v-d)(x) \leq 0}{[s,v] \xrightarrow{a(d)} (s', (v-d))}$$

The open behaviour of the rooted stochastic automaton (SA, s_0) in the initial valuation $v_0 \in \mathbf{V}$ is the rooted probabilistic transition system $(PTS_o(SA), (s_0, v_0))$. □

Notice that the difference between the semantics as closed and open systems relies only on the fact that the constraint of maximal progress is present in **Closed** but omitted in **Open**.

Example 9.8. We calculate now the open behaviour of the stochastic automaton in Example 9.2. According to Definition 9.7, we see that

$$PTS_o(System) = (\, (\mathcal{S} \times \mathbf{V}) \,, [\mathcal{S} \times \mathbf{V}] \,, \mathcal{A} \times \mathbb{R}_{\geq 0}, T, \longrightarrow \,)$$

where

$$T(s_0, v) = \mathcal{D}_v^{s_0}(\mathcal{R}(F_x))$$

$$T(s_1, v) = \mathcal{D}_v^{s_1}(\mathcal{R}(F_x, F_y))$$

$$T(s_2, v) = \mathcal{D}_v^{s_2}()$$

$$[s_0, (x := d, y := d')] \xrightarrow{on(d'')} (s_1, (x := d - d'', y := d' - d'')) \iff 0 \leq d'' \wedge d \leq d''$$

$$[s_1, (x := d, y := d')] \xrightarrow{on(d'')} (s_1, (x := d - d'', y := d' - d'')) \iff 0 \leq d'' \wedge d \leq d''$$

$$[s_1, (x := d, y := d')] \xrightarrow{off(d'')} (s_2, (x := d - d'', y := d' - d'')) \iff 0 \leq d'' \wedge d' \leq d''$$

$$[s_2, (x := d, y := d')] \xrightarrow{on(d'')} (s_1, (x := d - d'', y := d' - d'')) \iff 0 \leq d'' \wedge d \leq d''$$

Notice that in this case there is no correlation between the values d of x and d' of y. The only requirement is that the time d'' of occurrence of an action is positive and beyond the time in which the edge become enabled. □

9.4 Equivalences on Stochastic Automata

In the following we define several notions of equivalence for stochastic automata. In the first place, we obtain two different notions of equivalence by lifting probabilistic bisimulation to stochastic automata according to the probabilistic timed transition system we obtain from the closed or open behaviour.

Definition 9.9. Two locations s_1 and s_2 of a stochastic automaton SA are *probabilistically bisimilar on the closed behaviour*, or *closed p-bisimilar* for short, notation $s_1 \sim_c s_2$, if they are probabilistic bisimilar in $PTS_c(SA)$ when they are interpreted as probabilistic states together with any valuation, that is, if for every $v \in \mathbf{V}$, $(s_1, v) \sim_p (s_2, v)$. Similarly, two locations s_1 and s_2 are *probabilistically bisimilar on the open behaviour*, or *open p-bisimilar* for short, notation $s_1 \sim_o s_2$, if they are probabilistic bisimilar in $PTS_o(SA)$ when they are interpreted as probabilistic states together with any valuation.

Two rooted stochastic automata (SA_1, s_1) and (SA_2, s_2) are *closed p-bisimilar* if their interpretations as closed systems are probabilistically bisimilar for every initial valuation, that is, if $(PTS_c(SA_1), (s_1, v_0))$ and $(PTS_c(SA_2), (s_2, v_0))$ are probabilistically bisimilar for every valuation $v_0 \in \mathbf{V}$. Similarly we can say that (SA_1, s_1) and (SA_2, s_2) are *open p-bisimilar* if their interpretation as an open system are probabilistically bisimilar for every initial valuation. □

It is immediate that both $\sim_c$ and $\sim_o$ are equivalences on the set of locations $\mathcal{S}$.

Like we did for timed automata, we define different kinds of relations stronger than both open and closed p-bisimulations, all of them at the symbolic level of stochastic automata.

The intention of these relations is to have other means to prove open or closed p-bisimilarity, therefore we will use them whenever it is convenient for the sake of simpler and clearer proofs. Afterwards, we relate all equivalences relations we define in this section according to inclusion.

The strongest relation that we study is isomorphism.

Definition 9.10. Let $SA = (\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \kappa)$ and $SA' = (\mathcal{S}', \mathcal{A}, \mathcal{C}, \longrightarrow', \kappa')$ be two stochastic automata. We say that SA and SA' are *isomorphic*, notation $SA \cong SA'$, if there is a bijective function $\mathfrak{I} : \mathcal{S} \to \mathcal{S}'$ such that

1. $s \xrightarrow{a,C} s' \iff \mathfrak{I}(s) \xrightarrow{a,C}{}' \mathfrak{I}(s')$, and
2. $\kappa(s) = \kappa'(\mathfrak{I}(s))$.

In this case we say that the function $\mathfrak{I}$ is an *isomorphism*. Two rooted stochastic automata (SA, s_0) and (SA', s_0') are *isomorphic* if in addition $\mathfrak{I}(s_0) = s_0'$. □

The definition of isomorphism could be extended by allowing bijections on the sets of clocks and the set of actions. For our purposes, the simpler Definition 4.9 is sufficient. A slightly weaker notion is structural bisimulation. A structural bisimulation is a bisimulation at the level of stochastic automata that preserves both actions and sets of clocks.

Definition 9.11. Let $(\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \kappa)$ be a stochastic automaton. A relation $R \subseteq \mathcal{S} \times \mathcal{S}$ is a *structural bisimulation* if R is symmetric and for all $a \in \mathcal{A}$ and $C \in \mathcal{C}$, whenever $\langle s_1, s_2 \rangle \in R$ the following transfer properties hold

1. $s_1 \xrightarrow{a,C} s_1'$, then there is a $s_2' \in \mathcal{S}$ such that $s_2 \xrightarrow{a,C} s_2'$ and $\langle s_1', s_2' \rangle \in R$; and
2. $\kappa(s_1) = \kappa(s_2)$.

We say that two locations s_1 and s_2 are *structurally bisimilar*, and denote $s_1 \sim_s s_2$, if there exists a structural bisimulation R such that $\langle s_1, s_2 \rangle \in R$.

Two rooted stochastic automata (SA_1, s_1) and (SA_2, s_2) are *structurally bisimilar*, if their initial locations s_1 and s_2 are structurally bisimilar in the (disjoint) union of SA_1 and SA_2. □

Theorem 9.12.

1. $\sim_s$ is a structural bisimulation relation, and

2. $\sim_s$ is an equivalence relation on the set $\mathcal{S}$ of locations.

The proof of this last theorem follows exactly along the same lines as the proof for standard bisimulation. Moreover, repeating the same technique as in other cases, we can define the notion of structural bisimulation up to $\sim_s$.

Definition 9.13. Let $(\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \kappa)$ be a stochastic automaton. A relation $R \subseteq \mathcal{S} \times \mathcal{S}$ is a *structural bisimulation up to* $\sim_s$ if R is symmetric and for all $a \in \mathcal{A}$ and $C \in \mathcal{C}$, whenever $\langle s_1, s_2 \rangle \in R$ the following transfer properties hold

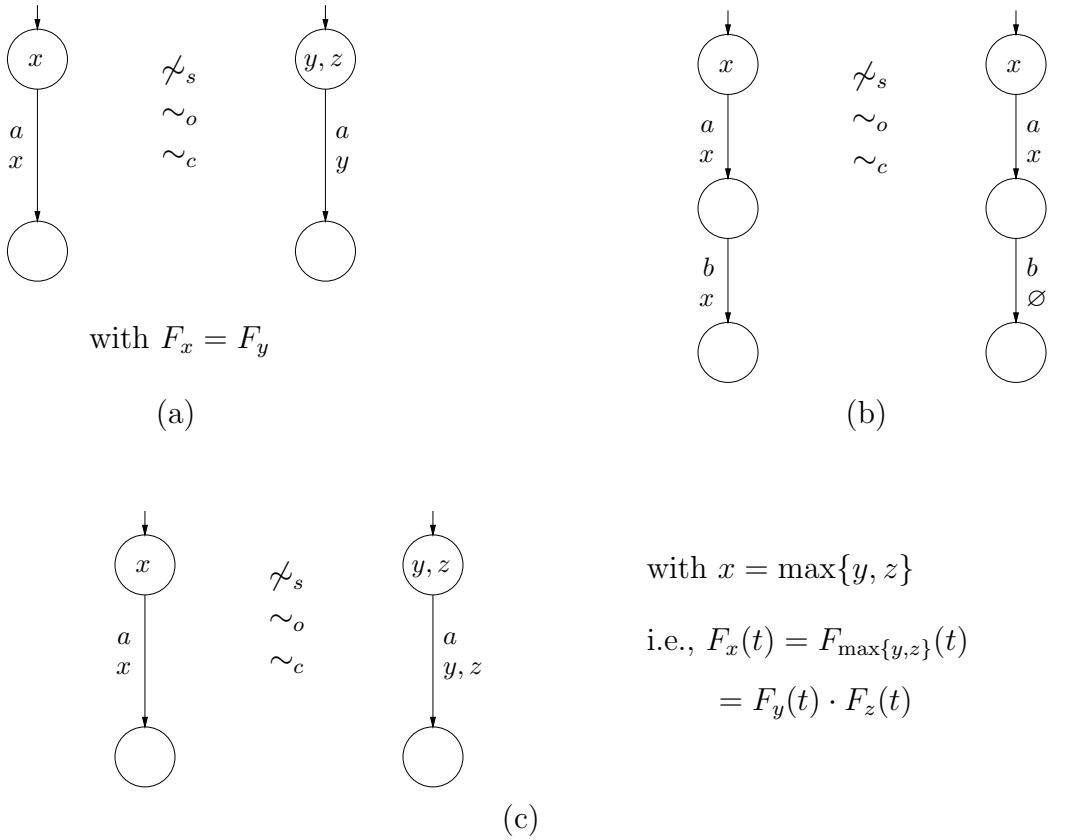


Figure 9.2: Examples of non structurally bisimilar stochastic automata

1. $s_1 \xrightarrow{a,C} s_1'$, then there is a $s_2' \in \mathcal{S}$ such that $s_2 \xrightarrow{a,C} s_2'$ and $\langle s_1', s_2' \rangle \in (\sim_s \cup R)^*$; and
2. $\kappa(s_1) = \kappa(s_2)$. □

As expected, finding a structured bisimulation up to $\sim_s$ is enough to prove structured bisimilarity, the proof follows the lines of that for Theorem 3.6.

Theorem 9.14. *Let R be a structural bisimulation up to $\sim_s$. Then $R \subseteq (\sim_s \cup R)^* \subseteq \sim_s$.*

Structural bisimulation does not consider stochastic information. Simple things like changing the name of a random variable while preserving the same probability function, or setting clocks that will not be used, do not lead to any behavioural difference. An example is shown in Figure 9.2(a). Similarly, the use of clocks that have already been used —and thus already expired— is irrelevant (Figure 9.2(b)). Moreover, a unique clock may replace a set of clocks if they are always used and initialised together (Figure 9.2(c)). Notice that all the examples in Figure 9.2 give pairs of open, and also closed, p-bisimilar stochastic automata.

In the following we define a symbolic bisimulation which captures all these cases. Therefore, this symbolic bisimulation considers stochastic information already at a symbolic level, although, we will see, it does not completely capture open p-bisimilarity.

As symbolic bisimulation needs to relate clocks, we characterise those clocks which are important to take into account. Basically, a clock is relevant if it is used somewhere, that is, it is in some triggering set on some edge. But we are only interested whether a clock is relevant in a particular location s; so we must also known whether the clock is either free or set in s.

Definition 9.15. Let $(\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \kappa)$ be a stochastic automaton. The set of *free variables* of a location $s \in \mathcal{S}$, notation $FV(s)$, is defined by the smallest set satisfying

$$FV(s) = \left\{ x \mid s \xrightarrow{a,C} s' \wedge x \in C \cup FV(s') \right\} - \kappa(s)$$

The set of *relevant clock variables* of a location $s \in \mathcal{S}$, notation $rel(s)$, is defined by

$$rel(s) \stackrel{\text{def}}{=} \left\{ x \mid s \xrightarrow{a,C} s' \wedge x \in C \cup FV(s') \right\}$$

□

Notice that $FV(s) \subseteq rel(s) \subseteq FV(s) \cup \kappa(s)$ (concretely, $FV(s) = rel(s) - \kappa(s)$) but not necessarily $\kappa(s) \subseteq rel(s)$. In fact, notice that clock z is not relevant in the right automaton of Figure 9.2(a).

Like for the timed case (see Section 4.4), a symbolic bisimulation relation contains triplets $\langle s_1, s_2, SR \rangle$, where s_1 and s_2 are locations, and SR is a component explaining how the clocks relate. The relation SR, which we call *synchronisation relation*, contains pairs of sets of clocks $\langle C_1, C_2 \rangle$. Each C_i contains clocks that are relevant in s_i, that is, $C_i \subseteq rel(s_i)$. The idea is that the set of clocks C_1 is "synchronised" to the set C_2 both stochastically and in time. By "stochastically synchronised", we mean that both set should represent the same stochastic value, in the sense that the random variables $\max(C_1)$ and $\max(C_2)$ are the same. This is precisely intended for cases like in Figure 9.2(c); there, it is expected that $\langle \{x\}, \{y, z\} \rangle$ is an element in SR in the relation of the initial states. By "synchronised in time", we mean that they should be set and used at the same moment. Therefore, either $C_i \subseteq \kappa(s_i)$ for both $i = 1, 2$, or $C_i \cap \kappa(s_i) = \varnothing$ for both $i = 1, 2$. Moreover, when both locations engage in a simulation step, say they are $s_i \xrightarrow{a,C_i^\star} s_i'$, then either $C_i \subseteq C_i^\star$ for both $i = 1, 2$, or $C_i \cap C_i^\star = \varnothing$. It is also important that locations s_1' and s_2' are related in a new triplet $\langle s_1', s_2', SR' \rangle$. The new synchronising relation SR' has to preserve the old relation imposed by SR for the clocks that are still relevant. We call this requirement *forward compatibility*.

We define the symbolic bisimulation as follows.

Definition 9.16. Let $(\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \kappa)$ be a stochastic automaton. A *symbolic bisimulation* is a relation $R \subseteq \mathcal{S} \times \mathcal{S} \times \wp(\wp(\mathcal{C}) \times \wp(\mathcal{C}))$ that, whenever $\langle s_1, s_2, SR \rangle \in R$, the following properties hold.

1. $\langle C_1, C_2 \rangle, \langle C_1', C_2' \rangle \in SR$ and $(C_1 \cap C_1') \cup (C_2 \cap C_2') \neq \varnothing$ then $\langle C_1, C_2 \rangle = \langle C_1', C_2' \rangle$.
2. For $i = 1, 2$, $\bigcup \{ C_i \mid \langle C_1, C_2 \rangle \in SR \} = rel(s_i)$.

3. If $\langle C_1, C_2\rangle \in SR$ and $(C_1 \cap \kappa(s_1)) \cup (C_2 \cap \kappa(s_2)) \neq \varnothing$, then
   (a) $C_1 \subseteq \kappa(s_1)$ and $C_2 \subseteq \kappa(s_2)$; and
   (b) for all $t \in \mathbb{R}$, $\prod_{x \in C_1} F_x(t) = \prod_{y \in C_2} F_y(t)$.
4. $s_1 \xrightarrow{a, C_1^\star} s_1'$, then there are s_2' and $C_2^\star$ such that
   (a) $s_2 \xrightarrow{a, C_2^\star} s_2'$;
   (b) $\langle C_1, C_2\rangle \in SR$ and $(C_1 \cap C_1^\star) \cup (C_2 \cap C_2^\star) \neq \varnothing$ implies $C_1 \subseteq C_1^\star$ and $C_2 \subseteq C_2^\star$; and
   (c) there exists SR' such that $\langle s_1', s_2', SR'\rangle \in R$ and it is *forward compatible* with SR, that is,

$$\{\langle C_1, C_2\rangle \in SR \mid (C_1 \cap FV(s_1')) \cup (C_2 \cap FV(s_2')) \neq \varnothing\} - \big(\wp(C_1^\star) \times \wp(C_2^\star)\big) = \{\langle C_1, C_2\rangle \in SR' \mid (C_1 \cap FV(s_1')) \cup (C_2 \cap FV(s_2')) \neq \varnothing\} - \big(\wp(C_1^\star) \times \wp(C_2^\star)\big).$$

5. $s_2 \xrightarrow{a, C_2^\star} s_2'$, then there are s_1' and $C_1^\star$ such that
   (a) $s_1 \xrightarrow{a, C_1^\star} s_1'$;
   (b) $\langle C_1, C_2\rangle \in SR$ and $(C_1 \cap C_1^\star) \cup (C_2 \cap C_2^\star) \neq \varnothing$ implies $C_1 \subseteq C_1^\star$ and $C_2 \subseteq C_2^\star$; and
   (c) there exists SR' such that $\langle s_1', s_2', SR'\rangle \in R$ and it is *forward compatible* with SR, that is,

$$\{\langle C_1, C_2\rangle \in SR \mid (C_1 \cap FV(s_1')) \cup (C_2 \cap FV(s_2')) \neq \varnothing\} - \big(\wp(C_1^\star) \times \wp(C_2^\star)\big) = \{\langle C_1, C_2\rangle \in SR' \mid (C_1 \cap FV(s_1')) \cup (C_2 \cap FV(s_2')) \neq \varnothing\} - \big(\wp(C_1^\star) \times \wp(C_2^\star)\big).$$

We say that two locations s_1 and s_2 are *symbolically bisimilar*, and denote $s_1 \sim_{\&} s_2$, if there exists a symbolic bisimulation R such that $\langle s_1, s_2, SR\rangle \in R$ for some SR satisfying

(a) if $\langle \{x\} \cup C_1, C_2\rangle \in SR$ and $x \in FV(s_1)$ then $C_1 \subseteq \{x\}$;

(b) if $\langle C_1, \{x\} \cup C_2\rangle \in SR$ and $x \in FV(s_2)$ then $C_2 \subseteq \{x\}$; and

(c) if $\langle \{x\}, \{y\}\rangle \in SR$, $x \in FV(s_1)$, and $y \in FV(s_2)$ then $x = y$.

Two rooted stochastic automata (SA_1, s_1) and (SA_2, s_2) are *symbolically bisimilar*, if their initial locations s_1 and s_2 are symbolically bisimilar in the (disjoint) union of SA_1 and SA_2. □

Condition 1 states that clocks can be grouped only in one pair of SR. If we do not do so, we could wrongly equate stochastic automata like in Figure 9.3. Notice that in the left hand side automaton, actions a and b always become enabled at the same time, while this

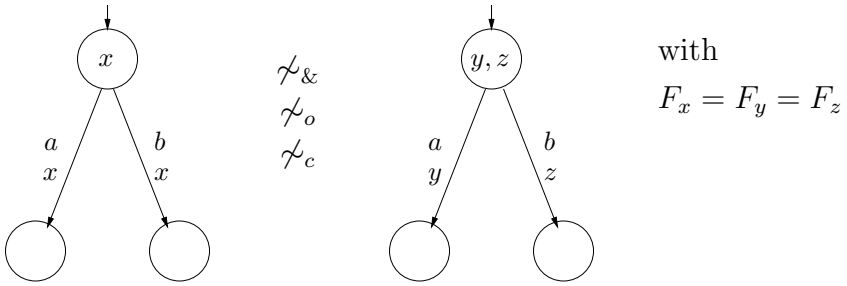


Figure 9.3: Two stochastic automata not to be equated

is not generally the case in the other automaton. In this case, it depends on the probability that y and z take the same value (and remember that they are independent!). Condition 2 requires that all clocks in SR are relevant, and conversely, that all relevant clocks are in the synchronisation relation. Condition 3a states that clocks related in the same pair must be set at the same time. This is part of the "synchronisation in time" imposed by SR. Condition 3b is the "stochastic synchronisation" requirement in which random variables $\max(C_1)$ and $\max(C_2)$ should be the same. Recall that the distribution function of $\max(C)$ is defined by

$$F_{\max(C)}(t) = \prod_{x \in C} F_x(t)$$

for all $t \in \mathbb{R}$.

Items 4 and 5 are the transfer properties, and they state how the edges must be simulated. So, if the left-hand side location has an outgoing arrow, the right one has also an outgoing arrow labelled with the same action name (condition 4a). Moreover the triggering sets should contain complete sets of clocks from the same relating pairs (condition 4b); this is the rest of the "synchronisation in time" imposed by SR. Finally, the target locations must be again related with a new SR', and SR' should be forward compatible with SR on those clocks which are still useful (condition 4c). Notice that clocks in the previous triggering sets are not required to keep synchronised since they already expired and, as a consequence, they are not anymore a restriction on the enabling condition. This is meant to relate stochastic automata like the one in Figure 9.2(b).

Like for timed automata, two locations may be related by a symbolic bisimulation but they are not necessarily symbolically bisimilar. This has to do with the fact that a triplet $\langle s_1, s_2, SR \rangle$ "remembers" in SR how the clocks were related in the past (refer to the forward compatibility property in 4c). For two locations to be symbolically bisimilar, their free clock variables cannot be arbitrarily synchronised. Following the criterion of closed and open p-bisimulation in which two locations are related if they are related in every valuation (see Definition 9.9), we require that SR satisfies the following additional conditions. First, we require that free variables are not related to any other variable at the same side. This is stated by items (a) and (b). We also require that if two free variables of different sides are related they should have the same name (condition (c)).

Thus defined, $\sim_{\&}$ is an equivalence relation.

Theorem 9.17. *$\sim_{\&}$ is an equivalence relation on the set $\mathcal{S}$ of locations.*

Proof. The fact the $\sim_{\&}$ is reflexive follows from Theorem 9.19 below, which states that $\sim_s \subseteq \sim_{\&}$. Symmetry is also simple to prove: if R is a symbolic bisimulation, it is not difficult to check that $\{\langle s_2, s_1, SR^{-1}\rangle \mid \langle s_1, s_2, SR\rangle \in R\}$ is also a symbolic bisimulation. Moreover, notice that this relation satisfies conditions (a), (b), and (c) in Definition 9.16 whenever the given relation R does. Transitivity is more involved and proved in Appendix E.1. *Q.E.D.*

The rest of this section is devoted to relate all the equivalences on stochastic automata previously defined. The first relation is simple: given two automata that are isomorphic, any pair of locations related by the isomorphism is also structurally bisimilar.

Theorem 9.18. *Let $\mathfrak{I}$ be an isomorphism from SA to SA′. Then the locations s and $\mathfrak{I}(s)$ are structurally bisimilar in the (disjoint) union of SA and SA′.*

Proof. The fact that the relation $R \stackrel{\text{def}}{=} \{\langle s, \mathfrak{I}(s)\rangle \mid s \in \mathcal{S}\}$ is a structural bisimulation should be clear. *Q.E.D.*

The following theorem states that if two locations are structurally bisimilar, they are also symbolically bisimilar.

Theorem 9.19. *Let s_1 and s_2 be two locations in a timed automata TA. If $s_1 \sim_s s_2$ then $s_1 \sim_{\&} s_2$.*

Proof. Let R_s be a structural bisimulation. Notice that that for any $\langle s_1, s_2\rangle \in R_s$, the set of free clock variables coincide, and as a consequence, the set of relevant clocks coincide as well, i.e., $rel(s_1) = rel(s_2)$. We define the relation

$$R_{\&} \stackrel{\text{def}}{=} \left\{\langle s_1, s_2, SR\rangle \mid \langle s_1, s_2\rangle \in R_s \wedge SR = \{\langle\{x\},\{x\}\rangle \mid x \in rel(s_1) = rel(s_2)\}\right\}$$

It is straightforward to check that $R_{\&}$ is a symbolic bisimulation. Moreover, for any tuple $\langle s_1, s_2, SR\rangle \in R_{\&}$, SR trivially satisfy conditions (a)–(c) in Definition 9.16. Thus, the theorem follows. *Q.E.D.*

Moreover, two locations that are symbolically bisimilar, are also open p-bisimilar. The proof of the following theorem can be found in Appendix E.2.

Theorem 9.20. *Let s_1 and s_2 be two locations in a stochastic automaton SA. If $s_1 \sim_{\&} s_2$ then $s_1 \sim_o s_2$.*

Finally, we have that open p-bisimulation is finer than closed p-bisimulation. That is, if two locations are probabilistic bisimilar in the open behaviour they are also probabilistic bisimilar in the closed behaviour. The proof of the next theorem can be found in Appendix E.3.

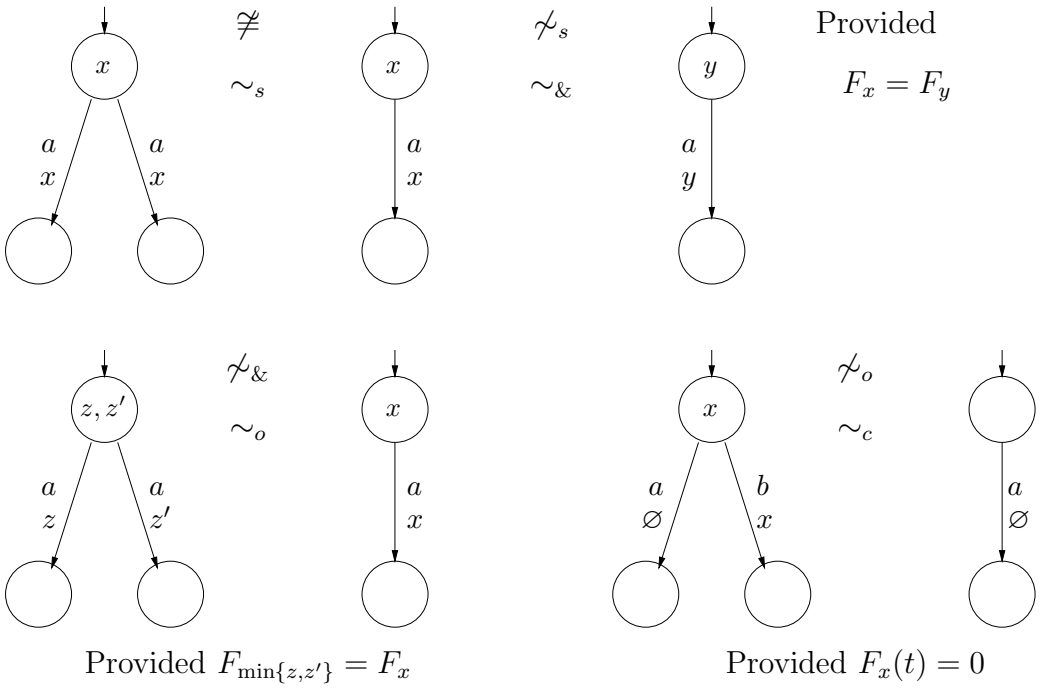


Figure 9.4: Example of different equivalences in stochastic automata

Theorem 9.21. *Let s_1 and s_2 be two locations in a stochastic automaton SA. If $s_1 \sim_o s_2$ then $s_1 \sim_c s_2$.*

Summarising the relation among the equivalences introduced before, we have that

1. two isomorphic (rooted) stochastic automata are also structurally bisimilar,
2. two structurally bisimilar stochastic automata are also symbolically bisimilar,
3. two symbolically bisimilar stochastic automata are also open p-bisimilar,
4. two open p-bisimilar stochastic automata are also closed p-bisimilar, and
5. in none of the previous cases the converse holds in general, which is shown in Figure 9.4.

Remark 9.22. Like for timed automata, all the previous bisimulations collapse in the subset of stochastic automata with no stochastic information (i.e. no clocks).

In fact, it is easy to show that structural bisimulation coincides with bisimulation on transition systems when stochastic automata are restricted to those representable by transition systems as explained by Remark 9.3. Moreover, it is not difficult to show that closed p-bisimulation also coincides with bisimulation under this condition, and consequently, also symbolic bisimulation and open p-bisimulation. □

9.5 Stochastic Automata and GSMPs

Generalised semi-Markov processes (GSMP for short) (Whitt, 1980) provide models for the behaviour of a wide class of discrete-event systems (see also Glynn, 1989; Cassandras, 1993; Shedler, 1993). It generalises the well known continuous-time Markov chains by allowing, on the one hand, that timing of events is distributed arbitrarily —not only according to memoryless distribution functions— and, on the other, that such a timing depends not only on the current state but also on the past states. The GSMP model has been shown to be an effective tool to study performance and dependability of complex and non-trivial systems (see, for instance, Nicola et al., 1993).

In the following, we first define the GSMP model and then we show how it relates to the stochastic automata model.

Generalised Semi-Markov Processes. A GSMP is defined on top of an automaton sometimes referred as generalised semi-Markov scheme[1].

Definition 9.23. A *generalised semi-Markov scheme* (GSMS for short) is a structure $G = (\mathcal{Z}, \mathcal{E}, \mathsf{active}, \mathsf{next}, F)$ where

- $\mathcal{Z}$ is the set of *(output) states*;
- $\mathcal{E}$ is a set of *events*;
- $\mathsf{active} : \mathcal{Z} \to \wp(\mathcal{E})$ assigns a set of *active* events to each output state; and
- $\mathsf{next} : \mathcal{Z} \times \mathcal{E} \to \mathcal{Z}$ assigns the *next state* according to the current state and the event that is triggered; and
- $F : \mathcal{E} \to (\mathbb{R} \to [0, 1])$ assigns to each event a *continuous distribution function* such that $F(e)(0) = 0$; we write F_e instead of $F(e)$.

As initial condition a state $z_0 \in \mathcal{Z}$ is appointed. A *generalised semi-Markov process* (GSMP for short) is the stochastic process defined by a GSMS □

We have restricted our attention to a subclass of GSMSs. In fact, the only significant restriction is that the next state function is deterministic. That is, the next state is uniquely determined by the present state and the triggered event. In general, this function is probabilistic, i.e., given the current state and an active event, function next returns some state in $\mathcal{Z}$ with certain probability. Two other minor restrictions are considered. First, the assignment of a distribution function to an event may depend on the past history of the execution of the GSMS instead of a single state. This is not a real restriction since we can introduce as many events as necessary to represent the more general GSMP. (Actually, each history of the general GSMP may become an event in the reduced GSMP.) Second, sometimes the timers that control the occurrence of events are allowed to have different rates, though this is not a usual choice.

[1] In many occasions the term GSMP is overloaded.

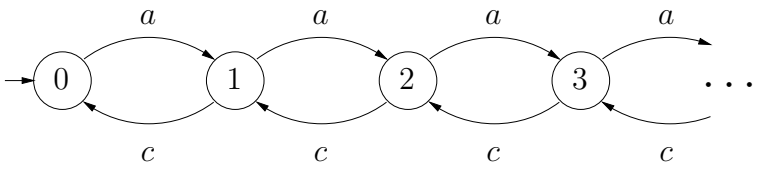


Figure 9.5: The GSMS of a $G/G/1/\infty$ queue

Example 9.24. Consider a simple queueing system in which jobs arrive and wait until they are executed by a single server. Assume that the queue has infinite capacity. Jobs arrive with an interarrival time that is determined by some distribution F_a, and the execution time of a job by the server is determined by a distribution function F_c. Suppose that both distribution functions are continuous. This system is known as a $G/G/1/\infty$ queue, where the G's stand for general distribution of the arrival and service process respectively, 1 indicates that there is only one server, and ∞ says that the queue is unbounded (it has infinite capacity). (The curious reader is referred to, e.g., Jain, 1991; Harrison and Patel, 1992; Cassandras, 1993, for more details.)

A typical GSMP description of such queuing system is given by the GSMS whose components are defined as follows.

- The set of output states is defined by the non-negative integers $\mathcal{Z} = \mathbb{N}$, where $i \in \mathcal{Z}$ indicates the amount of people in the system. Initially, the system does not contain any job. We therefore select 0 as the initial state.

- The set of events contains the event a that represents the arrival of a jobs, and c that represents the completion of a job. Thus, $\mathcal{E} = \{a, c\}$.

- In the initial state 0 there is no job in the system that can be completed, so, only an arrival is possible. In the others either a new job arrives or a job is completed . Therefore, $\mathsf{active}(0) = \{a\}$ and $\mathsf{active}(i) = \{a, c\}$ for $i > 0$.

- At any state i, a new job may arrive increasing as a consequence the number of jobs in the system to $i + 1$. At any state $i > 0$ a job can be completed decreasing by one the number of jobs. So, $\mathsf{next}(i, a) = i + 1$ and $\mathsf{next}(i + 1, c) = i$.

- The distribution function associated to the events are those enumerated above: F_a and F_c.

This GSMS is depicted in Figure 9.5 where a state z is represented by a circle, the set $\mathsf{active}(z)$ is given by the labels of its outgoing arrows, and arrows represent the next state function where $z \xrightarrow{e} z'$ if $\mathsf{next}(z, e) = z'$. □

A GSMS behaves as follows. Suppose that the system is in a certain state z. The active events in $\mathsf{active}(z)$ will have associated some non-negative number. Such a number is the time remaining to execute the event. All other events (the inactive ones) have no

particular value associated. The active event with the smallest associated value is chosen to be triggered. Say $e^\star$ is that event, and $d^\star$ its value. Notice that such an event is unique with probability 1, since in GSMS all the events are actually continuous random variables. Thus, the probability that two active events have the same value is zero. The next state is given by $\mathsf{next}(z, e^\star)$. The set of new events $N(z, e^\star)$ is given by

$$N(z, e^\star) \stackrel{\text{def}}{=} \mathsf{active}(\mathsf{next}(z, e^\star)) - (\mathsf{active}(z) - \{e^\star\}),$$

i.e., the events active in the new state which were not active before. The values of these new events are randomly defined according to the respective distribution function given by F. The set of old events is the set of all active events that are still active in the new state:

$$O(z, e^\star) \stackrel{\text{def}}{=} \mathsf{active}(z) \cap (\mathsf{active}(\mathsf{next}(z, e^\star)) - \{e^\star\})$$

The value of every old event is decreased by $d^\star$ units of time. In this new state with the new valuation, the process is repeated.

Example 9.25. In our queuing system example, we can calculate the following sets of old and new events.

$$\begin{array}{lll@{\qquad}lll} N(0,a) &=& \{a,c\} & O(0,a) &=& \varnothing \\ N(i,a) &=& \{a\} \quad \text{if } i>0 & O(i,a) &=& \{c\} \quad \text{if } i>0 \\ N(1,c) &=& \varnothing & O(1,c) &=& \{a\} \\ N(i,c) &=& \{c\} \quad \text{if } i>2 & O(i,c) &=& \{a\} \quad \text{if } i>2 \end{array}$$

□

We describe the behaviour of a GSMS in terms of probabilistic transition systems[2].

Let z be a state in $\mathcal{Z}$. Let $e^\star \in \mathsf{active}(z)$ be an active event in z. Let $\mathbf{V} : \mathcal{E} \to \mathbb{R} \cup \{\perp\}$ be a set of valuations, where $\perp$ represents the undefined value. Let $v \in \mathbf{V}$ and suppose $n = \#N(z, e^\star)$. We define the decoration $\mathcal{D}_v^{z,e^\star} : \mathbb{R}^n \to [\mathcal{Z} \times \mathbf{V}]$ by

$$\mathcal{D}_v^{z,e^\star}(\vec{D}) \stackrel{\text{def}}{=} \left[\mathsf{next}(z, e^\star), v[\overrightarrow{N(z, e^\star)} \mapsto \vec{D}][\mathcal{E} - \mathsf{active}(\mathsf{next}(z, e^\star)) \mapsto \perp]\right]$$

Notice that $\mathcal{D}_v^{z,e^\star}$ is injective.

Definition 9.26. Let $G = (\mathcal{Z}, \mathcal{E}, \mathsf{active}, \mathsf{next}, F)$ be a GSMS. The *behaviour* of G is the probabilistic transition system $PTS(G) \stackrel{\text{def}}{=} (\Sigma, \Sigma', \mathcal{E} \times \mathbb{R}_{\geq 0}, T, \longrightarrow)$ where $\Sigma \subseteq (\mathcal{E} \times (\mathcal{Z} \times \mathbf{V}))$ and $\Sigma' \subseteq [\mathcal{Z} \times \mathbf{V}]$ are defined as follows:

$$\Sigma \stackrel{\text{def}}{=} \{(e^\star, (z, v)) \mid v(e^\star) = 0 \wedge (e \in \mathsf{active}(z) \iff v(e) \neq \perp)\}$$

$$\Sigma' \stackrel{\text{def}}{=} \{[z, v] \mid e \in \mathsf{active}(z) \iff v(e) \neq \perp\}$$

and T and $\longrightarrow$ are defined by the following rules

[2]In the literature the behaviour of a GSMS is defined by giving the distribution function of each transition which directly corresponds to the probability transition of the probabilistic transition system defined here. For more information the reader is referred to (Glynn, 1989; Shedler, 1993).

$$\frac{\overrightarrow{N(z,e^\star)} = (e_1,\ldots,e_n)}{T\,(e^\star,(z,v)) \;=\; \mathcal{D}_v^{z,e^\star}(\mathcal{R}(F_{e_1},\ldots,F_{e_n}))}$$

$$\frac{e^\star \in \mathsf{active}(z) \quad d \in \mathbb{R}_{\geq 0} \quad (v-d)(e^\star) = 0 \quad \forall e \in \mathsf{active}(z).\ (v-d)(e) \geq 0}{[z,v] \xrightarrow{e^\star(d)} (e^\star,(z,v-d))}$$

For the special case of the initial state we extend $PTS(G)$ with a new distinguished probabilistic state (z_0) and define

$$\frac{\overrightarrow{\mathsf{active}(z_0)} = (e_1,\ldots,e_n)}{T\,(z_0) \;=\; \mathcal{D}_{id}^{z_0,e^\star}(\mathcal{R}(F_{e_1},\ldots,F_{e_n}))}$$

In this case, the initial valuation turn out to be irrelevant. Now, the complete behaviour of G is defined by the rooted probabilistic transition system $(PTS(G),(z_0))$. □

Notice that T is a function. As a consequence $PTS(G)$ is a well-defined probabilistic transition system.

Relating GSMPs to Stochastic Automata. We show the relation between stochastic automata and GSMP by presenting a mapping from GSMS into stochastic automata. We show that the translation preserves the semantics by giving a probabilistic bisimulation on the closed behaviour of the translated GSMS. As a consequence we can conclude that the GSMP model is properly included in the stochastic automata model.

Definition 9.27. Let $G = (\mathcal{Z},\mathcal{E},\mathsf{active},\mathsf{next},F)$ be a GSMS. The translation of G into a stochastic automaton, is defined by the rooted stochastic automata $(SA(G),\langle z_0,\varnothing\rangle)$ where $SA(G) \stackrel{\text{def}}{=} (\mathcal{Z}\times\wp(\mathcal{E}),\mathcal{E},\mathcal{E},\rightarrow,\kappa)$ with $\rightarrow$ defined by

$$\frac{e \in \mathsf{active}(z)}{\langle z,C\rangle \xrightarrow{e,\{e\}} \langle \mathsf{next}(z,e),\mathsf{active}(z)-\{e\}\rangle}$$

and $\kappa(\langle z,C\rangle) \stackrel{\text{def}}{=} \mathsf{active}(z) - C$. □

In the pair $\langle z,C\rangle$, C carries the information of which events were already active. Notice that there are much too many locations $\langle z,C\rangle$. In fact, the only "useful" (reachable) locations have the shape $\langle \mathsf{next}(z,e),\mathsf{active}(z)-\{e\}\rangle$ for some $z\in\mathcal{Z}$ and $e\in\mathsf{active}(z)$. This can be observed in the source of the edge defined in the rule of Definition 9.27. The only exception is the initial location. Notice, moreover, that $\kappa(\langle \mathsf{next}(z,e),\mathsf{active}(z)-\{e\}\rangle = \mathsf{active}(\mathsf{next}(z,e)) - (\mathsf{active}(z)-\{e\}) = N(z,e)$, which is the set of new events in the state $\mathsf{next}(z,e)$, and, in the initial case, $\kappa(\langle z_0,\varnothing\rangle) = \mathsf{active}(z_0)$. Besides, for each active event in the output state z, there is an output edge from any location $\langle z,C\rangle$. In fact, $\mathsf{active}(z) = \bigcup\{e \mid (z,C) \xrightarrow{e,\{e\}}\}$.

Example 9.28. The translation of the queue of Example 9.24 is the stochastic automaton depicted in Figure 9.6 □

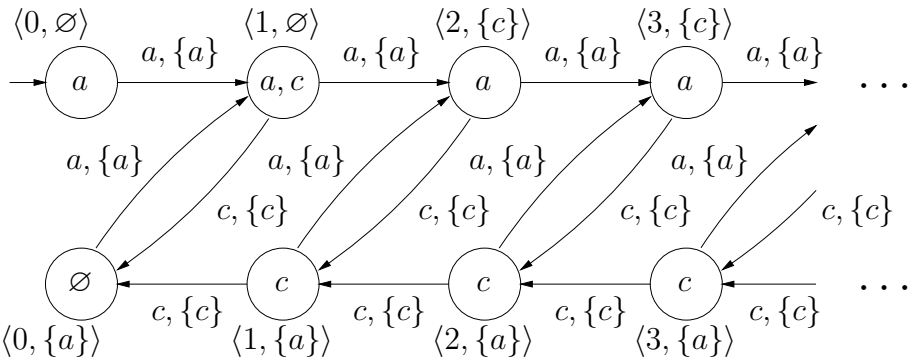


Figure 9.6: The translation of the GSMS of Figure 9.5

In the following theorem, we prove the correctness of the translation by showing that the behaviour of a GSMS G is probabilistic bisimilar with the closed behaviour of its translation $SA(G)$.

Theorem 9.29. *Let G be a GSMS. Then $(PTS(G), (z_0))$ and $(PTS_c(SA(G)), (\langle z_0, \varnothing\rangle, v))$ are probabilistic bisimilar for any valuation valuation v.*

Proof. Define the relation

$$
\begin{aligned}
R \stackrel{\text{def}}{=} \Big(& \left\{ \langle (e^\star, (z, v)), (\langle \mathsf{next}(z, e^\star), \mathsf{active}(z) - \{e^\star\}\rangle, v')\rangle \mid \forall e \in \mathsf{active}(z).\ v(e) = v'(e) \right\} \\
& \cup \left\{ \langle (z_0), (\langle z_0, \varnothing\rangle, v)\rangle \mid v \in \mathbf{V} \right\} \\
& \cup \left\{ \langle [z, v], [\langle z, C\rangle, v']\rangle \mid \forall e \in \mathsf{active}(z).\ v(e) = v'(e) \right\} \Big)^{\mathsf{s}}.
\end{aligned}
$$

The proof that R is a probabilistic bisimulation up to $\sim_p$ is routine. *Q.E.D.*

It is clear that, in general, a translation is not possible in the other way around since the stochastic automata model not only allows a more general class of distribution functions, but also non-determinism is inherent in the model.

9.6 Concluding Remarks

We have defined a new general model for the description of stochastic soft real-time systems: the stochastic automata model. We gave two definitions of its behaviour that are intended to describe two different points of view: the open system view and the closed system view. Several equivalence relations for stochastic automata were introduced in order to give a mathematical support to the model. We finally showed that stochastic automata contain a useful and quite general stochastic model, namely, the GSMP model.

Although we have introduced a diversity of technical matters involving stochastic automata, the important contribution of this new model has not yet been discussed. Compositionality is a key issue that allows us to build complex systems from smaller components

in a hierarchical manner. Stochastic automata were in particular conceived to be composed, specifically as components of a distributed system whose interaction results in a new stochastic automaton. This issue is discussed in depth in the remaining chapters.

Chapter 10

♤ – Stochastic Process Algebra for Discrete Event Systems

We already mentioned that compositionality is a major drawback in existing models for performance analysis and that stochastic automata offer an appropriate framework for composition. In this chapter, we introduce a process algebra to model and analyse stochastic systems. We call it ♤ (read *spades*) which is the acronym of *stochastic process algebra for discrete event systems*. Its underlying semantics is given in terms of stochastic automata. The methodology to define and study ♤ closely follows the methodology we used to develop the theory of ♡.

Usually, the semantics of stochastic process algebras such as TIPP (Götz et al., 1993; Hermanns and Rettelbach, 1994), PEPA (Hillston, 1996), and EMPA (Bernardo and Gorrieri, 1998; Bernardo, 1999) are defined in terms of extended transition systems, which basically have each transition labelled not only with the action name, but also with a distribution function that determines the (stochastic) timing of such a transition. However, the usual interleaving character of transition systems demands a careful treatment of the definition of parallel composition. We recall that the expansion law plays an important role in process algebras of interleaving nature such as ♧ or ♡: it says how parallel composition can be decomposed in terms of more primitive operations as, for instance, prefixing and non-deterministic choice. Stochastic process algebras extend prefixing into $a_F; P$ where F is a distribution function which determines the probability of the random delay after which the action a can happen. In this setting, the interleaving expansion law does no longer hold in general. To address this problem, the community has come up with different solutions.

A first proposal, and the most widely accepted, has been to restrict the attention to exponential distributions. Their memoryless property restores the expansion law (e.g., Hillston, 1996; Buchholz, 1994; Hermanns and Rettelbach, 1994). Others have faced the general case (Götz et al., 1993; Harrison and Strulo, 1995; Priami, 1996) but the underlying semantic model usually becomes infinite and cumbersome, which makes it practically intractable. An alternative solution is to drop the expansion law by moving to true concurrency models (Brinksma et al., 1995), but for simple recursive processes the semantic representations are infinite. Recently, however, Bravetti et al. (1998) presented a calculus

whose semantics is based on the so called ST-semantics[1]. This calculus preserves finiteness of semantic objects in a reasonable way. However, although this has not been studied yet, it can clearly be seen that this calculus would not allow either an expansion law without enriching the set of operations.

We propose a more elegant solution for ♤. We separate the stochastic information from the action name. (We remark that a similar approach has been used by Harrison and Strulo (1995) in a general setting, and Hermanns and Rettelbach (1996); Hermanns (1998) in a Markovian setting.) Instead of writing $a_F; P$, we write $\{\!|x|\!\}\,(\{x\} \mapsto a; P)$. The operator $\{\!|x|\!\}\ldots$ sets the clock x according to the distribution function F, and the operation $\{x\} \mapsto \ldots$ prevents the prefixing $a; P$ to happen until clock x has expired (i.e., reached value 0). This separation of concerns gives as a result a straightforward expansion law, and moreover, it introduces more expressive power.

Let us see a concrete example. Take processes $p_1 \equiv \{\!|x|\!\}\,(\{x\} \mapsto a; \mathbf{0})$ and $p_2 \equiv \{\!|y|\!\}\,(\{y\} \mapsto b; \mathbf{0})$. The stochastic automata that they represent are depicted in Figure 10.1(a).

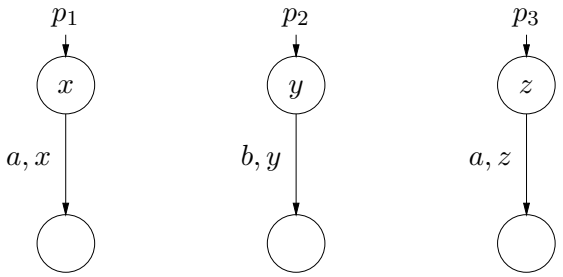
In the parallel composition $p_1 \,||_{\varnothing}\, p_2$, clocks x and y are set according to their respective distribution functions. Suppose that in this particular experiment the outcome is $x = 2.3$ and $y = 3.5$. After 2.3 time units the value of clock x decreases to 0 and action a becomes enabled. At this moment a can be executed. Notice that y still has a remaining value of 1.2 time units. This is the amount of time that the process needs to wait to be able to execute action b. If the outcome were such that $y < x$, then the order in which actions a and b are executed would be inverted. The stochastic automaton resulting from this parallel composition is depicted in Figure 10.1(b).

Let $p_3 \equiv \{\!|z|\!\}\,(\{z\} \mapsto a; \mathbf{0})$ (see Figure 10.1(a)) and suppose we want that processes p_1 and p_3 synchronise in action a, that is, we consider $p_1 \,||_a\, p_3$. Initially, both clocks x and z take random values according to their distributions. Since action a has to be performed by both p_1 and p_3 at the same time, it is necessary that both processes are ready to execute it. As a consequence, both x and z have to expire in order to enable a. This behaviour is represented by the stochastic automaton of Figure 10.1(c).

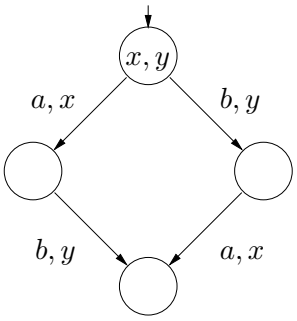
For the record, we observe that in principle *any* kind of (continuous, discrete, ...) distribution function is allowed in this model, while we maintain a finite semantic object in a reasonable way (comparable to regular processes in ♧ or ♡).

This chapter is organised as follows. After defining the syntax of ♤, we discuss its operational semantics which is given in terms of stochastic automata. We also show that every stochastic automata can be coded in ♤, stating as a consequence that both ♤ and the stochastic automata model are equally expressive. Afterwards, we discuss congruences for the different equivalence relations studied in the previous chapter. We conclude with a summary of the results reported in this and the previous chapter.

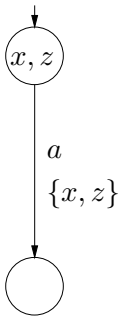
[1]ST-semantics is a specialisation of interleaving semantics that allows for refinement of actions (see van Glabbeek and Vaandrager, 1987). Usually, it is considered a true concurrency model, although it does not include the concept of causality.



(a) The stochastic automata of p_1, p_2, and p_3



(b) The parallel composition $p_1 \,||_{\varnothing}\, p_2$



(c) The synchronisation $p_1 \,||_{a}\, p_3$

Figure 10.1: Parallel composition and synchronisation

10.1 Syntax

♤ uses random clocks just like stochastic automata does. A term in ♤ is constructed using any operation of the basic process algebra ♧ plus two new operations. The first new operation is the *triggering condition*. Given a set of random clocks C, operation $C \mapsto p$ behaves like process p but for its execution it must wait until all clocks in C have expired, that is, they all should count down to 0. The other operation is the *clock setting*. Process $\{\!|C|\!\}\, p$ sets the clocks in C randomly according to their respective distribution function. We will usually write $\{\!|x_1, \ldots, x_n|\!\}\, p$ instead of $\{\!|\{x_1, \ldots, x_n\}|\!\}\, p$.

Definition 10.1. Let $\mathcal{A}$ be a set of *actions*. Let $\mathcal{C}$ be a set of random clocks. The syntax of ♤ is defined according to the following grammar:

$$\begin{aligned} p \quad ::= \quad & \mathbf{0} \;\mid\; a;p \;\mid\; C \mapsto p \;\mid\; \{\!|C'|\!\}\, p \;\mid\; p + p \\ \mid \quad & p \,||_A\, p \;\mid\; p \,\llcorner\!\!\lrcorner_A\, p \;\mid\; p \,|_A\, p \;\mid\; p[f] \;\mid\; X \end{aligned}$$

where $a \in \mathcal{A}$ is an *action name*, $C \in \wp_{\text{fin}}(\mathcal{C})$ is a *trigger set* of random clocks, $C' \in \wp_{\text{fin}}(\mathcal{C})$ is a *clock setting* set, $A \subseteq \mathcal{A}$ is a *synchronisation set*, $f : \mathcal{A} \to \mathcal{A}$ is a *renaming function*, and X is a *process variable* belonging to the set $\mathcal{V}$ of process variables.

Process variables are defined by *recursive equations* of the form $X = p$ where p is a ♤ term, and a set E of recursive equations defines a *recursive specification*. We occasionally consider a distinguished process variable in the recursive specification E called *root*. □

The notion of *guarded* and *regular* specifications are defined for ♤ exactly in the same way as for ♧. The reader is referred to Definition 2.14. In the context of ♤ we will in general consider guarded recursion. Unguarded recursion may easily yield processes with no defined semantics as we will see in Sections 10.2 and 10.3.

We will assume that the new operations $C \mapsto \_$ and $\{\!|C|\!\}\,\_$ have the highest binding precedence together with prefixing and renaming. Notice that no confusion arises if we use these operations (except renaming) without parenthesis. For instance, $\{\!|x|\!\}\,\{x\} \mapsto a; X$ is the process $\{\!|x|\!\}\,(\,\{x\} \mapsto (a; X)\,)$.

Example 10.2. We modify the ♧ specification of the switch given in Example 2.9. Using ♤, we can describe the stochastic behaviour of the different activities just like we did in Examples 8.4 and 9.2.

The arrival of people is a Poisson process with inter-arrival rate of 30 minutes. Thus, it is controlled by a clock x with exponential distribution function $F_x(t) = 1 - e^{-\frac{t}{30}}$.

$$\mathit{Arrival} \;=\; \{\!|x|\!\}\,\{x\} \mapsto \mathit{on}; \mathit{Arrival}$$

The switch can be turned on at any moment, however it turns itself off after exactly two minutes of idling. Thus, we consider a clock y with deterministic distribution function $F_y(t) = \textbf{if } t < 2 \textbf{ then } 0 \textbf{ else } 1$

$$\begin{aligned} \mathit{SwitchOff} &= \mathit{on}; \mathit{SwitchOn} \\ \mathit{SwitchOn} &= \mathit{on}; \mathit{SwitchOn} + \{\!|y|\!\}\,\{y\} \mapsto \mathit{off}; \mathit{SwitchOff} \end{aligned}$$

Notice that the timing of the action *on* is governed externally. It only depends on the context in which *Switch* is placed and thus it does not have associated stochastic information. This is reasonable since people in different hotels may arrive at different rate, or even according different distributions.

The complete system is described by the composition of the *Switch* and the *Arrival* process which interact on action *on*:

$$\mathit{System} \;=\; \mathit{Arrival} \,||_{\mathit{on}}\, \mathit{SwitchOff}$$

We introduce the following shorthand notation for the sake of clarity. We define

$$a(x); p \stackrel{\text{def}}{=} \{\!|x|\!\}\,\{x\} \mapsto a; p.$$

Similar notation is adopted as operation in the majority of the existing stochastic process algebras. Instead of the random clock x, we can find a distribution function as in (Götz et al., 1993; Brinksma et al., 1995; Priami, 1996; Bravetti et al., 1998) or a rate in most of the Markovian process algebras (Hermanns and Rettelbach, 1994; Hillston, 1996; Bernardo and Gorrieri, 1998, etc.).

$fv(\mathbf{0}) = \varnothing$		$fv(p \,\|\|_A\, q) = fv(p) \cup fv(q)$	
$fv(a;p) = fv(p)$		$fv(p \,\lfloor\!\lfloor_A\, q) = fv(p) \cup fv(q)$	
$fv(C \mapsto p) = C \cup fv(p)$		$fv(p \,\lfloor_A\, q) = fv(p) \cup fv(q)$	
$fv(p+q) = fv(p) \cup fv(q)$		$fv(p[f]) = fv(p)$	
$fv(\{\!\{C\}\!\}\, p) = fv(p) - C$		$fv(X) = fv(p)$	$(X = p)$

Table 10.1: Free and bound variables in ♤

Thus, we can rewrite the complete recursive specification as follows

$$\begin{aligned} \mathit{Arrival} &= \mathit{on}(x); \mathit{Arrival} \\ \mathit{SwitchOff} &= \mathit{on}; \mathit{SwitchOn} \\ \mathit{SwitchOn} &= \mathit{on}; \mathit{SwitchOn} + \mathit{off}(y); \mathit{SwitchOff} \\ \mathit{System} &= \mathit{Arrival} \,||_{on}\, \mathit{SwitchOff} \end{aligned}$$

□

Like in ♡, the clock setting operation $\{\!\{C\}\!\}\, p$ binds every free occurrence of clocks belonging to C in p. Intuitively a clock x is free in p if p has a subterm $(C \cup \{x\}) \mapsto q$ which does not appear in a context $\{\!\{\ldots x \ldots\}\!\}\ \ldots$. A clock x is bound in p if p has a subterm $\{\!\{\ldots x \ldots\}\!\}\, q$. We only define formally the set $fv(p)$ of free variables of the process p.

Definition 10.3. Let $p \in$ ♤. The set $fv(p)$ of *free (clock) variables* of p is defined as the smallest set satisfying the equations in Table 10.1. □

Thus, for instance, if $p \equiv \{\!\{x\}\!\}\, \{x,y\} \mapsto a; \mathbf{0} + \{x\} \mapsto b; \mathbf{0}$ then $fv(p) = \{x,y\}$; notice, however, that x appears also bound.

10.2 Semantics in Terms of Stochastic Automata

The semantics of ♤ is defined in terms of stochastic automata by using structured operational semantics as we have already done for the previously defined process algebras. Like for ♡, the definition of the rules for parallel composition needs special attention and operation $\overline{\mathsf{ck}}$ comes again into play. Recall that $\overline{\mathsf{ck}}(p)$ is a process that behaves like p except that no clock is set at the very beginning. Operation $\overline{\mathsf{ck}}(p)$ is defined recursively on the syntax of p. This definition is given in Table 10.2. Notice that $\overline{\mathsf{ck}}(X)$ defines a new variable and, besides, it holds in general that $\overline{\mathsf{ck}}(\overline{\mathsf{ck}}(p)) = \overline{\mathsf{ck}}(p)$.

$\overline{\mathsf{ck}}(\mathbf{0}) = \mathbf{0}$	$\overline{\mathsf{ck}}(a;p) = a;p$	$\overline{\mathsf{ck}}(\{\!\{C\}\!\}\, p) = \overline{\mathsf{ck}}(p)$
$\overline{\mathsf{ck}}(\mathsf{op}(p)) = \mathsf{op}(\overline{\mathsf{ck}}(p))$		$(\mathsf{op} \in \{C \mapsto \_, \_[f]\})$
$\overline{\mathsf{ck}}(p \oplus q) = \overline{\mathsf{ck}}(p) \oplus \overline{\mathsf{ck}}(q)$		$(\oplus \in \{+, \|\|_A, \lfloor\!\lfloor_A, \lfloor_A\})$
$\overline{\mathsf{ck}}(X) = X_{\overline{\mathsf{ck}}}$	(provided $X = p$ and $X_{\overline{\mathsf{ck}}} = \overline{\mathsf{ck}}(p)$)	

Table 10.2: Auxiliary clock removal operation

The different parts that form a stochastic automaton which is associated to a ♤ term are given in the following. The function κ and the relation $\longrightarrow$ are defined as the least relations satisfying rules in Table 10.3. Notice that

$$\kappa(\overline{\mathsf{ck}}(p)) = \varnothing \qquad \frac{p \xrightarrow{a,C} p'}{\overline{\mathsf{ck}}(p) \xrightarrow{a,C} p'} \tag{10.1}$$

and $fv(p) \subseteq fv(\overline{\mathsf{ck}}(p)) \subseteq fv(p) \cup \kappa(p)$.

The rules capture the intuitive behaviour explained in the following. As usual, process $\mathbf{0}$ represents a process that cannot perform any action but is allowed to idle indefinitely. In the action prefixing $a;p$, action a is immediately enabled and once it is performed the behaviour of p is exhibited. $C \mapsto p$ can perform any activity p can, but with the restriction that it must wait until all the clocks in C have expired. $p + q$ behaves either as p or q, but not both. We remark that the passage of time does not resolve the choice if the process is regarded as an open system; if instead it is regarded as a closed system, the fastest process is the one to be executed. This last case is known as the *race condition*. For $\{\!\{C\}\!\}\, p$, clocks in C are set together with the clocks to be set by p according to their respective distribution function. Afterwards, it proceeds exactly as p. The parallel composition $p \,||_A\, q$ executes p and q in parallel, and they are synchronised by actions in A. We should remark that synchronisation may happen if both processes are ready to do it, thus the requirement on the guard that all the clocks in both the trigger sets of p and q must expire. Non-synchronisation actions proceed as in the original process. It just has to be remembered that the other process, the one that remains idle, should not set again the clocks, which is controlled by the operation $\overline{\mathsf{ck}}$. The omission of $\overline{\mathsf{ck}}$ yields to an undesirable situation similar to the case in ♡ (see Section 5.2).

♤ also suffers of capture of variables. In the following we discuss several examples in which capture occurs.

The first situation we describe is quite similar to one of the cases in ♡. In process

$$p_1 \equiv \{\!\{x\}\!\}\, a; \{x\} \mapsto \{\!\{x\}\!\}\, \{x\} \mapsto b; \mathbf{0} \tag{10.2}$$

$$
\begin{array}{lll@{\qquad}lll}
\kappa(\mathbf{0}) & = & \varnothing & \kappa(p \,||_A\, q) & = & \kappa(p) \cup \kappa(q) \\
\kappa(a;p) & = & \varnothing & \kappa(p \,\llcorner\!\!\lrcorner_A\, q) & = & \kappa(p) \cup \kappa(q) \\
\kappa(C \mapsto p) & = & \kappa(p) & \kappa(p \,|_A\, q) & = & \kappa(p) \cup \kappa(q) \\
\kappa(\{\!|C|\!\}\, p) & = & C \cup \kappa(p) & \kappa(p[f]) & = & \kappa(p) \\
\kappa(p+q) & = & \kappa(p) \cup \kappa(q) & \kappa(X) & = & \kappa(p) \qquad (X = p)
\end{array}
$$

$$
a; p \xrightarrow{a,\varnothing} p \qquad \frac{p \xrightarrow{a,C'} p'}{C \mapsto p \xrightarrow{a,C \cup C'} p'} \qquad \frac{p \xrightarrow{a,C'} p'}{\{\!|C|\!\}\, p \xrightarrow{a,C'} p'}
$$

$$
\frac{p \xrightarrow{a,C} p'}{p + q \xrightarrow{a,C} p'} \qquad \frac{p \xrightarrow{a,C} p'}{q + p \xrightarrow{a,C} p'}
$$

$$
\frac{p \xrightarrow{a,C} p'}{p \,||_A\, q \xrightarrow{a,C} p' \,||_A\, \overline{\mathsf{ck}}(q)}\; a \notin A \qquad \frac{p \xrightarrow{a,C} p'}{q \,||_A\, p \xrightarrow{a,C} \overline{\mathsf{ck}}(q) \,||_A\, p'}\; a \notin A
$$

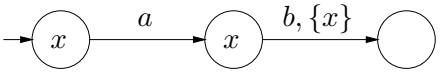
$$
\frac{p \xrightarrow{a,C} p' \quad q \xrightarrow{a,C'} q'}{p \,||_A\, q \xrightarrow{a,C \cup C'} p' \,||_A\, q'}\; a \in A
$$

$$
\frac{p \xrightarrow{a,C} p'}{p \,\llcorner\!\!\lrcorner_A\, q \xrightarrow{a,C} p' \,||_A\, \overline{\mathsf{ck}}(q)}\; a \notin A \qquad \frac{p \xrightarrow{a,C} p' \quad q \xrightarrow{a,C'} q'}{p \,|_A\, q \xrightarrow{a,C \cup C'} p' \,||_A\, q'}\; a \in A
$$

$$
\frac{p \xrightarrow{a,C} p'}{p[f] \xrightarrow{f(a),C} p'[f]} \qquad \frac{p \xrightarrow{a,C} p'}{X \xrightarrow{a,C} p'}\; X = p
$$

Table 10.3: SOS rules for ♤

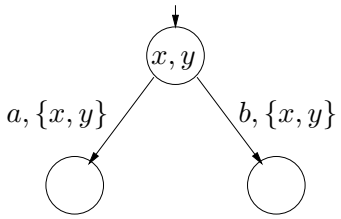
the second occurrence of x is intended to be bound to the outermost clock setting as shown by the grey arrow. However, rules in Table 10.3 induce the following stochastic automaton:



in which x is captured by the innermost clock setting yielding an unwanted situation. This is marked by the black arrow in equation (10.2). Like in ♡ we consider that clocks are different if they are set in different places, even when they have the same name. Following this idea, capture also occurs in summation. A free clock in one of the summands may be wrongly bound to a clock setting in the other summand as it is shown in the following process:

$$\begin{array}{l} fv \\ p_2 \equiv \{\!|x,y|\!\}\,\{x,y\} \mapsto a;\mathbf{0} + \{\!|y|\!\}\,\{x,y\} \mapsto b;\mathbf{0} \end{array} \tag{10.3}$$

A naive interpretation of process p_2 would derive in the following automaton:



which corresponds to process $p_3 \equiv \{\!|x,y|\!\}\,(\{x,y\} \mapsto a;\mathbf{0} + \{x,y\} \mapsto b;\mathbf{0})$. Notice that, in addition, another unexpected phenomenon occurs. Clock y on the left-hand side summand of p_2 has been "unified" with the clock y on the right-hand side. The similar situation in ♡ is harmless since both clocks are set to the same value, but in ♤ clocks are independent random variables that can take different values, even when they have the same associated distribution. This undesirable situation can be illustrated as follows.

$$p_2 \equiv \{\!|x,y|\!\}\,\{x,y\} \mapsto a;\mathbf{0} + \{\!|y|\!\}\,\{x,y\} \mapsto b;\mathbf{0}$$

While in process p_2, actions a and b will become usually enabled at different times —and this would be in general with probability 1 if x and y are continuously distributed—, in process p_3 they will be always enabled at the same time and bound to non-determinism even under the race condition.

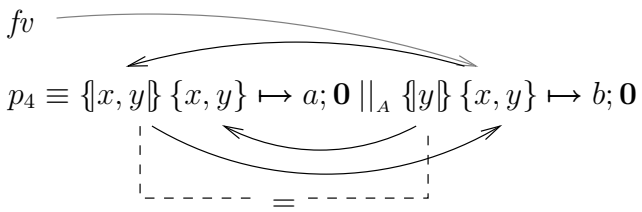
$$\frac{C \cap \kappa(p) \neq \varnothing}{\mathsf{cv}(C \mapsto p)} \qquad \frac{\mathsf{cv}(p) \quad X = p}{\mathsf{cv}(X)}$$

$$\frac{\mathsf{cv}(p) \quad \mathsf{op} \in \{C \mapsto \_, \{\!|C|\!\}\,\_, \_[f]\}}{\mathsf{cv}(\mathsf{op}(p))} \qquad \frac{\kappa(p) \cap fv(q) \neq \varnothing \quad \oplus \in \{+, ||_A, \between_A, |_A\}}{\mathsf{cv}(p \oplus q) \quad \mathsf{cv}(q \oplus p)}$$

$$\frac{\mathsf{cv}(p) \quad \oplus \in \{+, ||_A, \between_A, |_A\}}{\mathsf{cv}(p \oplus q) \quad \mathsf{cv}(q \oplus p)} \qquad \frac{\kappa(p) \cap \kappa(q) \neq \varnothing \quad \oplus \in \{+, ||_A, \between_A, |_A\}}{\mathsf{cv}(p \oplus q) \quad \mathsf{cv}(q \oplus p)}$$

Table 10.4: Conflict of clock variables in ♡

In a similar way, an attempt to define the associated stochastic automaton of process



would induce a capture of the free variable x and a unification of the two independent clocks y.

We give a characterisation of the processes that suffer from conflict of variables. We do it in two steps. First we define those processes that have a conflict "now"; using this notion we characterise afterwards processes which, although not necessarily having conflict at the root, they may eventually reach a process with local conflict.

Definition 10.4. A process $p \in$ ♤ has *local conflict of (clock) variables* if $\mathsf{cv}(p)$ can be proven using the rules in Table 10.4. Otherwise, we say that p is *locally free of conflict* and denote it by $\neg\mathsf{cv}(p)$. □

Definition 10.4 characterises the processes which have conflict at the very beginning. That is why we call it "local" conflict. To see whether a process is conflicting in general we have to observe each possible derivation and check whether a local conflict is encountered.

Definition 10.5. We say that $p \in$ ♤ is *conflict-free* (or *does not have conflict of variables*) if for every sequence

$$p \equiv p_0 \xrightarrow{a_1,C_1} p_1 \xrightarrow{a_2,C_2} p_2 \cdots p_{n-1} \xrightarrow{a_n,C_n} p_n,$$

$\neg\mathsf{cv}(p_i)$ for all $i \in \{0, \ldots, n\}$. Otherwise, we say that p *has conflict of variables*. We say that a recursive specification is *conflict-free* (or *does not have conflict of variables*) if every process variable it defines is also conflict-free. □

The reader is invited to check that p_2 and p_4 have local conflict, and, although p_1 does not, it still has conflict in the general sense since a process with local conflict is reached after a one-step derivation.

We should consider an additional technical restriction. Notice that we have defined the stochastic automata such that, for every location s, $\kappa(s)$ is finite. Similarly, we required that if $s \xrightarrow{a,C} s'$ then C should also be finite. Some unguarded (infinite) recursive specifications may instead induce infinite sets. To give an example, consider $\{X_n = X_{n+1} + \{\!|x_n|\!\}\,\{x_n\} \mapsto a_n; \mathbf{0} \mid n \in \mathbb{N}\}$.

Definition 10.6. We say that the behaviour of $p \in ⏚$ is *locally definable*, denoted by $\mathsf{def}(p)$, if p is locally free of conflict, i.e. $\neg\mathsf{cv}(p)$, and $\kappa(p)$ is finite. If in addition, p is conflict-free and for every sequence

$$p \equiv p_0 \xrightarrow{a_1,C_1} p_1 \xrightarrow{a_2,C_2} p_2 \cdots p_{n-1} \xrightarrow{a_n,C_n} p_n,$$

$\mathsf{def}(p_i)$ for all $i \in \{0,\ldots,n\}$, then the behaviour of p is *definable*. □

We can now define the semantics of ⏚ in terms of stochastic automata.

Definition 10.7. Let $⏚^{\mathsf{cf}} \stackrel{\text{def}}{=} \{p \in ⏚ \mid \text{the behaviour of } p \text{ is definable}\}$. The semantics of ⏚ is defined according to the stochastic automaton $SA(⏚^{\mathsf{cf}}) = (⏚^{\mathsf{cf}}, \mathcal{A}, \mathcal{C}, \longrightarrow, \kappa)$ where $\mathcal{A}$ and $\mathcal{C}$ are as in Definition 10.1 and $\longrightarrow$ and κ are the least relations satisfying the rules in Table 10.3. The behaviour of each process $p \in ⏚$ is defined only if it is definable by the rooted timed automaton $(SA(⏚^{\mathsf{cf}}), p)$. □

Example 10.8. The reader is invited to check that the stochastic automaton defined by the process *System* in Example 10.2 is the one depicted in Figure 9.1. □

10.3 Stochastic Automata up to α-conversion

Although process $p_2 \equiv \{\!|x,y|\!\}\,(\{x,y\} \mapsto a;\mathbf{0} + \{\!|y|\!\}\,\{x,y\} \mapsto b;\mathbf{0})$ does not have a defined semantics according to the previous discussion, the intuitive meaning is quite clear. Provided that $F_x = F_w$ and $F_y = F_z$ for some new clocks w and z, we could say that p_2 should show the same behaviour as $p_5 \equiv \{\!|w,z|\!\}\,(\{w,z\} \mapsto a;\mathbf{0} + \{\!|y|\!\}\,\{x,y\} \mapsto b;\mathbf{0})$. In fact p_2 and p_5 are α-congruent and this, like in Section 5.3, is the methodology we use to broaden the set of ⏚ processes to which we can give semantics.

Notice, however, that not every substitution will be useful in this case. Since the stochastic behaviour of a ⏚ process depends on the distribution of the clocks, replacing a clock by another one with a different distribution may affect the semantics. As a consequence we can only use substitutions that preserve distribution functions and in such a case we say that it is stochastically appropriate.

Definition 10.9. Let $C, C' \subseteq \mathcal{C}$. A function $g : C \to C'$ is *stochastically appropriate* if g is injective and for every $x \in C$, $F_x = F_{g(x)}$.

$\xi(\mathbf{0}) = \mathbf{0}$	
$\xi(C \mapsto p) = \xi(C) \mapsto \xi(p)$	
$\xi(\{\!\{C\}\!\}\, p) = \{\!\{C'\}\!\}\, \xi\, \xi_{[C'/C]}(p)$	$(C' \cap \xi(fv(p) - C) = \varnothing)$
$\xi(\mathsf{op}(p)) = \mathsf{op}(\xi(p))$	$(\mathsf{op} \in \{a;\_,\_[f]\})$
$\xi(p \oplus q) = \xi(p) \oplus \xi(q)$	$(\oplus \in \{+, \|\|_A, \between_A, \|_A\})$
$\xi(X) = X_\xi$	(provided $X = p$ and $X_\xi = \xi(p)$)

Table 10.5: Substitution in 🌳

In the context of 🌳 we restrict to *stochastically appropriate substitutions*. Therefore, $\xi_{[C'/C]}$ represents some stochastically appropriate bijection in $C \to C'$. (We reuse notations like $\xi\, \xi_{[C'/C]}$ or $\xi_{[C'/C]}(x)$ given on page 56.) □

Definition 10.10. A *substitution* on 🌳 is a stochastically appropriate function $\xi : \mathcal{C} \to \mathcal{C}$ that extends recursively to 🌳 terms as defined in Table 10.5. □

A 🌳 term α-converts to another if it has the same syntactic form with the exception that bound clocks might be renamed together with their binder according to a stochastically appropriate substitution.

Definition 10.11. The relation $\simeq_\alpha \subseteq$ 🌳 $\times$ 🌳 is the least relation satisfying rules in Table 10.6. We call $\simeq_\alpha$ *α-congruence*, and if $p \simeq_\alpha q$, we say that p and q are *α-congruent*, or that one can *α-convert* to the other. □

$\simeq_\alpha$ can be proved to be an equivalence in 🌳 and as a consequence of the definition it is also a congruence. Moreover, it can be proved by straightforward induction that $\simeq_\alpha$ preserves the set of free variables. Thus, we state,

Proposition 10.12. *(a)* $\simeq_\alpha$ *is a congruence.* *(b)* $p \simeq_\alpha q$ *implies* $fv(p) = fv(q)$.

In the previous section, we have seen that even simple 🌳 terms like p_1 or p_2, given in equations (10.2) and (10.3), respectively, may not have a defined semantics simply because of inappropriately named clocks. Yet each one of these processes can be appropriately α-converted in a process without clock conflict. In fact, we can prove that every 🌳 process defined with process variables in a guarded recursive specification can be α-converted into a new one whose behaviour is locally definable. This is a consequence of the following theorem.

$$\mathbf{0} \simeq_\alpha \mathbf{0} \qquad \frac{X \in \mathcal{V}}{X \simeq_\alpha X} \qquad \frac{p \simeq_\alpha q \quad X = p \quad Y = q}{X \simeq_\alpha Y}$$

$$\frac{\xi_{[C^\star/C]}(p) \simeq_\alpha \xi_{[C^\star/C']}(q) \quad C^\star \cap (\ (fv(p) - C) \cup (fv(q) - C')\ ) = \varnothing}{\{\!|C|\!\}\, p \simeq_\alpha \{\!|C'|\!\}\, q}$$

$$\frac{p \simeq_\alpha q \quad \mathsf{op} \in \{a;\_, C \mapsto \_, \_[f]\}}{\mathsf{op}(p) \simeq_\alpha \mathsf{op}(q)} \qquad \frac{p \simeq_\alpha q \quad p' \simeq_\alpha q' \quad \oplus \in \{+, ||_A, \between_A, |_A\}}{p \oplus p' \simeq_\alpha q \oplus q'}$$

Table 10.6: α-conversion in ♤

Theorem 10.13. *Let $p \in$ ♤ be a guarded process. Then there is a $q \in$ ♤ whose behaviour is locally definable and $p \simeq_\alpha q$.*

Proof sketch. To prove that $\neg\mathsf{cv}(q)$ we can simply proceed as in the proof of Theorem 5.12. So, it remains to show that $\kappa(q)$ is finite and that for every outgoing arrow $q \xrightarrow{a,C} q'$, C is also finite. But this can be proved by straightforward structural induction taking into account that q is guarded since $p \simeq_\alpha q$ and p is guarded, and as a consequence case $q \equiv X \in \mathcal{V}$ does not occur. *Q.E.D.*

The theorem is not true in general for unguarded recursion. For instance, process X defined by the equation $X = X + \{\!|x|\!\}\, \{x\} \mapsto a; \mathbf{0}$, which has local conflict of variables, cannot be α-converted into a process with locally definable behaviour. Neither does process X_0 defined by the recursive specification $\{X_n = X_{n+1} + \{\!|x_n|\!\}\, \{x_n\} \mapsto a; \mathbf{0} \mid n \in \mathbb{N}\}$[2].

The semantics of ♤ up to α-conversion is built on top of the rules in Table 10.3 and the new rule

$$\textbf{alpha} \quad \frac{p \xrightarrow{a,C} p' \quad p' \simeq_\alpha q' \quad \neg\mathsf{cv}(p) \quad \neg\mathsf{cv}(q')}{p \xrightarrow{a,C} q'}$$

This rule is the counterpart of the rule **alpha** for ♡. It allows to find an α-congruent target that is locally conflict-free whenever it is possible. To ensure that an edge never takes to a term with conflict, those undesirable edges are pruned. Therefore, the new semantics define a new edge $\xrightarrow{}'$ such that $p \xrightarrow{a,C}' q$ if and only if $p \xrightarrow{a,C} q$ and p and q are locally conflict free.

Since there are terms that cannot be α-converted to another whose behaviour is locally definable we need to rule them out.

[2]Notice that similar processes in ♡ —as they could be, for instance, $X = X + \{\!|x|\!\}\, (x > 1) \mapsto a; \mathbf{0}$ and $\{X_n = X_{n+1} + \{\!|x_n|\!\}\, (x_n > n) \mapsto a; \mathbf{0} \mid n \in \mathbb{N}\}$— do not suffer of conflict of variables and timed automata can be associated without any problem.

Definition 10.14. The behaviour of $p \in \mathcal{O}$ is *α-definable* if there is a $p_0 \in \mathcal{O}$ such that $p \simeq_\alpha p_0$, $\mathsf{def}(p_0)$, and for every $n > 0$, whenever

$$p_0 \xrightarrow{a_0,C_0} p_1 \xrightarrow{a_1,C_1} p_2 \cdots p_n \xrightarrow{a_n,C_n} p_{n+1},$$

and $\mathsf{def}(p_i)$, $0 < i \leq n$, then there is a $q \in \mathcal{O}$ such that $\mathsf{def}(q)$ and $p_{n+1} \simeq_\alpha q$. We remark that $\longrightarrow$ is calculated as the least relation satisfying rules in Table 10.3 and **alpha**. □

We insist that if $\kappa(p)$ is infinite for some $\mathcal{O}$ term p, then it is not possible to α-convert this term to some process q such that $\kappa(q)$ is finite. So, in the previous definition, if $\neg\mathsf{def}(p_{n+1})$, but $\neg\mathsf{cv}(p_{n+1})$, then it is not possible to find a q such that, $p_{n+1} \simeq_\alpha q$ and $\mathsf{def}(q)$. For this reason there was no need to have a more general rule **alpha** in which we require $\mathsf{def}(p)$ and $\mathsf{def}(q)$ instead of $\neg\mathsf{cv}(p)$ and $\neg\mathsf{cv}(q')$.

Definition 10.15. Let $\mathcal{O}^\alpha \stackrel{\text{def}}{=} \{p \in \heartsuit \mid \mathsf{def}(p)$ and the behaviour of p is α-definable$\}$. The *semantics of* $\mathcal{O}$ *up to α-conversion* is defined as the stochastic automaton $SA(\mathcal{O}^\alpha) = (\mathcal{O}^\alpha, \mathcal{A}, \mathcal{C}, \longrightarrow', \kappa)$ where $\mathcal{A}$ and $\mathcal{C}$ are as in Definition 10.1, κ is defined by the least relations satisfying the rules in Table 10.3, and $\longrightarrow' \stackrel{\text{def}}{=} \longrightarrow \cap (\mathcal{O}^\alpha \times \mathcal{A} \times \wp_{\text{fin}}(\mathcal{C}) \times \mathcal{O}^\alpha)$ with $\longrightarrow$ been the least relation satisfying rules in Table 10.3 and rule **alpha** above.

The semantics up to α-conversion of a process $p \in \mathcal{O}$ is defined whenever p is α-definable by any rooted stochastic automaton $(SA(\mathcal{O}^\alpha), q)$, provided $p \simeq_\alpha q \in \mathcal{O}^\alpha$. □

By Theorem 10.13 it follows that guarded processes are guaranteed to have semantics up to α-conversion.

Proposition 10.16. *If $p \in \mathcal{O}$ is defined with variables in a guarded recursive specification, then the semantics of p up to α-conversion is defined.*

In the following we show that the semantics up to α-congruence is well defined, in the sense that two α-congruent terms are symbolically bisimilar. We state this in the following.

Theorem 10.17. *Let $p, q \in \mathcal{O}^\alpha$, i.e., their behaviours are not only locally definable but also α-definable. If $p \simeq_\alpha q$ then $p \sim_{\&} q$ where p and q should be interpreted here as states in $SA(\mathcal{O}^\alpha)$.*

The proof of this theorem as well as the one of Theorem 10.18 below can be found in Appendix F.1. As a corollary we have that the semantics up to α-conversion is well defined.

Corollary 10.17.1. *For every $p \in \mathcal{O}^\alpha$, the semantics of p up to α-conversion is unique up to symbolic bisimulation, i.e., if $p \simeq_\alpha q$, $p \simeq_\alpha q'$, and q and q' are locally definable, then $(SA(\mathcal{O}^\alpha), q)$ and $(SA(\mathcal{O}^\alpha), q')$ are symbolically bisimilar.*

The following theorem states that the original semantics given in Definition 10.7 coincides, modulo symbolic bisimulation, with the semantics up to α-conversion.

Theorem 10.18. *Let $p \in \mathcal{O}^{\mathsf{cf}}$, i.e., the behaviour of p is definable. Then, the rooted stochastic automata $(SA(\mathcal{O}^{\mathsf{cf}}), p)$ and $(SA(\mathcal{O}^\alpha), p)$ are symbolically bisimilar.*

10.4 Representability

We can represent any stochastic automaton by means of a ♤ term p in a way that the semantics of p is isomorphic to the given stochastic automaton. Moreover, the behaviour of p is definable.

The way we proceed to prove this result follows closely the lines of Theorem 5.18 in Section 5.4. We mention that given a (finite or infinite) index set I, the *generalised summation* $\sum_{i\in I} C_i \mapsto a_i; p_i$ can be defined in ♤ just like we did for ♡. Moreover, the notion of *(symbolic) reachability* can be easily adapted to stochastic automata. Thus, we state:

Theorem 10.19. *For every rooted stochastic automaton (SA, s_0) with denumerable branching —i.e., for every location s the set $\{s \xrightarrow{a,C} s' \mid a \in \mathcal{A} \wedge C \in \mathcal{C} \wedge s' \in \mathcal{S}\}$ is denumerable— there is a definable recursive specification E in ♤, with root X_{s_0} such that the reachable part of (SA, s_0) and the reachable part of $(SA(♤^{\mathsf{cf}}), X_{s_0})$ are isomorphic.*

Since the proof presents no technical variation with respect to the one of Theorem 5.18, we will omit it and only give a simple example.

Example 10.20. Take the automaton of Example 9.2. The reader is invited to check that the stochastic automaton $(SA(♤^{\mathsf{cf}}) \upharpoonright \{X_{s_0}, X_{s_1}, X_{s_2}\}, X_{s_0})$ is isomorphic to that of Example 9.2. In this case, the process variables X_{s_0}, X_{s_1}, and X_{s_2} are defined by the following recursive specification.

$$\begin{aligned}
X_{s_0} &= \{\!|x|\!\}\,\{x\} \mapsto on; X_{s_1} \\
X_{s_1} &= \{\!|x, y|\!\}\ (\ \{x\} \mapsto on; X_{s_1} \\
&\qquad\qquad + \{y\} \mapsto off; X_{s_2}) \\
X_{s_2} &= \{\!|\varnothing|\!\}\,\{x\} \mapsto on; X_{s_1}
\end{aligned}$$

The way we proceeded to obtain this recursive specification follows the technique given in the proof of Theorem 5.18. □

By Definition 10.15, the following corollary of Theorem 10.19 is straightforward.

Corollary 10.19.3. *For every α-definable $p \in ♤^{\alpha}$ there is a definable $q \in ♤^{\mathsf{cf}}$ such that $p \sim_{\&} q$.*

The following is a corollary of the proof of Theorem 10.19 (see the proof of Theorem 5.18 in page 61)

Corollary 10.19.4. *Under Theorem 10.19, the recursive specification E that represents the stochastic automaton SA is, moreover, guarded if SA is finitely branching, and it is regular if SA is finite.*

10.5 Congruences

Assume that $\overline{\mathsf{ck}}$ is also an operation of the language. Under this condition, it is not difficult to observe that the rules for $\longrightarrow$ given in Table 10.3 are in *path* format (see Chapter 2, page 20). Moreover, predicate κ can be alternatively defined in terms of a proof system (like we did for $\heartsuit$, see Table B.2) such that it is equivalent to its definition in Table 10.3. Such a proof system can be checked to be in *path* format as well. In addition, rules (10.1) given on page 138 for operation $\overline{\mathsf{ck}}$ are also in *path* format. All together, the entire system is in *path* format, and as a consequence of (Baeten and Verhoef, 1993), structural bisimulation is a congruence for all the $\spadesuit$ operations as well as for $\overline{\mathsf{ck}}$. Since in addition none of the rules has look-ahead, by (Rensink, 1997), structural bisimulation is also a congruence for recursion (see Chapter 2, page 20).

Theorem 10.21. *Let p and q be two $\spadesuit$ terms such that $p \sim_s q$. For any $\spadesuit$ context $\mathbf{C}[\ ]$ it holds that $\mathbf{C}[p] \sim_s \mathbf{C}[q]$. Moreover, $\sim_s$ is also a congruence for recursion in $\spadesuit$ (see Theorem 2.13 for an appropriate formulation).*

Like structural and symbolic bisimulation, the different probabilistic bisimulations also extend naturally to $\spadesuit$: $p \sim_c q$ if there are p' and q' such that their behaviours are α-definable, $p \simeq_\alpha p'$, $q \simeq_\alpha q'$, and $p' \sim_c q'$ where p' and q' are locations in the stochastic automaton $SA(\spadesuit^\alpha)$ —or $SA(\spadesuit^{\mathsf{cf}})$, if it applies— (i.e., $(p', v) \sim_p (q', v)$ for all valuation v, where (p', v) and (q', v) are states in the probabilistic transition system resulting after interpreting the stochastic automaton $SA(\spadesuit^\alpha)$ (or equivalently $SA(\spadesuit^{\mathsf{cf}})$) as a closed system). Similarly, $p \sim_o q$ if there are p' and q' such that their behaviours are α-definable, $p \simeq_\alpha p'$, $q \simeq_\alpha q'$, and $p' \sim_o q'$ where p' and q' are locations in the stochastic automaton $SA(\spadesuit^\alpha)$ —or $SA(\spadesuit^{\mathsf{cf}})$, if possible.

In Section 9.3, we have already anticipated that probabilistic bisimilarity in the closed behaviour is not a congruence. This is shown by the following examples.

Example 10.22. *$\sim_c$ is not a congruence for parallel composition.* Processes $p_1 \equiv a; \mathbf{0} + \{\!|x|\!\}\ \{x\} \mapsto b; \mathbf{0}$ and $p_2 \equiv a; \mathbf{0} + \{\!|x|\!\}\ \{x\} \mapsto c; \mathbf{0}$ $(b \neq c)$ are closed p-bisimilar if $F_x(0) = 0$, since in both cases only the action a at time 0 can be performed. However, $p_1 \,||_a\, \mathbf{0}$ and $p_2 \,||_a\, \mathbf{0}$ are not. In this context, the execution of action a is preempted since there is no possible synchronisation, and therefore b or c may happen (at a certain time greater than 0). This example is depicted in Figure 10.2. □

Example 10.23. *$\sim_c$ is not a congruence for the triggering condition.* Take p_1 and p_2 as in the previous example. Then, $\{y\} \mapsto p_1 \not\sim_c \{y\} \mapsto p_2$. To understand this, assume F_x is a uniform distribution in $[1, 2]$. For any valuation v in which y takes a value greater than 2, all edges become enabled at the same time $v(y)$. In particular, edges $\{y\} \mapsto p_1 \xrightarrow{b,\{x,y\}} \mathbf{0}$ and $\{y\} \mapsto p_2 \xrightarrow{c,\{x,y\}} \mathbf{0}$ which are those edges which make the difference. □

These are precisely the situations that occur when dealing with open systems, and hence they justify the introduction of the interpretation of stochastic automata as open

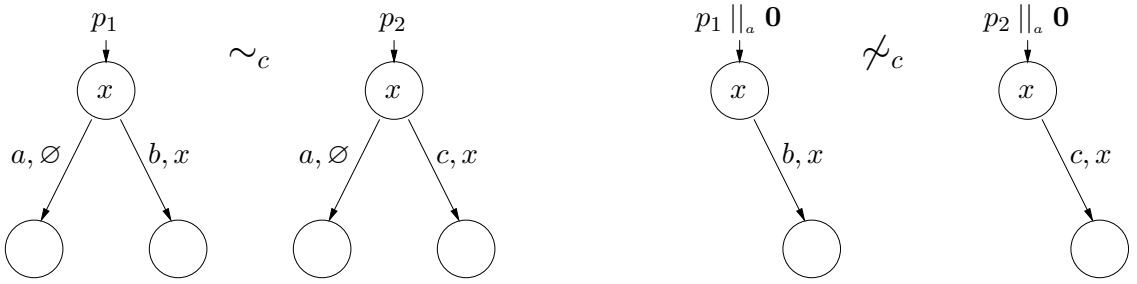


Figure 10.2: Closed p-bisimilarity is not a congruence

systems. In fact, open p-bisimilarity is a congruence in ♤ as it is stated in the following theorem whose proof can be found in Appendix F.2.

Theorem 10.24. *Let p and q be two ♤ terms whose behaviours are α-definable. Assume $p \sim_o q$. Let $\mathbf{C}[\ ]$ be a ♤ context such that $\mathbf{C}[p]$ and $\mathbf{C}[q]$ are α-definable. Then, it holds that $\mathbf{C}[p] \sim_o \mathbf{C}[q]$.*

10.6 Summary

In Figure 10.3 we summarise the results we have achieved in this chapter. Black arrows identify the interpretation function, while grey arrows can be read as "are expressible in terms of". Thus, an α-definable term p can be interpreted in terms of a stochastic automaton using the semantics up to α-conversion. If p is in addition definable, we can use the direct semantics without any need of α-conversion. One way or the other, the obtained automata are equivalent up to symbolic bisimulation. So, if we give semantics to the obtained stochastic automata, the underlying probabilistic transition systems are open p-bisimilar as well as closed p-bisimilar. This is an immediate consequence of the fact that symbolic bisimilarity implies open and closed p-bisimilarity as it was stated in Chapter 9.

In addition a generalised semi-Markov process can be described by a stochastic automaton in such a way that the behaviour (as a closed system) is preserved up to probabilistic bisimulation. Moreover, any GSMS can be also represented by a (definable) ♤ term. This is an immediate consequence of the fact that any (denumerably branching) stochastic automaton can also be represented by a (definable) ♤ term, as stated by Theorem 10.19.

10.7 Related Work

Apart from the Markovian process algebras (e.g. Hillston, 1996; Hermanns and Rettelbach, 1994; Bernardo and Gorrieri, 1998; Hermanns, 1998), some general stochastic process algebras have been introduced.

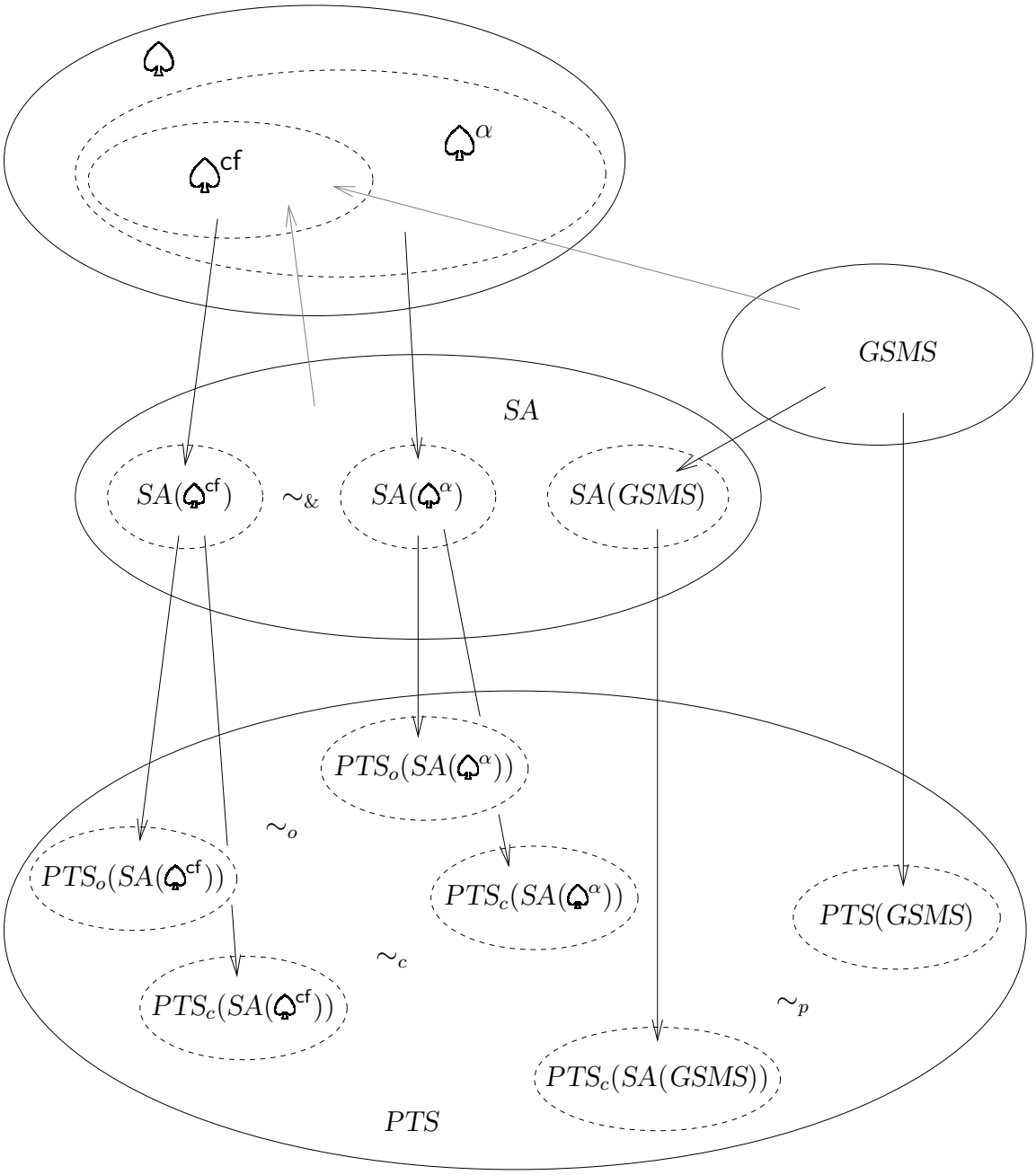


Figure 10.3: Summary of semantics

- TIPP (Götz et al., 1993) is the earliest approach to the general case. Its syntax has the integrated prefix $a_F; p$ which in ♤ corresponds to $\{\!|x_F|\!\}\,\{x_F\} \mapsto a; p$. Its semantics is based on labelled transition systems in which transitions are decorated with the associated distribution function and, to keep track of the execution of parallel processes, a number that indicates how many times an action has not been chosen to execute. This number introduces infinite semantic objects, even for simple regular processes. Priami (1996) followed a similar approach to give semantics to a stochastic extension of the π-calculus. In this case transitions are decorated with locality information to keep track which process performed it.

- Harrison and Strulo (1995) introduced a process algebra for discrete event simulation. The concerns of randomly setting a timer, expiration of such a timer, and actual activity are split in a rather similar way to ours. The semantic model is similar to our probabilistic transition systems where non-deterministic transitions are instead split into discrete transitions and timed transitions. As a consequence, semantic objects contain usually uncountably many states and transitions. The process algebra includes an urgent and a delayable prefixing, so its interpretation combines both views of closed and open system.

- Brinksma et al. (1995) studies a semantics for a process algebra similar to TIPP in terms of a stochastic extension of event structures. This model seems to be more natural to deal with general distributions since activities that are not causally dependent (i.e. concurrent activity) are not related in the model, contrarily of what occurs in interleaving based models. However, recursive processes always have associated an infinite semantic object.

- A general semi-Markovian process algebra based on EMPA is discussed by Bravetti et al. (1998). Terms in this process algebra have semantics in a semi-interleaving model that allows for refinement of actions (the so-called ST-semantics). Although this calculus preserves finiteness of semantic objects in a reasonable way (like ours), the manner in which semantics is given is not straightforward. A nice characteristic of this process algebra is that it represents the GSMP model in a complete way.

Among the stochastic process algebras enumerated above, (Harrison and Strulo, 1995) is the closest to ♤. As ♤, (Harrison and Strulo, 1995) and Hermanns's *IMC —Interactive Markov Chains—* (1998) also allow non-determinism. In all the other cases (including the Markovian process algebras), choice is always solved either probabilistically or by the race condition.

Chapter 11

Equational Theory for ♤

In the last two chapters we have defined the stochastic automata model and the process algebra ♤ as a language that allows for compositional specifications of stochastic systems. In this chapter we provide an equational theory for ♤, and hence an axiomatic framework to manipulate stochastic automata. The major achievement of this axiomatisation is that an expansion law can be easily derived.

The necessity of a law to express how to expand the parallel composition had restricted the stochastic process algebras to only Markovian process algebras (e.g., Hillston, 1996; Buchholz, 1994; Hermanns and Rettelbach, 1994; Hermanns, 1998; Bernardo, 1999). Attempts to define stochastic process algebras with general distributions followed two different approaches. The first idea, was to cope with a cumbersome interleaving solution in the hope of eventually obtaining an expansion law with, perhaps, conditional distributions (Götz et al., 1993; Harrison and Strulo, 1995; Priami, 1996). A second choice, which gave an apparently neater solution, was to move to a non-interleaving setting and hence to abandon the possibility of an expansion law, as in the case of (Brinksma et al., 1995). Brinksma et al. (1995) (see also Katoen, 1996) do not, in fact, provide an axiomatisation for their process algebra. The solution proposed by Götz et al. (1993), and followed by Priami (1996), is to deal with conditional probabilities that leads to more complicated models, in which distribution functions are not independent any longer. Again, none of these works provide an axiomatisation and neither does the algebra of (Bravetti et al., 1998) based on ST-semantics. Harrison and Strulo (1995) (see also Strulo, 1993) do provide a set of axioms, although it is not complete enough as to derive a general expansion law. In particular, it is not possible to expand the parallel composition of terms with stochastic information.

In the following, we define an equational theory which is sound and complete with respect to structural bisimulation. We also introduce some axioms for open p-bisimulation and give some derived results. We first discuss the core algebra with only the basic operations and then we extend it with the static operations. We will show shortly that the discussed theories are conservative extensions of the theory for ♧.

A1	$p + q = q + p$	
A2	$(p + q) + r = p + (q + r)$	
A3	$a; p + a; p = a; p$	
A4	$p + \mathbf{0} = p$	
T1	$C \mapsto \mathbf{0} = \mathbf{0}$	
T2	$\varnothing \mapsto p = p$	
T3	$C \mapsto C' \mapsto p = C \cup C' \mapsto p$	
T4	$C \mapsto \{\!\{C'\}\!\}\, p = \{\!\{C'\}\!\}\, C \mapsto p$	if $C \cap C' = \varnothing$
T5	$C \mapsto (p + q) = C \mapsto p + C \mapsto q$	
CS1	$\{\!\{\varnothing\}\!\}\, p = p$	
CS2	$\{\!\{C\}\!\}\, \{\!\{C'\}\!\}\, p = \{\!\{C \cup C'\}\!\}\, p$	
CS3	$\{\!\{C\}\!\}\, p + \{\!\{C'\}\!\}\, q = \{\!\{C \cup C'\}\!\}\, (p + q)$	if $C \cap (fv(q) \cup \kappa(q)) = C' \cap (fv(p) \cup \kappa(p)) = \varnothing$

Table 11.1: Axioms for $\clubsuit^b$

11.1 Basic Axioms for Structural Bisimulation

In this section we introduce an equational theory for the sub-language obtained with the basic operations. So the signature we consider for $\clubsuit^b$ only includes prefixing, nil, summation, triggering condition, and clock setting. We show that this axiomatisation is sound and complete for structural bisimulation.

The axioms for $\clubsuit^b$ are given in Table 11.1 and can be explained as follows. The choice is commutative (**A1**) and associative (**A2**). Axiom **A3** states a weaker notion of idempotency of $+$ and **A4** states that $\mathbf{0}$ is the neutral element for $+$. Axioms **T1**–**T5** show the way in which triggering conditions can be simplified. In particular, **T3** defines how to reduce nested triggering conditions into only one, and axioms **T4** and **T5** say how to move clock settings and summations out of the scope of a guard. Axiom **CS1** says that it is irrelevant to set an empty set of clocks. **CS2** gathers all the clocks settings in only one operation and **CS3** moves clocks settings out of the scope of a summation.

Definition 11.1. The equational theory for $\clubsuit^b$ is defined as follows. The signature contains the constant $\mathbf{0}$, the unary operators $a;\_$, $C \mapsto \_$, and $\{\!\{C\}\!\}\_$, for every $a \in \mathcal{A}$, and $C \in \wp_{\text{fin}}(\mathcal{C})$, and the binary operation $+$. The axiom system is given in Table 11.1. We identify with $sig(\clubsuit^b)$ and $ax(\clubsuit^b)$ the signature and axiom system of $\clubsuit^b$. □

We observe that idempotency is not valid in $\clubsuit$ in general. To understand this, consider

the process $p \equiv \{\!|x|\!\}\,\{x\} \mapsto a;\mathbf{0}$ where G is uniformly distributed in the interval on $[0,2]$. The probability that a occurs in the interval $[0,1]$ in process p is $\frac{1}{2}$, while in process $p+p$ such probability is $\frac{3}{4}$. It follows that $p \not\sim_c p+p$ and so they are not related by any of the other bisimulations (which are finer). However, if we restrict p to be a process that does not begin with some stochastic information, idempotency does hold.

Property 11.2. Let $p \in \mathbb{O}^b$ such that any clock setting operation happens within the context of a prefix. That is, every subterm $\{\!|C|\!\}\,r$ of p occurs only in another subterm of p of the form $a;q$. Then by using axioms $ax(\mathbb{O}^b)$ and structural induction, it holds that $p = p+p$.

Proof. In effect, we proceed by structural induction.

Case $\mathbf{0}$. $\mathbf{0} \overset{\text{A4}}{=} \mathbf{0}+\mathbf{0}$

Case $a;p$. $a;p \overset{\text{A3}}{=} a;p + a;p$

Case $C \mapsto p$. $C \mapsto p \overset{\text{ind.}}{=} C \mapsto (p+p) \overset{\text{T5}}{=} C \mapsto p + C \mapsto p$

Case $\{\!|C|\!\}\,p$. It does not apply.

Case $p+q$. $p+q \overset{\text{ind.}}{=} (p+p)+(q+q) \overset{\text{A1,A2}}{=} (p+q)+(p+q)$ *Q.E.D.*

Notice that p does not necessary have to be closed. It only needs to be guarded. We will give a "more stochastic" version of idempotency in the next section.

In the following we prove that the axiom system $ax(\mathbb{O}^b)$ is sound and complete for structural bisimulation.

Theorem 11.3. *Let p and q be $\mathbb{O}$ terms. If $p = q$ is proved by means of equational reasoning using $ax(\mathbb{O}^b)$, then $p \sim_s q$.*

Proof. For every axiom $p = q$ in $ax(\mathbb{O}^b)$, we define the relation $R \overset{\text{def}}{=} \{\langle p,q\rangle, \langle q,p\rangle\}$. It can be straightforwardly checked that R is a structural bisimulation up to $\sim_s$ in all the cases. From here the theorem follows. *Q.E.D.*

The following proposition will be useful in the proof of Theorem 11.6. It states that axioms in Table 11.1 preserve definability. The proof is given in Appendix G.1.

Proposition 11.4. *Let $p = q$ be a particular instance of any axiom in Table 11.1. Then the behaviour of p is locally definable if and only if the behaviour of q is locally definable.*

The axiom system $ax(\mathbb{O}^b)$ allows to derive a normal form for every term. The existence of such a normal form is a key factor in the proof of completeness.

Definition 11.5. We define the set $B \subseteq \mathbb{O}$ of *basic terms* using an auxiliary set B'. We proceed inductively as follows:

- $\mathbf{0} \in B'$
- $p \in B$, $C \subseteq \wp_{\text{fin}}(\mathcal{C})$ and $a \in \mathcal{A} \quad \Longrightarrow \quad C \mapsto a; p \in B'$
- $p, q \in B' \quad \Longrightarrow \quad p + q \in B'$
- $p \in B'$, $C \subseteq \wp_{\text{fin}}(\mathcal{C}) \quad \Longrightarrow \quad \{\!|C|\!\}\, p \in B$

Terms in B' are like basic terms, but with the clock setting operation only appearing within the scope of a prefix construction. We say that a term in B' is a *pre-basic term* . Notice that a basic term has the general format (modulo **A1**, **A2**, **A3**, and **A4**)

$$p = \{\!|C|\!\} \left(\sum_{i \in I} C_i \mapsto a_i; p_i \right)$$

where I is finite, each p_i is a basic term, and $\sum$ is as in Section 10.4. Notice that a pre-basic term has the general format

$$p = \sum_{i \in I} C_i \mapsto a_i; p_i$$

where I is finite and each p_i is a basic term. Moreover, if p is a pre-basic term then $\{\!|C|\!\}\, p$ is a basic term for any $C \in \mathcal{C}$. □

The following theorem can be proven by structural induction using the axioms. Its proof is given in Appendix G.2.

Theorem 11.6. *For every term $p \in ♤^b$ whose behaviour is definable, there is a basic term q such that $p = q$ can be proven by means of the axiom system $ax(♤^b)$. Moreover, the behaviour of q is definable as well.*

Notice that if $p \in ♤^b$, we can only require that it is conflict-free and be sure that it is definable in general, due to the fact that p is a closed (or finite) term and by construction no infinite clock setting $\kappa(p)$ can be induced. In addition, notice that the sets of sequential finite terms whose behaviour is definable is in fact the set $♤^b \cap ♤^{\mathsf{cf}}$.

We dedicate the rest of the section to prove completeness of $ax(♤^b)$ with respect to $\sim_s$. The proof proceeds similarly to the proof of completeness of ♡ (see Section 6.1), but it is simpler.

Definition 11.7. Let p and q be two pre-basic terms. We say that p is a *summand* of q, notation $p \leq q$, if $q = q + p$ can be proved in $ax(♤^b)$. □

The proof of the following proposition follows straightforwardly from axioms **A1**, **A2**, and Property 11.2.

Proposition 11.8. *$\leq$, as defined above, is a partial ordering on the set B' of pre-basic terms.*

In addition, using axioms **A4**, **CS1**, and **CS3**, it can be shown that $\mathbf{0}$ is the unique minimal element for $\leq$. The notion of summand relates to the operational semantics as stated in the following proposition, whose proof is straightforward.

Proposition 11.9. *Let p be a pre-basic term whose behaviour is definable.*

1. *If $C \mapsto a; p'$ is a summand of p then $p \xrightarrow{a,C} p^\star$ and $p^\star = p'$.*
2. *Conversely, $p \xrightarrow{a,C} p^\star$ implies that there is a summand $C \mapsto a; p'$ of p such that $p^\star = p'$.*

Here, equality means that $p^\star = p'$ can be derived from $ax(\spadesuit^b)$, and $\longrightarrow$ is the edge of the straight semantics $SA(\spadesuit^{\mathsf{cf}})$ given in Defintion 10.7.

We are now in a condition to prove completeness.

Theorem 11.10. *The equational theory $ax(\spadesuit^b)$ is sound and complete for the algebra $((\spadesuit^b \cap \spadesuit^{\mathsf{cf}}), sig(\spadesuit^b), \sim_s)$.*

Proof. Soundness has already being stated in Theorem 11.3. In the following we prove completeness.

Let $p, q \in \spadesuit^b \cap \spadesuit^{\mathsf{cf}}$ such that $p \sim_s q$. Because of Theorem 11.6 and soundness, there are basic terms $\hat{p}$ and $\hat{q}$ whose behaviors are definable such that $\hat{p} \sim_s p \sim_s q \sim_s \hat{q}$.

Let $\hat{p} = \{\!|C|\!\}\, p'$ and $\hat{q} = \{\!|C'|\!\}\, q'$. Because of the second transfer property of $\sim_s$ (see Definition 9.11) we have that $C = C'$. For the rest of the proof, we proceed by induction on the maximum number of nested prefixes in $\hat{p}$ and $\hat{q}$. If it is 0 then $\hat{p} = \{\!|C|\!\}\, \mathbf{0} = \hat{q}$. Otherwise, $\hat{p} = \{\!|C|\!\}\, p'$ and $\hat{q} = \{\!|C|\!\}\, q'$ with

$$p' = \sum_{i \in I} C_i \mapsto a_i; p_i \quad \text{and} \quad q' = \sum_{j \in J} C_j \mapsto a_j; q_j.$$

Take a summand $C_k \mapsto a_k; p_k$ of p'. Then $p' \xrightarrow{a_k, C_k} p^\star$ with $k \in I$ and $p_k = p^\star$. It is easy to check that $\hat{p} \sim_s \hat{q}$ implies $p' \sim_s q'$. As a consequence, by the first transfer property in Definition 9.11, $q' \xrightarrow{a_k, C_k} q^\star$ and $p^\star \sim_s q^\star$. By Proposition 11.9, we can assume there is an $h \in J$ such that, $a_k \equiv a_h$, $C_k = C_h$, $q^\star = q_h$, and $C_h \mapsto a_h; q_h$ is a summand of q'. Since $p_k \sim_s p^\star \sim_s q^\star \sim_s q_h$, by induction $p_k = q_h$. As a consequence $C_k \mapsto a_k; p_k = C_h \mapsto a_h; q_h \leq q'$.

Proceeding in a similar way for every summand of p', we conclude that $p' \leq q'$. By symmetry $q' \leq p'$. As a consequence $p' = q'$. Finally, we calulcate $p = \hat{p} = \{\!|C|\!\}\, p' = \{\!|C|\!\}\, q' = \hat{q} = q$. *Q.E.D.*

11.2 Axioms and Laws for Open p-Bisimulation

We have introduced a sound and complete axiomatisation for $\sim_s$. Yet we can go a little further and give axioms that, although they do not preserve structural bisimulation, they

Sy1	$\{\!\{C\}\!\}\, p = p$	if $C \cap fv(p) = \varnothing$
Sy2	$C \mapsto a; C \mapsto p = C \mapsto a; p$	
Sy3	$\{\!\{C\}\!\}\, p = \{\!\{z\}\!\}\, \xi_{\{z/C\}} p$	if $\mathsf{Lnk}(C,p) \wedge z \notin fv(p) \wedge \forall t \in \mathbb{R}_{\geq 0}.\ F_z(t) = \prod_{x \in C} F_x(t)$
A3′	$\{\!\{x,y\}\!\}\, (\{x\} \mapsto a; p + \{y\} \mapsto a; p) = \{\!\{z\}\!\}\, \{z\} \mapsto a; p$	if $\{x,y,z\} \cap fv(p) = \varnothing \wedge \forall t \in \mathbb{R}_{\geq 0}.\ F_z(t) = F_{\min\{x,y\}}(t)$
Red	$\{\!\{x\}\!\}\, \{x\} \mapsto p = \{\!\{x\}\!\}\, p$	if $F_x(0) = 1$

Ln1	$\dfrac{C \cap fv(p) = \varnothing}{\mathsf{Lnk}(C,p)}$	**Ln4**	$\dfrac{\mathsf{Lnk}(C,p) \quad C \cap C' = \varnothing}{\mathsf{Lnk}(C, C' \mapsto p)}$
Ln2	$\dfrac{\mathsf{Lnk}(C,p)}{\mathsf{Lnk}(C, a; p)}$	**Ln5**	$\dfrac{\mathsf{Lnk}(C,p) \quad C \cap C' = \varnothing}{\mathsf{Lnk}(C, \{\!\{C'\}\!\}\, p)}$
Ln3	$\dfrac{C \subseteq C' \quad C \cap fv(p) = \varnothing}{\mathsf{Lnk}(C, C' \mapsto p)}$	**Ln6**	$\dfrac{\mathsf{Lnk}(C,p) \quad \mathsf{Lnk}(C,q)}{\mathsf{Lnk}(C, p+q)}$

Table 11.2: More axioms for $\clubsuit^b$

do preserve open p-bisimilarity. In this sense the axioms we give in Table 11.2 speak about the stochastic properties of a system instead of restricting to only structural manipulation. We denote by $ax(\clubsuit^b) + \mathbf{O}$ the extension of $ax(\clubsuit^b)$ with the set of axioms enumerated in Table 11.2.

Axioms **Sy1**–**Sy3** respect symbolic bisimulation. **Sy1** eliminates redundant clocks, that is, it is not necessary to set clocks that are not free in a process p since either they are never used or they are going to be set later on before they are actually used. Axiom **Sy2** says that clocks that have already been used —i.e., they have already expired— are not useful anymore, at least not before they are set again. The most involved axiom is **Sy3**. It takes care of relating clocks as symbolic bisimulation does. Recall that symbolic bisimulations contains triplets of the form $\langle p_1, p_2, SR \rangle$ where SR is the synchronisation relation that ensures that clocks are synchronised in time and stochastics. Predicate Lnk, which is defined in rules **Ln1**–**Ln6**, checks whether a set C of clocks is synchronised in time (or *linked*) in process p, that is, whether they are set and used at the same moment by process p. **Ln1** and **Ln5** check that the clocks are *set* at the same moment. **Ln3** and **Ln4** check that they are *used* at the same moment. So, axiom **Sy3** says that a set C containing clocks that are set at the same time can be replaced in a process p by a single fresh clock z if all the clocks in C are linked in p (i.e., $\mathsf{Lnk}(C,p)$), and z and C are "stochastically

synchronised", that is, the distributions of z and $\max(C)$ are equal.

Axioms **A3′** and **Red** preserve open p-bisimulation, but not symbolic bisimulation. **A3′** gives a weaker notion of idempotency in comparison to axiom **A3** given in the previous section. Notice that the branching process is reduced to only one process that is faster than each of the summands in the original process. This is due to the already mentioned race condition. We recall that $F_{\min\{x,y\}}(t) = F_x(t)+F_y(t)-F_x(t)F_y(t) = 1-(1-F_y(t))(1-F_x(t))$. Axiom **Red** states that clocks which are stochastically redundant —i.e., they are never set to a positive value— cannot affect the timing of a process. Notice that using this axiom together with axiom **Sy1** and **Sy2** allows to eliminate clocks whose support do not contain positive reals.

In the following, we discuss several interesting properties that can be derived from the extended equational theory $ax(\clubsuit^b) + \mathbf{O}$. To begin with, notice that axiom **Sy3** implies α-conversion. In fact, if $F_x = F_y$ and $y \notin fv(p)$, $\{\!|x|\!\}\, p = \{\!|y|\!\}\, \xi_{[y/x]}p$ is a special instance of **Sy3** in which $C = \{x\}$. Moreover, axiom **Sy3** also encodes the elimination of redundant clocks. Suppose $\mathsf{Lnk}(\{x,y\},p)$, and $F_x(t) = 0$ and $F_y(t) = 1$ for some $t \in \mathbb{R}$. Then

$$\{\!|x,y|\!\}\, p \overset{\text{Sy3}}{=} \{\!|z|\!\}\, \xi_{\{z/\{x,y\}\}}p \overset{(\simeq_\alpha)}{=} \{\!|x|\!\}\, \xi_{[x/z]}(\xi_{\{z/x,z/y\}}p) = \{\!|x|\!\}\, \xi_{[x/y]}p$$

provided $z \notin fv(p)$ and $F_z = F_x$.

Using axiom **A3′** we can prove that, given $x \notin fv(p)$, $a;p + \{\!|x|\!\}\, \{x\} \mapsto a;p = a;p$. To calculate it, we assume fresh variables $y, z \notin fv(p)$ such that $F_y(0) = 1$ and $F_z = F_{\min\{x,y\}}$. As a consequence $F_z(0) = 1$. Now, we can proceed as follows.

$$\begin{aligned} a;p + \{\!|x|\!\}\, \{x\} \mapsto a;p &\overset{\text{Sy1}}{=} \{\!|y|\!\}\, a;p + \{\!|x|\!\}\, \{x\} \mapsto a;p \\ &\overset{\text{Red}}{=} \{\!|y|\!\}\, \{y\} \mapsto a;p + \{\!|x|\!\}\, \{x\} \mapsto a;p \\ &\overset{\text{CS3,A3}'}{=} \{\!|z|\!\}\, \{z\} \mapsto a;p \\ &\overset{\text{Red,Sy1}}{=} a;p \end{aligned}$$

In the previous section we argued that $\{\!|x|\!\}\, \{x\} \mapsto a;p + \{\!|x|\!\}\, \{x\} \mapsto a;p \neq \{\!|x|\!\}\, \{x\} \mapsto a;p$. Using axiom **A3′**, we can show easily that indeed this is the case. Assume $F_x = F_y$ and $x, y, z \notin fv(p)$. Then

$$\begin{aligned} \{\!|x|\!\}\, \{x\} \mapsto a;p + \{\!|x|\!\}\, \{x\} \mapsto a;p &\overset{(\simeq_\alpha)}{=} \{\!|x|\!\}\, \{x\} \mapsto a;p + \{\!|y|\!\}\, \xi_{[y/x]}\{x\} \mapsto a;p \\ &\overset{\text{CS3}}{=} \{\!|x,y|\!\}\, (\{x\} \mapsto a;p + \{y\} \mapsto a;p) \\ &\overset{\text{A3}}{=} \{\!|z|\!\}\, \{z\} \mapsto a;p \end{aligned}$$

with $F_z(t) = F_x(t) \cdot F_y(t) = (F_x(t))^2$. Clearly $\{\!|z|\!\}\, \{z\} \mapsto a;p \neq \{\!|x|\!\}\, \{x\} \mapsto a;p$ for every non-trivial random variable x such that $F_x(0) < 1$.

Notice that instead $\{\!|x|\!\}\, (\{x\} \mapsto a;p + \{x\} \mapsto a;p) = \{\!|x|\!\}\, \{x\} \mapsto (a;p + a;p) = \{\!|x|\!\}\, \{x\} \mapsto a;p$.

The next property says that once a clock "is used", then it is not necessary anymore and can be eliminated. It can be seen as a generalisation of axiom **Sy2**.

Property 11.11. Let $p \in \clubsuit^b$, i.e., p is a sequential closed term. Then, for all $C \subseteq \mathcal{C}$ there is a term q such that $C \cap fv(q) = \varnothing$ and $C \mapsto p = C \mapsto q$ can be proved from $ax(\clubsuit^b) + \mathbf{O}$.

Proof. The proof follows by induction on the size of the term p by doing case analysis. In the following we assume that $C \cap fv(q) = C \cap fv(q_1) = C \cap fv(q_2) = \varnothing$ (which will be consequence of the inductive reasoning).

Case $C \mapsto \mathbf{0}$. Trivial.

Case $C \mapsto a; p$.

$$C \mapsto a; p \overset{\text{Sy2}}{=} C \mapsto a; C \mapsto p \overset{\text{ind.}}{=} C \mapsto a; C \mapsto q \overset{\text{Sy2}}{=} C \mapsto a; q$$

Case $C \mapsto C' \mapsto p$.

$$\begin{aligned} C \mapsto C' \mapsto p &\overset{\text{T3}}{=} (C' - C) \mapsto C \mapsto p \overset{\text{ind.}}{=} (C' - C) \mapsto C \mapsto q \\ &\overset{\text{T3}}{=} C \mapsto (C' - C) \mapsto q \end{aligned}$$

Case $C \mapsto \{\!|C'|\!\}\, p$. By α-conversion, we can assume $\{\!|C'|\!\}\, p = \{\!|C^\star|\!\}\, \xi_{[C^\star/C']} p$ with $C^\star \cap C = \varnothing$. Then,

$$\begin{aligned} C \mapsto \{\!|C'|\!\}\, p &= C \mapsto \{\!|C^\star|\!\}\, \xi_{[C^\star/C']} p \overset{\text{T4}}{=} \{\!|C^\star|\!\}\, C \mapsto \xi_{[C^\star/C']} p \\ &\overset{\text{ind.}}{=} \{\!|C^\star|\!\}\, C \mapsto q \overset{\text{T4}}{=} C \mapsto \{\!|C^\star|\!\}\, q \end{aligned}$$

Case $C \mapsto (p_1 + p_2)$.

$$\begin{aligned} C \mapsto (p_1 + p_2) &\overset{\text{T5}}{=} C \mapsto p_1 + C \mapsto p_2 \overset{\text{ind.}}{=} C \mapsto q_1 + C \mapsto q_2 \\ &\overset{\text{T5}}{=} C \mapsto (q_1 + q_2) \end{aligned}$$

Q.E.D.

As we anticipated, the axiom system $ax(\clubsuit^b) + \mathbf{O}$ is sound for $\sim_o$. The proof of the following theorem can be found in Appendix G.3.

Theorem 11.12. *Let p and q be two $\clubsuit$ processes such that their behaviour is α-definable. If $p = q$ is proved by means of equational reasoning using $ax(\clubsuit^b) + \mathbf{O}$ and α-conversion, then $p \sim_o q$.*

11.3 Axioms for the Static Operators

In the previous section we have presented two equational theories for the core of $\clubsuit$. In the following we extend the equational treatment to the rest of the operations. In fact, we give a set of axioms for the static operations that explain how they can be decomposed and eliminated in favour of the basic operations. As a consequence, two extended theories

R1	$\mathbf{0}[f] = \mathbf{0}$	**R4**	$(\{\!\{C\}\!\}\, p)[f] = \{\!\{C\}\!\}\, (p[f])$
R2	$(a; p)[f] = f(a); (p[f])$	**R5**	$(p + q)[f] = p[f] + q[f]$
R3	$(C \mapsto p)[f] = C \mapsto (p[f])$		

Mrg1	$(\{\!\{C\}\!\}\, p) \,\|\|_A\, q = \{\!\{C\}\!\}\, (p \,\|\|_A\, q)$	if $C \cap (\kappa(q) \cup fv(q)) = \varnothing$
Mrg2	$p \,\|\|_A\, (\{\!\{C\}\!\}\, q) = \{\!\{C\}\!\}\, (p \,\|\|_A\, q)$	if $C \cap (\kappa(p) \cup fv(p)) = \varnothing$
Mrg3	$p \,\|\|_A\, q = p \,\llcorner\!\llcorner_A\, q + q \,\llcorner\!\llcorner_A\, p + p \,\|_A\, q$	if $\mathsf{TR}(p) \wedge \mathsf{TR}(q)$
LM1	$\mathbf{0} \,\llcorner\!\llcorner_A\, q = \mathbf{0}$	if $\mathsf{TR}(q)$
LM2	$a; p \,\llcorner\!\llcorner_A\, q = \mathbf{0}$	if $\mathsf{TR}(q) \wedge a \in A$
LM3	$a; p \,\llcorner\!\llcorner_A\, q = a; (p \,\|\|_A\, q)$	if $\mathsf{TR}(q) \wedge a \notin A$
LM4	$(C \mapsto p) \,\llcorner\!\llcorner_A\, q = C \mapsto (p \,\llcorner\!\llcorner_A\, q)$	
LM5	$(\{\!\{C\}\!\}\, p) \,\llcorner\!\llcorner_A\, q = \{\!\{C\}\!\}\, (p \,\llcorner\!\llcorner_A\, q)$	if $C \cap (\kappa(q) \cup fv(q)) = \varnothing$
LM6	$p \,\llcorner\!\llcorner_A\, (\{\!\{C\}\!\}\, q) = \{\!\{C\}\!\}\, (p \,\llcorner\!\llcorner_A\, q)$	if $C \cap (\kappa(p) \cup fv(p)) = \varnothing$
LM7	$(p + q) \,\llcorner\!\llcorner_A\, r = (p \,\llcorner\!\llcorner_A\, r) + (q \,\llcorner\!\llcorner_A\, r)$	if $\mathsf{TR}(r)$
CM0	$p \,\|_A\, q = q \,\|_A\, p$	
CM1	$\mathbf{0} \,\|_A\, \mathbf{0} = \mathbf{0}$	
CM2	$\mathbf{0} \,\|_A\, a; q = \mathbf{0}$	
CM3	$a; p \,\|_A\, a; q = a; (p \,\|\|_A\, q)$	if $a \in A$
CM4	$a; p \,\|_A\, b; q = \mathbf{0}$	if $a \notin A$
CM5	$(C \mapsto p) \,\|_A\, q = C \mapsto (p \,\|_A\, q)$	
CM6	$(\{\!\{C\}\!\}\, p) \,\|_A\, q = \{\!\{C\}\!\}\, (p \,\|_A\, q)$	if $C \cap (\kappa(q) \cup fv(q)) = \varnothing$
CM7	$(p + q) \,\|_A\, r = (p \,\|_A\, r) + (q \,\|_A\, r)$	if $\mathsf{TR}(r)$

TR1 $\mathsf{TR}(\mathbf{0})$ **TR2** $\mathsf{TR}(a; p)$ **TR3** $\dfrac{\mathsf{TR}(p)}{\mathsf{TR}(C \mapsto p)}$ **TR4** $\dfrac{\mathsf{TR}(p) \quad \mathsf{TR}(q)}{\mathsf{TR}(p + q)}$

Table 11.3: Axioms for the static operators

are obtained: one which is sound and complete for structural bisimulation, and the other which is sound for open p-bisimulation.

The new axioms are given in Table 11.3. Axioms **R1**–**R6** define the renaming operation in terms of the basic operations. Notice that if we consider the axioms as rewrite rules from left to right, the renaming operation is "pushed" inside the term while appropriately renaming actions according to f.

The rest of the axioms concern the different parallel compositions. Axioms **Mrg1** and **Mrg2** move clock settings out of the scope of the parallel composition. This is necessary because when expanding parallel composition in terms of summations, we would like to avoid the duplication of clocks. Duplicating clocks would transform processes without conflict of variables into (semantically different!) processes with conflict of variables. **Mrg3** decomposes the parallel composition in terms of the left merge and the communication merge provided that no clock setting is wrongly duplicated. Thus the requirement that both p and q satisfy the predicate TR. Predicate TR, defined by the rules **TR1**–**TR4**, checks that a given process does not reset any clock before performing any initial action. In fact, TR encodes in a way the function that $\overline{\mathsf{ck}}$ has at the semantic level. Observe that if $\mathsf{TR}(p)$, then $\kappa(p) = \varnothing$ and hence $p \equiv \overline{\mathsf{ck}}(p)$. We are not interested in having $\overline{\mathsf{ck}}$ explicitly in our signature since it does not preserve $\sim_o$. Notice moreover, that a pre-basic term $p \in B'$ satisfies $\mathsf{TR}(p)$. (**TR** stands for "trivially rooted process".)

The axiomatisation of the parallel composition is completed with axioms **LM1**–**LM7** and **CM0**–**CM7**. Axioms **LM1**–**LM7** define the left merge depending on the structure not only of the left operand, but also of the right one. In particular, notice in **LM3** that, in order to move a prefix in the left operand out of the scope of the left merge, it is necessary that the right operand does not reset any clock (predicate $\mathsf{TR}(q)$). Finally, axioms **CM0**–**CM7** define the communication merge depending on the form of both operands.

Definition 11.13. The equational theory for $\hat{\Delta}$ is defined as follows. The signature $sig(\hat{\Delta})$ contains the constant $\mathbf{0}$, the unary operations $a;\_$, $C \mapsto \_$, $\{\!|C|\!\}\_$, and $\_[f]$, and the binary operations $+$, $||_A$, $\between_A$, and $|_A$. The axiom system $ax(\hat{\Delta})$ is given by axioms in Table 11.1 and Table 11.3. We denote by $ax(\hat{\Delta}) + \mathbf{O}$ the extension of $ax(\hat{\Delta})$ with the set of axioms enumerated in Table 11.2. □

In the next theorem we state that the axiomatisations $ax(\hat{\Delta})$ and $ax(\hat{\Delta}) + \mathbf{O}$ are sound for structural bisimulation and open p-bisimulation respectively.

Theorem 11.14. *The following soundness results hold.*

1. *Let p and q be $\hat{\Delta}$ terms such that their behaviours are definable. If $p = q$ is proved by means of equational reasoning using $ax(\hat{\Delta})$ then $p \sim_s q$.*

2. *Let p and q be $\hat{\Delta}$ terms such that their behaviours are α-definable. If $p = q$ is proved by means of equational reasoning using $ax(\hat{\Delta}) + \mathbf{O}$ and α-conversion, then $p \sim_o q$.*

Proof. In both cases, it is only necessary to check that axioms in Table 11.3 preserve $\sim_s$. For the first item, terms in $\hat{\Delta}^{\mathsf{cf}}$ should be considered, and for the second one, terms in $\hat{\Delta}^{\alpha}$.

In both cases it is straightforward to check that for every axiom $p = q$ in Table 11.3, the relation $R \stackrel{\text{def}}{=} \{\langle p, q\rangle\}^{\mathsf{s}}$ is a structural bisimulation up to $\sim_s$. For axioms **Mrg3** and **CM0**, it is necessary to know in addition that $p||_A q \sim_s q||_A p$ which can be easily verified. *Q.E.D.*

In addition, it is not difficult to check that the new axioms preserve definability. That is, Proposition 11.4 also holds for axioms in Table 11.1.

In the following we show completeness of $ax(\spadesuit)$. To do so we first prove that the static operations can be eliminated in favour of the basic operations, that is, for every closed term in $\spadesuit$ there is a term in $\spadesuit^b$ that can be proved equal using the axioms. From now on we denote by $\spadesuit^c$ the set of all closed and definable $\spadesuit$ terms, i.e., the subset of $\spadesuit^{\mathsf{cf}}$ with terms not containing process variables.

Theorem 11.15. *For every term p in $\spadesuit^c$, there is a $q \in \spadesuit^b$ (not containing $||_A$, $\between_A$, $|_A$, and $\_[f]$) such that $p = q$ can be proved using $ax(\spadesuit)$.*

Proof sketch. Consider axioms in Table 11.3 from left to right as rewrite rules modulo axioms in Table 11.1 and **CM1**. It is simple to prove that the normal form is a term $q' \in \spadesuit^b$. *Q.E.D.*

Because of Theorem 11.10, completeness of $ax(\spadesuit)$ follows as a corollary.

Theorem 11.16. *The axiom system $ax(\spadesuit)$ is complete for the algebra $(\spadesuit^c, sig(\spadesuit), \sim_s)$.*

As we mentioned, one of the reasons why many approaches to stochastic process algebras stick to only exponential distributions is that general distributions do not preserve Milner's expansion law in their models. In other cases, combining interleaving-based process algebras with general distributions lead to infinite and cumbersome models that are very difficult to deal with. Instead, in our case, the expansion law is inherent to the model and the way in which parallel composition is defined. It can be smoothly derived from the axioms as stated by the following theorem.

Theorem 11.17. *Let p and q be $\spadesuit$ terms such that $p = \{\!\{C\}\!\}\, p'$ and $q = \{\!\{C'\}\!\}\, q'$ with $p' = \sum_i C_i \mapsto a_i; p_i$ and $q' = \sum_j C'_j \mapsto b_j; q_j$. Suppose the behaviour of $p \,||_A\, q$ is α-definable and, in particular, locally definable. From the axiom system $ax(\spadesuit)$ we can derive*

$$\begin{aligned} p \,||_A\, q \;=\; \{\!\{C \cup C'\}\!\} \Big(& \sum_{a_i \notin A} C_i \mapsto a_i; (p_i \,||_A\, q') \\ & + \sum_{b_j \notin A} C'_j \mapsto b_j; (p' \,||_A\, q_j) \\ & + \sum_{a_i = b_j \in A} (C_i \cup C'_j) \mapsto a_i; (p_i \,||_A\, q_j) \Big) \end{aligned}$$

Moreover, if the behaviour of $p[f]$ is α-definable, and also locally definable, the following equality can also be derived,

$$p[f] \;=\; \{\!\{C\}\!\} \;\big(\textstyle\sum_i C_i \mapsto f(a_i); (p_i[f]) \big)$$

We mention that ♤ also satisfies the recursion principles given in Definition 6.17. The reader is referred to page 80 to learn about this. In the following examples we use the fact that ♤ satisfy RSP and RDP$^-$.

Example 11.18. (... or more than example, an exercise.) We invite the reader to show that the switch as modelled in Example 10.2 is equal to the recursive specification that represents the stochastic automaton of the switch in Example 10.20. The way to prove it follows closely the exercise we did for the timed switch using ♡ (see Example 6.16). □

Example 11.19. We use ♤ to obtain a hierarchical representation of the queueing system of Example 9.24. Recall that a $G/G/1/\infty$ queue describes a system where jobs arrive and wait in a queue of infinite capacity until they are executed by a single server. We assume that jobs arrive with an interarrival time that is determined by some distribution F_x, and that the execution time of a job by the server is determined by a distribution function F_y.

The queue can be described by the following recursive specification:

$$\begin{aligned} Queue_0 &= in; Queue_1 \\ Queue_{n+1} &= in; Queue_{n+2} + out; Queue_n \qquad (n \geq 0) \end{aligned}$$

The subindex n in the process variable $Queue_n$ indicates the number of jobs in the queue. The process that determines the arrival of jobs is like the one we use for the switch:

$$Arrival = \{\!|x|\!\}\,\{x\} \mapsto in; Arrival$$

The server takes a job out of the queue and processes it. The execution time is controlled by a clock y with distribution F_y.

$$Server = out; \{\!|y|\!\}\,\{y\} \mapsto done; Server$$

The complete system is described by the process

$$QSystem = (Arrival \,||_\varnothing\, Server) \,||_Q\, Queue_0$$

where $Q = \{in, out\}$. Notice that process *Arrival* synchronises with the queue on the action *in*, while the *Server* does it on action *out*.

In the following, we proceed to reduce the process *QSystem* to a simple recursive expression. We will use the following abbreviations:

$$Arrival' \equiv \{x\} \mapsto in; Arrival \qquad \text{and} \qquad Server' \equiv \{y\} \mapsto done; Server$$

Thus $Arrival = \{\!|x|\!\}\, Arrival'$ and $Server = out; \{\!|y|\!\}\, Server'$. Now, we calculate

$$QSystem = \{\!|x|\!\}\,((Arrival' \,||_\varnothing\, Server) \,||_Q\, Queue_0)$$

where

$$\begin{aligned} &((Arrival' \,||_\varnothing\, Server) \,||_Q\, Queue_0) \\ &\quad = (\{x\} \mapsto in; (Arrival \,||_\varnothing\, Server) \,||_Q\, in; Queue_1) \\ &\quad = \{x\} \mapsto in; ((Arrival \,||_\varnothing\, Server) \,||_Q\, Queue_1) \\ &\quad = \{x\} \mapsto in; \{\!|x|\!\}\,((Arrival' \,||_\varnothing\, Server) \,||_Q\, Queue_1) \end{aligned}$$

For the case $n \geq 0$, we proceed:

$$\begin{aligned}
&((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_{n+1}) \\
&\quad = (\{x\} \mapsto in; (Arrival \,||_{\varnothing}\, Server) + out; (Arrival' \,||_{\varnothing}\, \{\!|y|\!\}\, Server')) \\
&\qquad ||_{Q}\, (in; Queue_{n+2} + out; Queue_{n}) \\
&\quad = \{x\} \mapsto in; ((\{\!|x|\!\}\, Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_{n+2}) \\
&\qquad + out; ((Arrival' \,||_{\varnothing}\, \{\!|y|\!\}\, Server') \,||_{Q}\, Queue_{n}) \\
&\quad = \{x\} \mapsto in; \{\!|x|\!\}\, ((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_{n+2}) \\
&\qquad + out; \{\!|y|\!\}\, ((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{n})
\end{aligned}$$

For process $((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{n})$, we also consider two cases. For $n = 0$,

$$\begin{aligned}
&((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{0}) \\
&\quad = (\{x\} \mapsto in; (Arrival \,||_{\varnothing}\, Server') + \{y\} \mapsto done; (Arrival' \,||_{\varnothing}\, Server)) \\
&\qquad ||_{Q}\, in; Queue_{1} \\
&\quad = \{x\} \mapsto in; ((\{\!|x|\!\}\, Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{1}) \\
&\qquad + \{y\} \mapsto done; ((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, in; Queue_{1}) \\
&\quad = \{x\} \mapsto in; \{\!|x|\!\}\, ((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{1}) \\
&\qquad + \{y\} \mapsto done; ((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_{0})
\end{aligned}$$

For $n \geq 0$,

$$\begin{aligned}
&((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{n+1}) \\
&\quad = (\{x\} \mapsto in; (Arrival \,||_{\varnothing}\, Server') + \{y\} \mapsto done; (Arrival' \,||_{\varnothing}\, Server)) \\
&\qquad ||_{Q}\, (in; Queue_{n+2} + out; Queue_{n}) \\
&\quad = \{x\} \mapsto in; ((\{\!|x|\!\}\, Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{n+2}) \\
&\qquad + \{y\} \mapsto done; ((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, (in; Queue_{n+2} + out; Queue_{n}) \\
&\quad = \{x\} \mapsto in; \{\!|x|\!\}\, ((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{n+2}) \\
&\qquad + \{y\} \mapsto done; ((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_{n+1})
\end{aligned}$$

Notice that in both cases we conclude that

$$\begin{aligned}
&((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{n}) \\
&\quad = \{x\} \mapsto in; \{\!|x|\!\}\, ((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{n+1}) \\
&\qquad + \{y\} \mapsto done; ((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_{n})
\end{aligned}$$

Summarising, we have:

$$\begin{aligned}
QSystem &= \{\!|x|\!\}\,((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_0)\\
((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_0)&\\
&= \{x\} \mapsto in; \{\!|x|\!\}\,((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_1)\\
((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_{n+1}) &\qquad (n \geq 0)\\
&= \{x\} \mapsto in; \{\!|x|\!\}\,((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_{n+2})\\
&\quad + out; \{\!|y|\!\}\,((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_n)\\
((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_n) &\qquad (n \geq 0)\\
&= \{x\} \mapsto in; \{\!|x|\!\}\,((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_{n+1})\\
&\quad + \{y\} \mapsto done; ((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_n)
\end{aligned}$$

By RSP, we can conclude that

$$\begin{aligned}
QSystem &= QSpec\\
((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_0) &= QWait_0\\
((Arrival' \,||_{\varnothing}\, Server) \,||_{Q}\, Queue_{n+1}) &= QWait_{n+1}\\
((Arrival' \,||_{\varnothing}\, Server') \,||_{Q}\, Queue_n) &= QServ_n
\end{aligned}$$

where

$$\begin{aligned}
QSpec &= \{\!|x|\!\}\,QWait_0\\
QWait_0 &= \{x\} \mapsto in; \{\!|x|\!\}\,QWait_1\\
QWait_{n+1} &= \{x\} \mapsto in; \{\!|x|\!\}\,QWait_{n+2} + out; \{\!|y|\!\}\,QServ_n \qquad (n \geq 0)\\
QServ_n &= \{x\} \mapsto in; \{\!|x|\!\}\,QServ_{n+1} + \{y\} \mapsto done; QWait_n \qquad (n \geq 0)
\end{aligned}$$

Assuming that $F_x(0) = 0$, in addition, we have that this recursive specification shows the following *closed* behaviour,

$$\begin{aligned}
CQSpec &= \{\!|x|\!\}\,CQWait_0\\
CQWait_0 &= \{x\} \mapsto in; \{\!|x|\!\}\,CQWait_1\\
CQWait_{n+1} &= out; \{\!|y|\!\}\,CQServ_n \qquad (n \geq 0)\\
CQServ_n &= \{x\} \mapsto in; \{\!|x|\!\}\,CQServ_{n+1} + \{y\} \mapsto done; CQWait_n \qquad (n \geq 0)
\end{aligned}$$

that is, $QSpec \sim_c CQSpec$. In Figure 11.1 we depict the stochastic automaton defined by the specification $CQSpec$. □

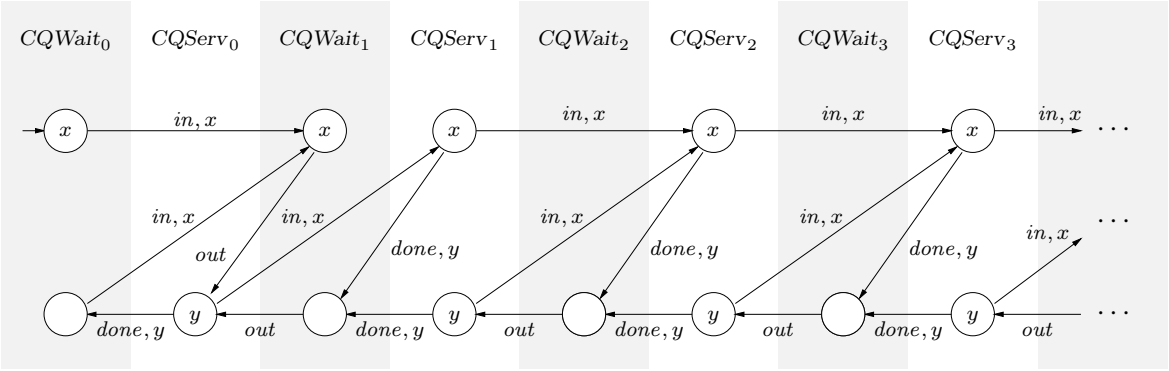


Figure 11.1: The stochastic automaton of *CQSpec*

11.4 Conservative Extension

$\clubsuit$ extends the syntax of $\spadesuit$ with some new operators. Moreover, both $(\clubsuit^c, sig(\clubsuit^c), \sim_s)$ and $(\clubsuit^c, sig(\clubsuit^c), \sim_o)$ are model conservative extensions of $(\spadesuit^c, sig(\spadesuit^c), \sim)$. The proof can be sketched as follows. First, we notice that for all $p \in \spadesuit^c$, $\kappa(p) = \varnothing$ and $p \xrightarrow{a,\varnothing} p' \iff p \xrightarrow{a} p'$, just as we observed in Remark 9.3. Second, $\sim_s$, $\sim_o$, and $\sim$ turn to be equivalent for the subset $\spadesuit^c \subseteq \clubsuit^c$, as it was indicated in Remark 9.22. Now, model conservative extension follows by (D'Argenio and Verhoef, 1997, see also Section 2.3, page 25).

However, $\clubsuit$ is not an (equational) conservative extension of $\spadesuit$. The problem arises that axiom $p + p = p \in ax(\spadesuit)$ cannot be proved from $ax(\clubsuit)$ in general. Instead, $\clubsuit$ is a conservative extension of a slightly weaker axiomatisation of $\spadesuit$.

Let $ax(\spadesuit)^-$ be the same set of axioms as $ax(\spadesuit)$ (see Definition 2.15) except that the idempotency axiom $p + p = p$ is replaced by axiom $a; p + a; p = a; p$. $ax(\spadesuit)^-$ can still be proved sound and complete for $(\spadesuit^c, sig(\spadesuit^c), \sim)$. Now, it is indeed the case that every axiom in $ax(\spadesuit)^-$ can be proven in the equational theory $ax(\clubsuit)$. As a consequence, by (D'Argenio and Verhoef, 1997), the following theorem follows.

Theorem 11.20. *Both the equational theories $ax(\clubsuit)$ and $ax(\clubsuit) + \mathbf{O}$ as given in Definition 11.13 are conservative extensions of the equational theory $ax(\spadesuit)^-$ defined above.*

Chapter 12

Analysis of ♤ Specifications

A system specification in ♤ contains functional and quantitative aspects. In order to obtain a concrete idea about the impact of the stochastic delays in the specification on measures of interest like throughput, or response time, the specification is analysed. Since arbitrary distributions are allowed, analytical or numerical techniques are applicable only in restricted cases, e.g. when all delays are governed by negative exponential distributions. We take a more general approach by using *simulation*, in particular discrete-event simulation, where in contrast to continuous-time simulation techniques, state changes take place at discrete points in time (those points in which an action takes place), although time itself is taken to be continuous.

Apart from the fact that discrete-event simulation allows a wider spectrum of probability distributions, an inherent characteristic of this technique is that the semantic object can be generated "on-the-fly". In fact, only the current simulated state is relevant. In this sense, simulation techniques do not suffer from the main problem of analytical and numerical methods: the state space explosion. To carry out simulation, simulations runs (also called sample paths) are generated, and on the basis of these runs, data are gathered and analysed to determine the desired measure of interest in an estimative fashion. The main problem we must address is the resolution of possible non-determinism in a stochastic automaton. Although it is widely recognised that non-determinism is of significant importance in a step-wise design methodology for the purpose of under-specification, it needs to be resolved before simulations can be carried out.

The advantage of stochastic process algebras is its power of combining the qualitative and quantitative aspects of the analysis of systems in a simple and modular framework. In addition to discrete-event simulation we also discuss a classical analysis technique for functional correctness: reachability analysis. Reachability analysis is the key technique for proving safety properties, which are those properties that state that "something bad can never happen". A typical reachability property is the absence of a deadlock, a reachable state from which no further progress can be made. In order to check such properties for stochastic automata, and thus ♤ terms, the underlying semantics in terms of probabilistic transition systems needs to be examined. However, even for finite terms, these transition systems are highly infinite due to the continuous nature of distribution functions. We

therefore investigate the usage of *symbolic* reachability analysis. This means that we want to check reachability properties without having the need for building and examining the infinite underlying probabilistic transition system, but instead check reachability at the level of stochastic automata. In fact, it turns out that such analysis can be viewed as being carried out on ordinary labelled transition systems. Consequently, well-known techniques can be applied to reduce the complexity of reachability analysis. We investigate this for the open and closed interpretation of stochastic automata and confine ourselves to finite stochastic automata.

In this chapter we give the foundational framework for discrete-event simulation and reachability analysis, and report on prototypical algorithms that implement such results. We give several examples mostly concentrating on performance and dependability analysis.

12.1 Runs and Schedulers

Basically, a run of a rooted probabilistic transition system is a path obtained by traversing it starting from its initial state.

Definition 12.1. Let $PTS = (\Sigma, \Sigma', \mathcal{L}, T, \longrightarrow)$ be a probabilistic transition system. Take $\sigma_0 \in \Sigma$ to be the initial state. A *run* of (PTS, σ_0) is a (finite or infinite) sequence $\rho \equiv \sigma_0\sigma'_0\ell_1\sigma_1\sigma'_1\ell_2\sigma_2\sigma'_2\ldots$ such that, for all $i \geq 0$,

1. σ'_i is in the support set of the probability measure of $T(\sigma_i)$,
2. $\sigma'_i \xrightarrow{\ell_{i+1}} \sigma_{i+1}$, and
3. if ρ is finite, it ends in a non-deterministic state (i.e., some $\sigma'_i \in \Sigma'$).

We denote by $runs(PTS, \sigma_0)$ the set of all runs of (PTS, σ_0). If ρ is a finite run, $last(\rho)$ denotes the last (non-deterministic) state in the execution ρ. □

The first constraint states that probabilistic steps should be probable ones. That is, the probability of every open set containing a non-deterministic state that the run pass through should be strictly larger than 0 (see definition of support set in Appendix D). In other words, the probability mass (or probability density) of every probabilistic step in a run, should be strictly positive. In a more general setting it may be interesting to also consider "improbable" runs. In our case, we want the simulator to generate runs with positive probability mass only. The second constraint defines the non-deterministic steps in the run, and the last constraint states that a run should end in a non-deterministic state. Notice that as a consequence runs cannot be empty.

In Figure 12.1, we give an example of a probabilistic transition system with a run as we defined it and an invalid run. The grey triangles indicates the support set of the probabilistic transitions (i.e., the "probable" part of the transition). We have highlighted two different paths. The lighter one gives an example of a valid run: $\sigma_0\,\sigma'_0\,a(2)\,\sigma_1\,\sigma'_1\,c(4)\,\sigma_2$. Notice that σ'_0 and σ'_1 fall inside the support set of the probabilistic transitions $T(\sigma_0)$ and

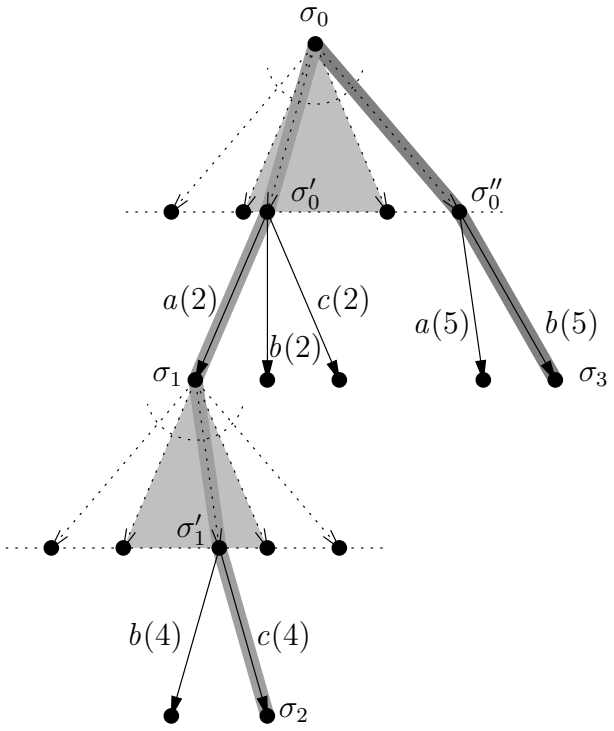


Figure 12.1: Example of valid and invalid run

$T(\sigma_1)$, respectively. Compare with the darker path, which violates the first requirement of our definition of run: we can chose a small enough open set I containing the state σ''_0 such that the probability $P(I) = 0$.

Non-determinism is useful for under-specifying the frequency with which an alternative will take place. Often this information is not available in the early steps of the design, or is deliberately left unspecified. To still study the performance of such system specifications, the idea is to impose an additional machinery —called a *scheduler* or *adversary* (Vardi, 1985; Segala and Lynch, 1995)— on top of the system. If the system has reached a state in which a choice occurs between several non-deterministic possibilities, the adversary will choose one such possibility. One thus considers a system (in our case a probabilistic transition system *PTS*), that contains a certain implementation freedom, in the context of an adversary $\mathcal{A}$. *PTS* can be viewed as the system specification, and $\mathcal{A}$ as the representation of the architecture on which the system is realised. The pair $(PTS, \mathcal{A})$ is thus the entire system under consideration. As a result, the simulation data for *PTS* that is obtained should be considered with respect to the adversary $\mathcal{A}$.

More precisely, an adversary is a function that schedules the next non-deterministic transition based on the past history of the system. We consider probabilistic adversaries like in (Segala and Lynch, 1995), although our definition is adapted to probabilistic transition systems following the style of (Hansson, 1994).

Definition 12.2. An *adversary* is a partial function $\mathcal{A}$ that given a finite run ρ, it returns a probability distribution whose domain (viz., the sample space) contains non-deterministic transitions outgoing from the the last state of ρ. Formally, an *adversary* is a partial function $\mathcal{A} : \mathit{runs}(PTS, \sigma_0) \to (\,(\longrightarrow) \to [0,1])$ such that for all $\rho \in \mathit{runs}(PTS, \sigma_0)$, $\mathcal{A}(\rho) \stackrel{\text{def}}{=} P$ for some discrete probability measure P on the (discrete) σ-algebra $\wp(\Omega_\rho)$, where

$$\varnothing \neq \Omega_\rho \subseteq \left\{ \sigma' \stackrel{\ell}{\longrightarrow} \sigma \mid \mathit{last}(\rho) = \sigma' \right\},$$

is a (countable) sample space. □

This notion can be lifted to stochastic automata as follows. A run of a stochastic automaton SA with initial location s_0, is a run of its underlying (closed) semantics $(PTS_c(SA), (s_0, v_0))$ where v_0 is the appointed initial valuation. Notice that for a given non-deterministic state $[s, v]$, the next transition is fully determined by v and the outgoing edges from s. This allows us to define:

Definition 12.3. An *adversary* for the stochastic automaton SA is a partial function $\mathcal{A} : \mathit{runs}(PTS_c(SA), (s_0, v_0)) \to (\,(\longrightarrow) \to [0,1])$ such that for all $\rho \in \mathit{runs}(PTS_c(SA), (s_0, v_0))$, $\mathcal{A}(\rho) \stackrel{\text{def}}{=} P$ for some discrete probability measure P on the (discrete) σ-algebra $\wp(\Omega'_\rho)$, where

$$\varnothing \neq \Omega'_\rho \subseteq \left\{ s \xrightarrow{a,C} s' \mid \mathit{last}(\rho) = [s, v] \wedge s \xrightarrow{a,C} s' \text{ is enabled in } v \right\},$$

is a (countable) sample space. By 'enabled', we mean that $s \xrightarrow{a,C} s'$ produces an outgoing transition in the state $[s, v]$ (see Definition 9.5, in particular rule **Closed**). □

12.2 Discrete Event Simulation

The simulation algorithm is implemented as a so-called *variable time-advance procedure* (Shedler, 1993). A variable time-advance procedure is a simple algorithm in which simulated time goes forward to the next time at which a transition is triggered and the intervening time is skipped.

In our setting, the simulation algorithm requires the following inputs:

(i) a recursive ⌂ specification E representing the system specification,

(ii) an adversary $\mathcal{A}$ that resolves possible non-determinism, and

(iii) the initial process p_0.

It is assumed that the initial valuation v_0 equals 0 for all clocks. (For a process p_0 that does not contain free clock variables this poses no restriction.) The detailed structure of the simulation algorithm is depicted in Figure 12.2.

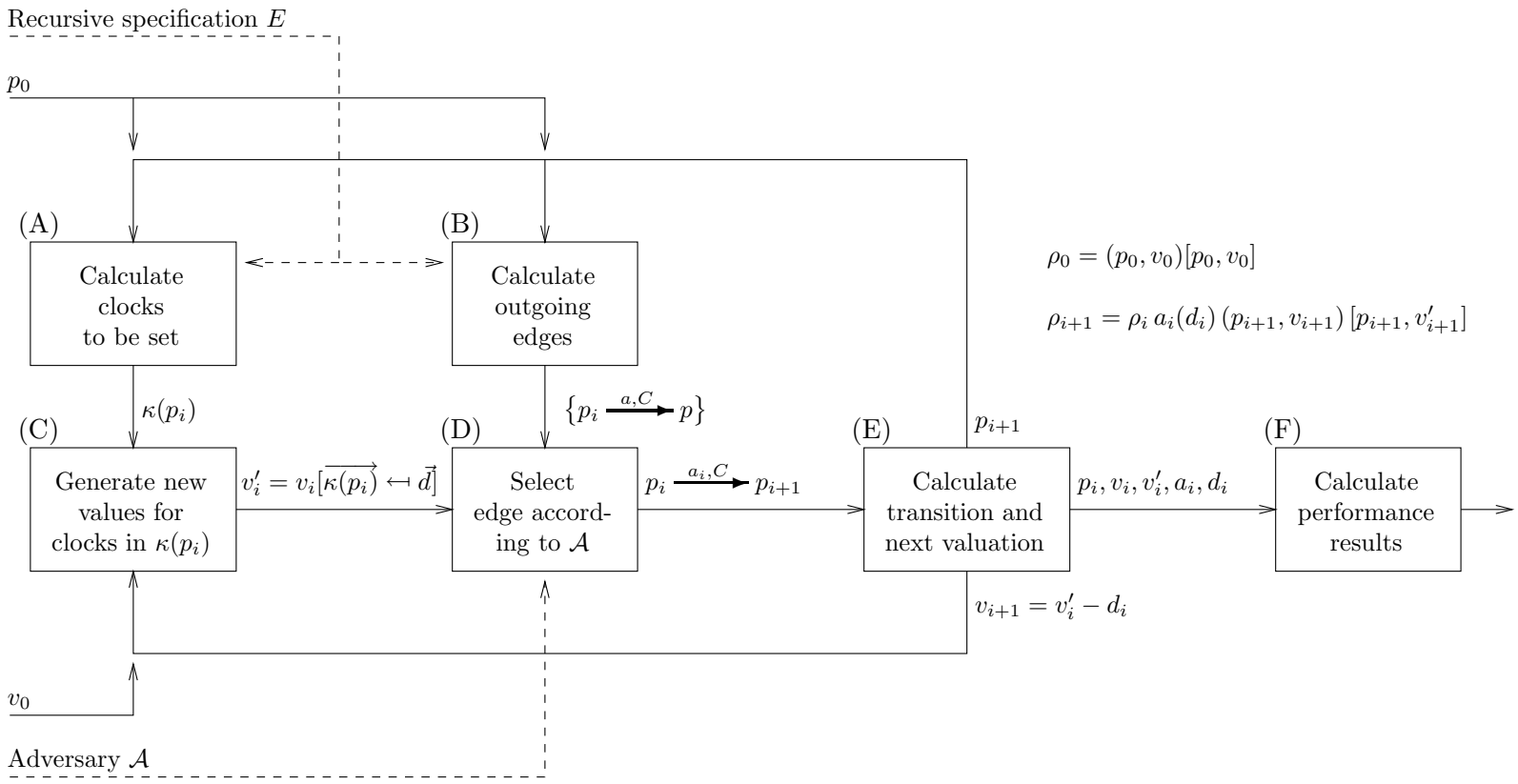


Figure 12.2: Schema of the simulation algorithm

Since in our semantics a location corresponds to a term, simulation can be carried out on the basis of expressions rather than using their semantic representation. This means that the stochastic automaton is not entirely generated a priori but only the parts that are required to choose the next step. The simulation starts in (p_0, v_0), the initial state of the (rooted) probabilistic transition system defined by the closed behaviour of (SA, p_0). Once started, the stochastic automaton is constructed in an on-the-fly fashion on the basis of the current term p_i (i.e. location) and the input specification E. From term p_i the set of clocks $\kappa(p_i)$ to be set is determined (by module (A)) and the set of possible next edges is computed according to the inference rules of Table 10.3 (this is done by module (B)).

Subsequently, the next valuation is calculated. It is sufficient for this purpose to keep track off the last valuation v_i only. To each clock x in $\kappa(p_i)$ a random value according to the distribution function F_x is assigned, while the other clocks remain unchanged (this is done by module (C)). This step corresponds to the rule **Prob** in Definition 9.5. In Figure 12.2 this new valuation is denoted by $v_i' = v_i[\overrightarrow{\kappa(p_i)} \leftarrow \vec{d}]$. There, the different numbers in $\vec{d}$ are randomly selected values which should be generated using random number generation techniques (see e.g. Knuth, 1981; Jain, 1991; Cassandras, 1993).

Given the new valuation and the set of possible edges, the adversary $\mathcal{A}$ is used in order to select the next edge to be executed. This is done by module (D). From the set of possible edges (calculated by module (B)), the subset of enabled edges is selected. This step corresponds to rule **Closed** in Definition 9.5. From this set of enabled edges (if any), an edge is selected by the adversary $\mathcal{A}$ in a probabilistic fashion.

The impact of traversing the selected edges is carried out subsequently by module (E). This comprises the calculation of the next step according to rule **Closed**. More precisely, it determines the executed action a_i and its timing d_i plus the next location p_{i+1} and the next valuation $v_{i+1} = v_i' - d_i$ which are used to calculate the next step in the simulation run.

Finally, the measure of interest (like throughput, utilisation or response time) needs to be distilled from the generated simulation run. For that purpose, information is gathered from the run and analysed. This is done by module (F). This component is user-driven, since the calculations to be performed are determined by the user's needs.

We have implemented a prototype of the simulation algorithm using the functional language Haskell 1.4 (Peterson and Hammond, 1997). In this implementation we have confined ourselves to guarded recursive specifications. Unguarded recursion may yield an infinite set of outgoing edges or an infinite set of clock settings. In this way, modules (D) and (E) would never finish their computations. In the current implementation we only deal with adversaries that are history independent in the following sense: adversary $\mathcal{A}$ should satisfy the property that $last(\rho) = last(\rho')$ implies $\mathcal{A}(\rho) = \mathcal{A}(\rho')$. Accordingly, in order to select an enabled next edge probabilistically, only the current location is of importance, but not how we reached that location. In this way, we are not required to store a complete run. This assumption also reduces the time complexity of the simulation algorithm. Notice that modules (C) and (D) require the use of "randomness". To that purpose, we use a random number generator which is a multiplicative linear congruential

generator with modulus $m = 2^{31} - 1$ and multiplier $a = 16807$ calculated by Schrage's algorithm (see Knuth, 1981; Jain, 1991, for more information).

12.3 Reachability Analysis

In the following we discuss the foundations that justify that reachability analysis techniques can be applied directly at the stochastic automata level in order to ensure the preservation of safety properties. We define the notions of reachability in a probabilistic transition system and symbolic reachability in a stochastic automaton and investigate their correspondence.

Definition 12.4. Let PTS be a probabilistic transition system and σ_0 its designated initial state. We say that a nondeterministic state σ' is *reachable* if there exists a run $\rho \in runs(PTS, \sigma_0)$ such that $last(\rho) = \sigma'$. We denote by $reach(PTS, \sigma_0)$ the set of all states in PTS that are reachable from the initial state σ_0. □

Definition 12.5. Let $SA = (\mathcal{S}, \mathcal{C}, \mathcal{A}, \longrightarrow, \kappa, F)$ be a stochastic automaton and suppose $s_0 \in \mathcal{S}$ is the initial state. A *symbolic run of SA starting from* s_0 is a finite sequence $s_0 a_1 C_1 s_1 \ldots s_{n-1} a_n C_n s_n$, $n \geq 0$, such that, for all $0 < i \leq n$, $s_{i-1} \xrightarrow{a_i, C_i} s_i$.

We say that a location s is *symbolically reachable* if there exists a symbolic run that ends in s. We denote by $reach(SA, s_0)$ the set of all locations in SA that are symbolically reachable from s_0. □

The following lemma states a connection between the edges of a stochastic automaton and the actual transitions in its behaviour as open system.

Lemma 12.6. *Let SA be a stochastic automaton and let $PTS_o(SA)$ be its open behaviour. Then, for all valuations $v \in \mathbf{V}$,*

$$s \xrightarrow{a,C} s' \iff \exists d \in \mathbb{R}_{\geq 0}.\ [s, v] \xrightarrow{a(d)} (s', v - d)\,.$$

Stated in words, given an arbitrary valuation v, there is an edge from location s to s' if and only if there is a transition in the open behaviour from the non-deterministic state $[s, v]$ to the (probabilistic) state $(s', v-d)$. The "$\Leftarrow$" part of the lemma is rather straightforward: there can only be a transition in the open semantics if there is a corresponding edge in the stochastic automaton. The "$\Rightarrow$" part holds, since in case $s \xrightarrow{a,C} s'$ for a given valuation v, there always exists a sufficiently large d such that $v - d$ is at most 0 for all clocks in C (e.g. let d be the maximum clock value in v of all clocks in C). Thus, the constraint "$\forall x \in C.\ (v-d)(x) \leq 0$" holds. Consequently there is a transition in the open behaviour, due to rule **Open** in Definition 9.7.

Given the fact that the probabilistic transition T is a function, as a consequence of Lemma 12.6, every symbolic run has a corresponding run in its open interpretation, and vice versa. So, the following theorem follows as a corollary.

Theorem 12.7. *Given a location s of a stochastic automaton SA, for every initial location s_0 and initial valuation v_0, $s \in reach(SA, s_0)$ if and only if there exists $v \in \mathbf{V}$ such that $[s, v] \in reach(PTS_o(SA), (s_0, v_0))$.*

Recall that, as opposed to the open behaviour, in the closed behaviour an edge can only be taken if there is no earlier point in time at which the current location can be left (maximal progress). As a consequence, certain edges present in the stochastic automaton do not necessarily result in a transition in their closed behaviour, since there may exist an alternative edge that is "faster" and thus it will be taken instead. Technically speaking, the "$\Rightarrow$" part of Lemma 12.6 does not hold anymore. Instead we can state the following.

Lemma 12.8. *Let SA be a stochastic automaton and let $PTS_c(SA)$ be its open behaviour. Then, for all valuations $v \in \mathbf{V}$,*

1. *if $[s, v] \xrightarrow{a(d)} (s', v - d)$ for some $d \in \mathbb{R}_{\geq 0}$ then $s \xrightarrow{a,C} s'$ for some $C \subseteq \mathcal{C}$; and*
2. *if $[s, v] \xrightarrow{a(d)}\!\!\!\!\!\!/\;\;$ for all $a \in \mathcal{A}$, $d \in \mathbb{R}_{\geq 0}$, then $s \xrightarrow{b,C}\!\!\!\!\!\!/\;\;$, for all $b \in \mathcal{A}$, $C \subseteq \mathcal{C}$.*

The first part of the lemma follows immediately from rule **Closed** in Definition 9.5. The second part can be proven by contradiction. Suppose that $s \xrightarrow{a,C} s'$. Then, arguing as before, it follows that there is a d sufficiently large such that $\forall x \in C.\ (v - d)(x) \leq 0$ holds. Take d to be the smallest possible. Without loss of generality, assume d to be the least enabling value for all edges outgoing from s. Then, it follows that the maximal progress condition $\forall d' \in [0, d).\ \forall s \xrightarrow{b,C'} .\ \exists y \in C'.\ (v - d')(y) > 0$ holds, and by rule **Closed**, $[s, v] \xrightarrow{a(d)} (s', v - d)$, which take us to a contradiction.

Because of the first item of the previous lemma, every run of $PTS_c(SA)$ has a corresponding symbolic run of SA. As a consequence, we have the following result.

Theorem 12.9. *Given a location s of a stochastic automaton SA, for every initial location s_0 and initial valuation v_0, if $s \notin reach(SA, s_0)$ then, for all $v \in \mathbf{V}$, $[s, v] \notin reach(PTS_c(SA), (s_0, v_0))$.*

Together with the second part of Lemma 12.8, this result is sufficient to check for freedom of deadlock: if SA does not have a reachable deadlock state, neither does $PTS_c(SA)$.

These results allow us to carry out reachability analysis at a purely symbolic level, without the construction of the underlying infinite probabilistic transition system and without using the clock information in the stochastic automaton. In this way we can exploit powerful existing tools like CADP (Fernandez et al., 1996) or SPIN (Holzmann, 1997) for carrying out the reachability analysis.

Although an appropriate translation from ♤ to LOTOS or PROMELA —the respective input languages of CADP and SPIN— and the use of those powerful tools would achieve an ideal performance in this setting, we have implemented a simple tool for deadlock detection as a prototype in order to show feasibility.

Contrarily to the simulation algorithm, deadlock detection cannot be applied to any ♤ terms. Instead, the associated stochastic automaton needs to be finite. To obtain this, we restrict the ♤ terms according to the following syntax.

$$\begin{array}{lcllllllllll} q & ::= & \mathbf{0} & | & a;q & | & C \mapsto q & | & \{\!|C|\!\}\, q & | & q+q & | & X \\ p & ::= & q & | & q[f] & | & p \,||_A\, p \end{array}$$

A term constructed according to the first clause is a *sequential process*; terms constructed using the second clause are referred to as *parallel processes*. In addition, we need to require that the recursive specification which defines the process variables contains finitely many guarded recursive equations of the from $X = q$. That is, a specification must have finitely many process variables and each of them is defined by guarded sequential processes.

The deadlock detection algorithm is implemented in Haskell 1.4 (Peterson and Hammond, 1997) as part of our discrete-event simulator. The algorithm has as input a parallel process (the system specification) together with the recursive specification that defines its variables. If a deadlock location is reachable, the algorithm returns a symbolic execution that ends in such a location. The implementation is based on a selective state space search using a partial-order reduction technique based on persistent sets (Overmans, 1981; Godefroid, 1996). The adoption of this technique allows the construction of the stochastic automata only when it is demanded. In this way, it can be avoided to completely construct the stochastic automaton for every sequential process in the system specification.

12.4 Simple Queueing Systems

In this and the subsequent sections we report on experiments carried out using the developed tools. In this section we report on simple examples of performance analysis on queueing systems. In the following sections we concentrate on more involved examples.

In Example 11.19, we have seen a simple queueing system in which jobs arrive and wait until they are served by a single server. The different parts of the system were specified as follows:

$$\begin{array}{lcll} \mathit{Queue}_0 & = & \mathit{in};\mathit{Queue}_1 & \\ \mathit{Queue}_{n+1} & = & \mathit{in};\mathit{Queue}_{n+2} + \mathit{out};\mathit{Queue}_n & (n \geq 0) \\ \mathit{Arrival} & = & \{\!|x|\!\}\,\{x\} \mapsto \mathit{in};\mathit{Arrival} & \\ \mathit{Server}_i & = & \mathit{out};\mathit{start}_i;\{\!|y_i|\!\}\,\{y_i\} \mapsto \mathit{done}_i;\mathit{Server}_i & \end{array}$$

We have slightly altered the definition of the process *Server* in order to specify and analyse systems with many servers. The subindex i helps to ensure that there is no conflict of clock names and, in addition, to keep track on a run of which server is active without the need to inspect the state. That is why we have subscripted actions start_i and done_i. This last decision is merely pragmatic in order to gain efficiency at implementation level.

A queueing system with an unbounded queue and m servers can be described as follows

$$\mathit{QSystem} \;=\; \mathit{Queue}_0 \,||_Q\, \left(\mathit{Arrival} \,||_\varnothing\, (|||_{i=1}^{m}\, \mathit{Server}_i)\right)$$

where $Q = \{in, out\}$ and $|||_{i=1}^{m} p_i \stackrel{\text{def}}{=} p_1 \,||_{\varnothing}\, p_2 \,||_{\varnothing} \cdots ||_{\varnothing}\, p_m$, with $m > 0$. If $F_{y_i} = F_{y_j}$ for all $0 \leq i < j \leq m$, and F_{y_i} and F_x are arbitrary distribution functions, this kind of system is known as a $G/G/m/\infty$ queue. Notice that if $m = 1$ then we have the same system as in Example 11.19. If in particular F_{y_i} and F_x are negative exponential distribution functions then the system is called an $M/M/m/\infty$ queue (the Ms standing for "Markovian").

We can also describe a system with a bounded queue. A bounded queue with capacity $b > 0$ can be described as follows.

$$\begin{array}{lcll} BQueue_0 & = & in; BQueue_1 & \\ BQueue_{n+1} & = & in; BQueue_{n+2} + out; BQueue_n & (0 \leq n < b) \\ BQueue_b & = & in; BQueue_b + out; BQueue_{b-1} & \end{array}$$

Notice that as soon as the queue reaches its capacity, then any new arrival is lost. Since we abstract from data, this might not seem obvious. Nevertheless, it is revealed in process $BQueue_b$ by observing that an action in does not increase the capacity of the queue. A $G/G/m/b$ queue system can be described by the process

$$BQSystem \;=\; BQueue_0 \,||_Q \left(Arrival \,||_{\varnothing}\, (|||_{i=1}^{m} Server_i)\right).$$

In order to run the simulation we need to define an adversary. Assuming that clocks respond to continuous distribution functions, the only situation of non-determinism that may happen with non-zero probability is if more than one server attempt to take a job out of the queue. Since the servers are symmetric, it is irrelevant for the performance of the system which of them will process the job. As a consequence we use a simple adversary that resolves non-deterministic situations with equal probability. Thus, we take the sample space Ω'_ρ to be defined as the set of all enable edges at the end of the run, i.e.

$$\Omega'_\rho \stackrel{\text{def}}{=} \left\{ s \xrightarrow{a,C} s' \mid last(\rho) = [s, v] \wedge s \xrightarrow{a,C} s' \text{ is enabled in } v \right\},$$

and the adversary to be defined by

$$\mathcal{A}(\rho)(e) \stackrel{\text{def}}{=} \frac{1}{\#\Omega'_\rho}$$

We studied the following parameters:

- the *queue length Nq*, that is, the number of jobs that are waiting in the queue to be served.
- the *number of jobs Nj in the system*. They are the jobs that are waiting in the queue plus those that are currently being served.
- the *waiting time Wt*, that is, the amount of time a job waits in a queue to be served.
- the *response time Rt*. This is the time that a job spends in the system, since it is queued until its service finished.

We are interested in estimating the average of these parameters on the long run execution of the system. If such long run execution reaches an equilibrium as the time goes to infinity, we say that the system finds a *steady-state*.

Some additional performance measures we are interested in are

- the *throughput* Tp, which is the number of jobs served per time unit, and
- the *utilisation* Ut, which is the fraction of time that a server is busy.

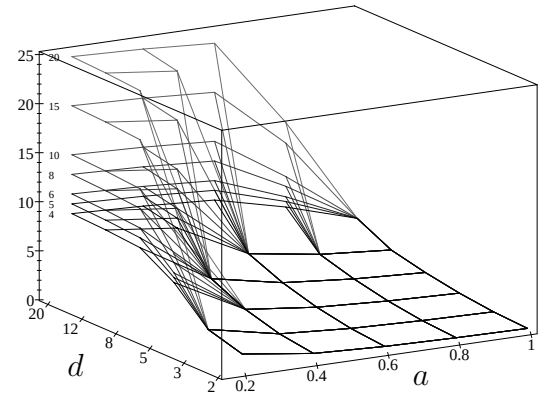
For an efficient functioning of the system, it is desirable to keep the mean of the parameter Nq, Nj, Wt, and Rt as small as possible, whereas parameters Tp and Ut are expected to be as large as possible.

We report on experiments made on a $G/G/m/b$ queue system where the arrival of jobs is controlled by an Erlang distribution with scale parameter a (or rate $\frac{1}{a}$) and k phases, and the service time is given by a uniform distribution in the interval $[d - 0.2d, d + 0.2d]$, i.e., a uniform distribution with mean d ranging on the $\pm 20\%$ interval. Thus,

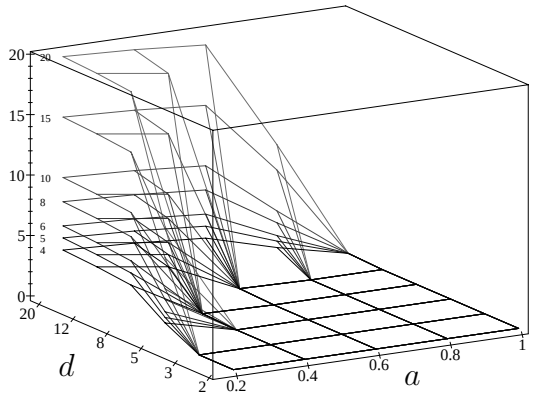
$$F_x(t) = 1 - e^{-t/a}\left(\sum_{j=0}^{k-1}\frac{(t/a)^j}{j!}\right) \qquad F_{y_i}(t) = \begin{cases} 0 & \text{if } t < 0.8\,d \\ \dfrac{t - 0.8\,d}{0.4\,d} & \text{if } 0.8\,d \le t < 1.2\,d \\ 1 & \text{if } 1.2\,d \le t \end{cases}$$

In particular, we fix the number of phases $k = 5$. In Figure 12.3, we give several results for a system with 5 servers. Each plot reports on studies with seven different queue lengths, taking the values 4, 5, 6, 8, 10, 12, and 20, each one of them represented by a different plane. The x-axis indicates the variation of the arrival rate. It varies the scale parameter a of the Erlang distribution. We let a take the values 0.2, 0.4, 0.6, 0.8, and 1. As a consequence, the average of arrivals are of 1 job each 1, 2, 3, 4, and 5 time units, respectively[1]. The y-axis indicates the variation of the service rate, that is the parameter d in the uniform distribution. We let d take the values 2, 3, 5, 8, 12, and 20, which corresponds with the mean of the service time. Figure 12.3(a) gives the mean number of jobs in the system, and Figure 12.3(b) the mean number of jobs in the queue. Notice that there is a correlation between both of them. Moreover, notice that the number of jobs in the system (and in the queue) increases when the service is slower and the arrival is faster. Similarly, Figure 12.3(c), which gives the response time, and Figure 12.3(d), which gives the waiting time, are also correlated and they increase when the service is slower and the arrival is faster. Notice that in all cases, there are certain parameters in which the results studied under different queue lengths start to be indistinguishable from each other. In particular, the points (a, d) in which they start to be quite close are $(0.2, 3)$, $(0.4, 8)$, $(0.6, 12)$, $(0.8, 12)$, and $(1, 20)$. In fact those values are a fraction before the arrival rate starts to be larger than the service rate which would happen at the values in which the mean of interarrival time, $5 \cdot a$, is equal to the apparent average of service time d/m where $m = 5$ is the number of servers. In the picture, these points would correspond to the

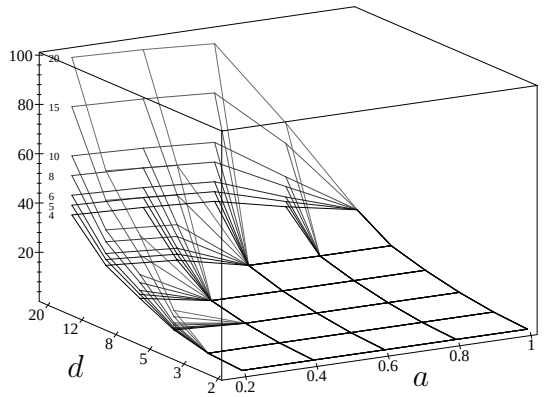
[1]The mean of an Erlang distribution with k phases and scale parameter a is $k \cdot a$



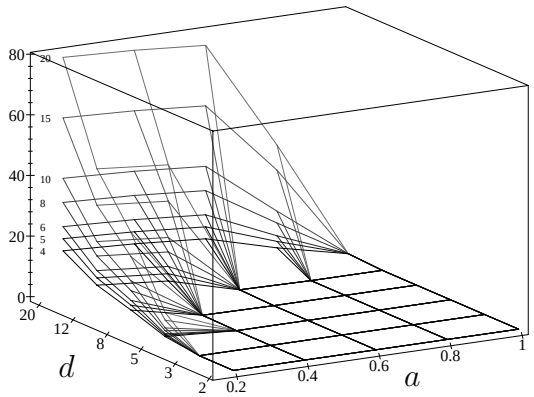
(a) Numbers of jobs in the system



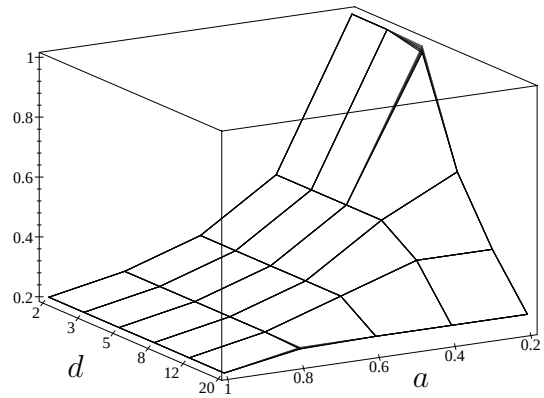
(b) Numbers of jobs in the queue



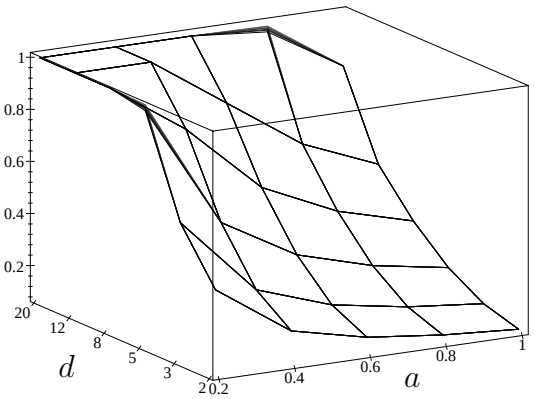
(c) Response time



(d) Waiting time



(e) Throughput



(f) Utilisation

Figure 12.3: Results of the $G/G/5/b$ system

coordinates $(0.2, 5)$, $(0.4, 10)$, $(0.6, 15)$, $(0.8, 20)$, and $(1, 25)$. According to our estimations, we could say that if the relation $\frac{d/5}{5 \cdot a}$ is between $\frac{4}{5}$ and $\frac{3}{5}$, then the minimal queue length of 4 would not produce results sensibly different from any of the longer queues.

For the throughput, given in Figure 12.3(e), and the utilisation, reported in Figure 12.3(f), the different queue lengths do not make a significant difference in the obtained measurements. Notice that the throughput increases when both the arrival and the service are faster. In particular, notice that if the service becomes slower, it is more likely that the queue is full, thus increasing the probability that a new job arrives and misses the chance to enter the queue and eventually been processed. Utilisation follows the pattern of the previous plots: it increases when the service is slower and the arrival faster. However, in this case, the consequences are different since higher utilisation is desirable although, for instance, higher response time is not.

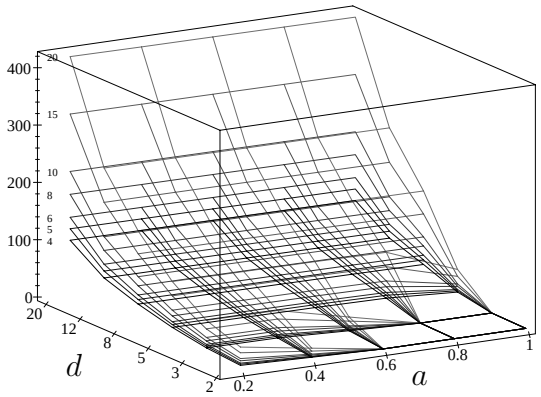
In Figure 12.4, we compare the response time varying the number of servers from 1 to 5. We can see that the curves are steeper as the number of servers grows. This is due to the fact that by increasing the number of servers, the total service rate also increases and hence also the likelihood that a job is served in shorter time. Notice that the relation between a and d for which the queue length does not make a significant difference on the response time, also varies as the number of servers varies, always keeping the value of $\frac{d/m}{5 \cdot a}$ a fraction smaller than 1.

The simulations have been carried out using the method of *batch means*. It consists of running a long simulation run, discarding the initial transient interval, and dividing the remainder of the run into several batches or subsamples (Jain, 1991). We took 20 subsamples, each one of approximately 10000 time units long. The values in the figures are the overall averages. In every case, we calculated the respective confidence interval. For those setups in which $\frac{d/m}{5 \cdot a} = 1$, the intervals grow really large. The (proportionally) widest confidence interval was obtained for the $G/G/1/20$ system with $a = 1$ and $d = 5$ in the estimation of the waiting time, being its value 48.482628 ± 491.053 with 90% of confidence. Such an enormous interval is due to the variability occurring in this setting. The difficulty to reach a stable situation as a consequence of the variation of the frequency on both the arrival and the service would make the system now work at full queue, now with perhaps no jobs in the system at all. To overcome this situation, the length of each run should be much larger as well as the number of subsamples should be considerably increased.

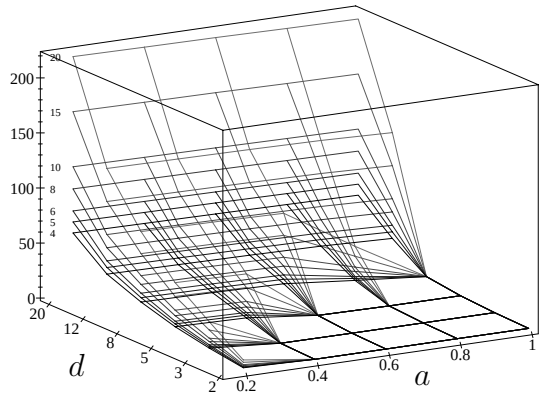
Instead, the system reaches a stable situation more rapidly if $\frac{d/m}{5 \cdot a}$ grows larger or gets closer to 0. The worst case situation we found in the analysis of the system with 5 servers, and in which $\frac{d/m}{5 \cdot a} \neq 1$, was in the $G/G/4/5$ with $a = 1.0$ and $d = 20$ (notice that $\frac{d/m}{5 \cdot a} = 4/5$) with the following confidence intervals

$E[Nj] \in$	4.128682	$\pm$ 0.00650699 (90%conf.)	$\pm$ 0.0107318 (99%conf.)
$E[Nq] \in$	0.129055	$\pm$ 0.00138082 (90%conf.)	$\pm$ 0.00227239 (99%conf.)
$E[Rt] \in$	20.630921	$\pm$ 0.0393234 (90%conf.)	$\pm$ 0.0648552 (99%conf.)
$E[Wt] \in$	0.642913	$\pm$ 0.0318108 (90%conf.)	$\pm$ 0.0524648 (99%conf.)

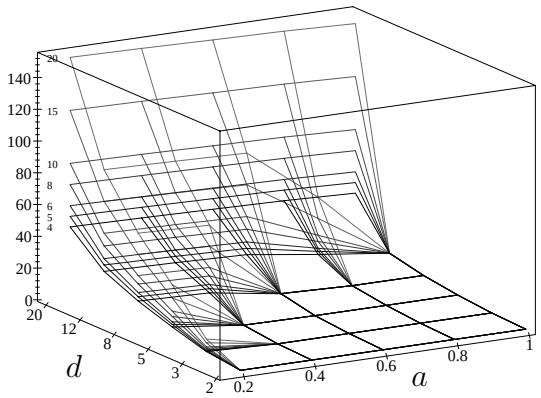
The proportionally longer confidence interval for the throughput was found when $a = 0.6$



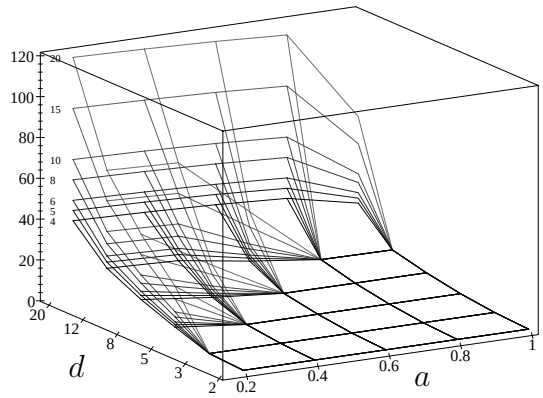
(a) 1 Server



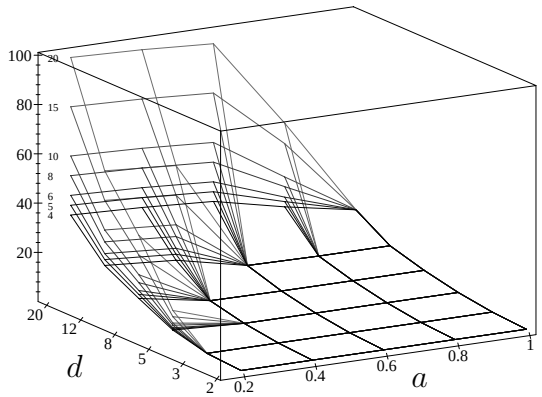
(b) 2 Servers



(c) 3 Servers



(d) 4 Servers



(e) 5 Servers

Figure 12.4: Comparison of the response time in the $G/G/m/b$ system

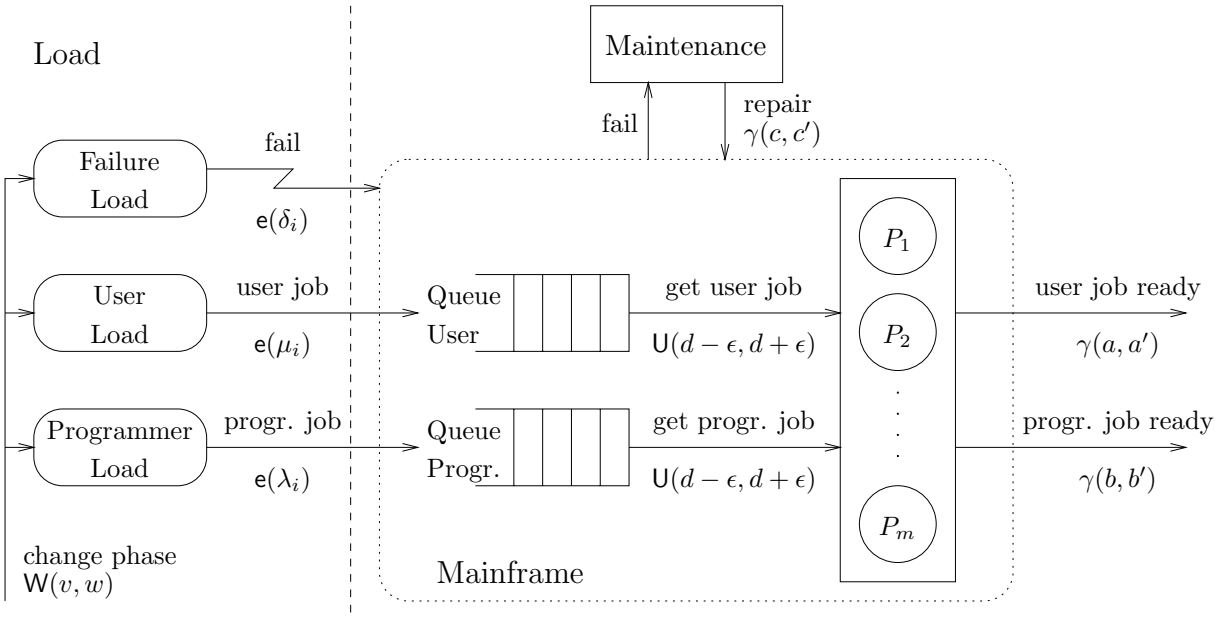


Figure 12.5: Architecture of the mainframe system

and $d = 2$, being its value $0.333805 \pm 2.47523 \cdot 10^{-5}$ with 99% confidence. For the utilization, it was found in a $G/G/5/5$ with $a = 1.0$ and $d = 20$, being $0.800640 \pm 1.62106 \cdot 10^{-4}$ with 99% confidence.

12.5 A Multiprocessor Mainframe

In this section we discuss an example that we borrowed from (Herzog and Mertsiotakis, 1994). There all activities are delayed according to an exponential distribution. Instead, we will use arbitrary distributions. The example deals with a multiprocessor mainframe that serves two purposes. On the one hand, the system has to maintain a database and therefore has to process transactions submitted by different users. On the other hand, it is used for program development and thus it has to serve programmer's requirements such as compilation and testing. Besides, the system can be altered by the occurrence of software failures such as database inconsistencies, or operating system panics. The architecture of the multiprocessor system is presented in Figure 12.5.

We describe the compositional specification of the system, the approach that led us to an adversary, and finally present the results of analysing this system both qualitatively and quantitatively.

We will use the following notation. We subscript a clock with the kind of distribution it responds to. Thus, if $\mathsf{e}(\lambda)$ represents an exponential distribution with parameter λ, then $x_{\mathsf{e}(\lambda)}$ is a clock whose distribution function is the exponential distribution function $F_{x_{\mathsf{e}(\lambda)}}(t) = 1 - e^{-\lambda t}$. In addition, we will use the traditional way of representing prefixes as

a short-hand notation:

$$a(x); p \stackrel{\text{def}}{=} \{\!|x|\!\}\ \{x\} \mapsto a; p.$$

Formal specification using ⌂. The system has three parts: the mainframe itself, the maintenance module that takes care of repairing failures, and the system load which basically is the environment representing the user and programmer job arrivals and the occurrence of failures.

$$System \ = \ Load \,||_{L}\, (Mainframe \,||_{F}\, Maintain)$$

Process *Load* synchronises with the *Mainframe* each time a user or a programmer sends a job requirement or when a failure occurs. Process *Maintain* only synchronises when a failure occurs and when it is repaired. So

$$L = \{usrJob, prgJob, fail\} \quad \text{and} \quad F = \{fail, repair\}$$

Process *Load* models the user and programmer load, and the failure occurrences

$$Load \ = \ PrgLoad_1 \,||_{\{chng\}}\, UsrLoad_1 \,||_{\{chng\}}\, FailLoad_1 \,||_{\{chng\}}\, ChngPhase$$

Process *ChngPhase* is intended to model the variation of the load along the time (e.g. the load during the night is different from that on peak hours.) Phases change according to a Weibull distribution function with parameters (v, w) (denoted by $\mathsf{W}(v, w)$)[2].

$$ChngPhase \ = \ chng(x_{\mathsf{W}(v,w)}); ChngPhase$$

There are three phases. In the first phase, user jobs arrive according to an exponential distribution with rate μ_1 (notation $\mathsf{e}(\mu_1)$); in the second phase, arrivals are distributed according to $\mathsf{e}(\mu_2)$, and in the third phase no user job is generated. In any case, if a job cannot be queued because either the queue is full or the system has failed, the job is simply rejected. Similarly, programmer jobs arrive according to $\mathsf{e}(\lambda_1)$ and $\mathsf{e}(\lambda_2)$ in the first and second phase, and failures originate according to $\mathsf{e}(\delta_1)$ and $\mathsf{e}(\delta_2)$, respectively. We model the occurrence of a system failure regardless how many errors induce that failure. Processes *UsrLoad*, *PrgLoad*, and *FailLoad* are as follows.

$$\begin{aligned}
UsrLoad_1 \ &= \ nextUsrJob(xu_{\mathsf{e}(\mu_1)}); (usrJob; UsrLoad_1 + reject; UsrLoad_1) \\
&\quad + chng; UsrLoad_2 \\
UsrLoad_2 \ &= \ nextUsrJob(xu_{\mathsf{e}(\mu_2)}); (usrJob; UsrLoad_2 + reject; UsrLoad_2) \\
&\quad + chng; UsrLoad_3 \\
UsrLoad_3 \ &= \ chng; UsrLoad_1
\end{aligned}$$

[2]Distribution functions along the example were chosen arbitrarily with the intention of showing the versatility of ⌂ and the simulator in this respect.

$$
\begin{aligned}
PrgLoad_1 &= nextPrgJob(xp_{\mathsf{e}(\lambda_1)}); (prgJob; PrgLoad_1 + reject; PrgLoad_1) \\
&\quad + chng; PrgLoad_2 \\
PrgLoad_2 &= nextPrgJob(xp_{\mathsf{e}(\lambda_2)}); (prgJob; PrgLoad_2 + reject; PrgLoad_2) \\
&\quad + chng; PrgLoad_3 \\
PrgLoad_3 &= chng; PrgLoad_1 \\
FailLoad_1 &= nextFail(xf_{\mathsf{e}(\delta_1)}); (fail; FailLoad_1 + reject; FailLoad_1) \\
&\quad + chng; FailLoad_2 \\
FailLoad_2 &= nextFail(xf_{\mathsf{e}(\delta_2)}); (fail; FailLoad_2 + reject; FailLoad_2) \\
&\quad + chng; FailLoad_3 \\
FailLoad_3 &= chng; FailLoad_1
\end{aligned}
$$

The *Mainframe* consists of two *Queues* and a given number of processors P_i. The different processes are fully synchronised with the actions *fail* and *repair*: when a failure occurs the complete system must stop until it is repaired. Besides, each processor p_i can synchronise with both *Queues* in order to load the next user or programmer job that is going to be processed. Since we consider that transferring a job from a queue to a processor has a certain duration, we split the action into a begin part and an end part. So, we define the *Mainframe* by the following equation.

$$Mainframe = Queues \,||_{G \cup F}\, (P_1 \,||_F\, P_2 \,||_F \cdots ||_F\, P_m)$$

where $G = \{getUsrJobBegin, getPrgJobBegin\}$.

The *Queues* respond to the FIFO policy. One of the queues is used by the user jobs and the other by the programmer jobs. They are symmetric, only varying in their length. So, we use a unique process scheme for the queue and we take advantage of the renaming operation.

$$
\begin{aligned}
Queues &= QUsr \,||_F\, QPrg \\
QUsr &= Q_0^{nu}[f_u] \\
QPrg &= Q_0^{np}[f_p]
\end{aligned}
$$

with

$f_u(job) = usrJob$ $\qquad$ $f_p(job) = prgJob$

$f_u(getBegin) = getUsrJobBegin$ $\qquad$ $f_p(getBegin) = getPrgJobBegin$

otherwise: $f_u(a) = a$ $\qquad$ otherwise: $f_p(a) = a$

Each queue is modelled by a set of processes Q_i^n where n ($n > 0$) is the length of the queue and i the number of queued jobs. A queue is blocking if there is a failure or if it is full. In such case, jobs are rejected.

$$
\begin{aligned}
Q_0^n &= job; Q_1^n + fail; repair; Q_0^n \\
Q_i^n &= job; Q_{i+1}^n + getBegin; Q_{i-1}^n + fail; repair; Q_i^n \qquad 0 < i \leq n-1 \\
Q_n^n &= getBegin; Q_{n-1}^n + fail; repair; Q_n^n
\end{aligned}
$$

The processors are symmetric. Nevertheless, we should be careful in order to avoid conflict of clock variables. We use indices to distinguish clock names. A processor gets either a user or a programmer job from the respective queue. In any case, this process takes d units of time with an error of $\pm\epsilon$ distributed uniformly (i.e. according to a uniform distribution $\mathsf{U}(d-\epsilon, d+\epsilon)$). Notice that actions *getUsrJobEnd* and *getPrgJobEnd* are local to the processor. We do so because we consider that this activity takes part of the service time of the processor, and not of the queue. After loading a job, the processor executes it. The execution time of a user job is distributed according to a gamma distribution with parameters (a, a') (notation $\gamma(a, a')$). The execution of a programmer job is distributed according to $\gamma(b, b')$. A failure induces the processor to abort any activity. When the system is repaired, each processor restarts from its initial state.

$$\begin{aligned}
P_i &= \mathit{getUsrJobBegin}; (\mathit{getUsrJobEnd}(y^i_{\mathsf{U}(d-\epsilon,d+\epsilon)}); PWU_i + PF_i) \\
&\quad + \mathit{getPrgJobBegin}; (\mathit{getPrgJobEnd}(y^i_{\mathsf{U}(d-\epsilon,d+\epsilon)}); PWP_i + PF_i) \\
&\quad + PF_i \\
PWU_i &= \mathit{usrJobReady}(zu^i_{\gamma(a,a')}); P_i + PF_i \\
PWP_i &= \mathit{prgJobReady}(zp^i_{\gamma(b,b')}); P_i + PF_i \\
PF_i &= \mathit{fail}; \mathit{repair}; P_i
\end{aligned}$$

Process *Maintain* simply repairs a failure each time it occurs. The reparation time is distributed according to $\gamma(c, c')$.

$$\mathit{Maintain} = \mathit{fail}; \mathit{repair}(z_{\gamma(c,c')}); \mathit{Maintain}.$$

Defining an adversary. Previously, we presented the model of the mainframe. We notice some choices we would like to add to the specification in order to define a suitable adversary.

(a) In process *UsrLoad* a situation of non-determinism may arise between actions *reject* and *usrJob*. We would like not to reject a user job if there exists a chance that the mainframe may become available at the same moment. The same consideration applies to processes *PrgLoad* and *FailLoad*. In fact, we want that *reject* has the lowest priority amongst all actions.

(b) A failure is an arbitrary event that at any moment may disturb the normal execution of the system. For that reason, we would not like to make a failure wait if it is enabled. So, we consider that action *fail* has the highest priority.

(c) User jobs are usually short activities that have to be processed as soon as possible, such as saving a file or processing a small database transaction. Programmer jobs are more complicated tasks that may involve compilation, simulation or testing of a system. From this observation, we would like that user jobs have higher priority than programmer jobs.

Architecture	Load		Processing	
$m = 4$	$\mu_1 = 0.033$	$\mu_2 = 2$	$d = 0.021$	$e = 0.001$
$nu = 4$	$\lambda_1 = 0.01667$	$\lambda_2 = 0.16$	$a = 0.16667$	$a' = 0.5$
$np = 5$	$v = 300$	$w = 6$	$b = 0.16667$	$b' = 2.0$

Table 12.1: Parameters for the studied mainframe

Summarising, we consider the priority relation $\prec$ defined as the least (strict partial) order satisfying the following conditions

$$\begin{aligned} \mathit{reject} &\prec a \;\Leftrightarrow a \neq \mathit{reject} \\ a &\prec \mathit{fail} \;\Leftrightarrow a \neq \mathit{fail} \\ \mathit{getPrgJobBegin} &\prec \mathit{getUsrJobBegin} \end{aligned}$$

This priority relation is used to define the adversary that we use to simulate the multiprocessor mainframe. If after reducing the possible activities to be executed according to the defined priorities there is some nondeterminism remaining, we let the adversary to resolve it according to a (discrete) uniform probability distribution. Formally we define the adversary as follows.

The sample space Ω'_ρ is defined as the set of all enable edges at the end of the run, i.e.

$$\Omega'_\rho \stackrel{\text{def}}{=} \left\{ s \xrightarrow{a,C} s' \mid \mathit{last}(\rho) = [s, v] \wedge s \xrightarrow{a,C} s' \text{ is enabled in } v \right\}$$

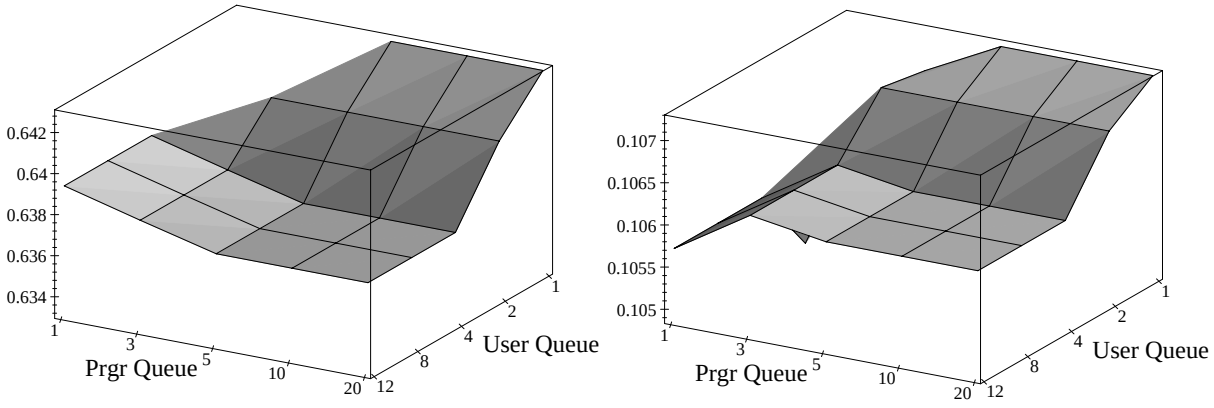
We use the auxiliary operation pri to prune those enabled edges with lower probability than other enabled edges, that is, $\mathit{pri}(\rho)$ returns the maximal elements in Ω'_ρ according to the $\prec$ order.

$$\mathit{pri}(\rho) \stackrel{\text{def}}{=} \left\{ s \xrightarrow{a,C} s' \in \Omega'_\rho \mid \neg\exists s \xrightarrow{b,C'} t \in \Omega'_\rho \wedge a \prec b \right\}$$

The adversary is then simply defined as follows

$$\mathcal{A}(\rho)(e) \stackrel{\text{def}}{=} \begin{cases} \dfrac{1}{\#\mathit{pri}(\rho)} & \text{if } e \in \mathit{pri}(\rho) \\ 0 & \text{otherwise} \end{cases}$$

Simulation results. We set the values of the different parameters according to Table 12.1. As in (Herzog and Mertsiotakis, 1994), we studied the behaviour of the system with different queue lengths. We ran several simulations changing the length of the queues (with $\delta_1 = 0.0007$, $\delta_2 = 0.00035$, and $c' = 100$). We can see in Figure 12.6 that both user



(a) Throughput User Jobs

(b) Throughput Programmer Jobs

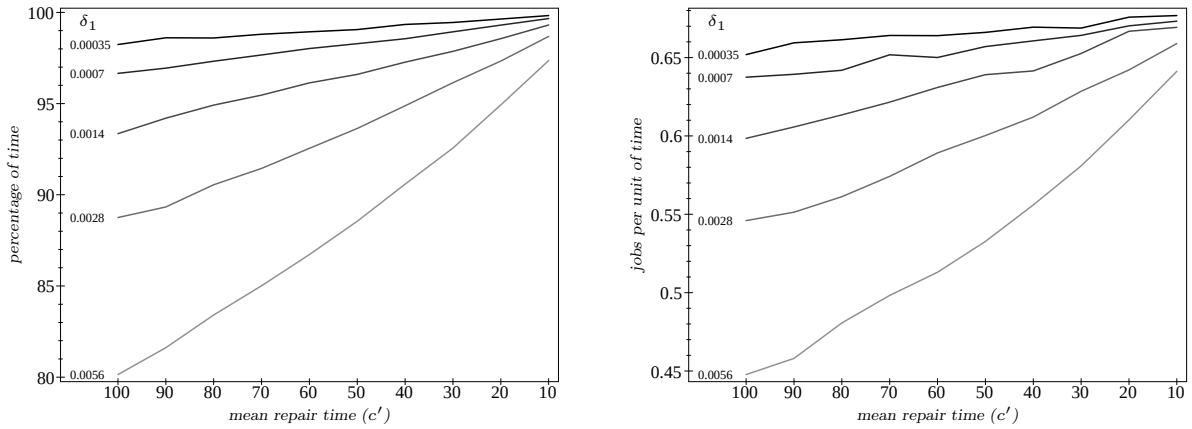
Figure 12.6: Studying the length of the queues

and programmer job throughput stabilise when the user and programmer queue length are at least 4 and 5, respectively (notice that the planes in the pictures become horizontal from that point on). That is, queues larger than those values do not affect the throughput. Therefore, we finally fixed the values to 4 and 5 respectively (see Table 12.1).

Different simulations have been carried out while changing the parameters related to failure and repairing. In all cases we took $\delta_2 = \delta_1/2$. For the repair time we take $c = 1$. Hence, the average repair time equals c'. The simulation results are depicted in Figures 12.7(a) and 12.7(b). Figure 12.7(a) represents the availability of the system, that is, the percentage of time the mainframe is processing jobs. Figure 12.7(b) depicts the throughput of user jobs, i.e. the number of jobs that are processed successfully per time unit.

To calculate the user job throughput, we simply count the number of occurrences of action *usrJobReady* per time unit. To determine the availability we count the occurrences of the action *fail* per time unit instead, say *fpm*, and then calculate $100 \cdot (1 - fpm \cdot c')$. Since c' is the average of repair time, then $fpm \cdot c'$ is the fraction that the system is unavailable per time unit. The simulation algorithm also allows for the direct calculation of the availability, but in our case the method we followed is more precise.

We have used again the method of batch means to calculate the results. We took 30 subsamples, each one of approximately 80000 minutes length. The values in the figures are the overall averages. In every case, we calculated the respective confidence interval. The (proportionally) widest confidence interval was obtained for $\delta_1 = 0.0028$ and $c' = 90$ in the case of the throughput: $0.551299 \pm 8.65232 \cdot 10^{-4}$ with 99% of confidence. In the case of the availability, the widest confidence interval was for $\delta_1 = 0.0028$ and $c' = 100$: 88.755299 ± 3.14581 with confidence 99%.



(a) Availability

(b) Throughput User Jobs

Figure 12.7: Results in a mainframe with values fixed as in Table 12.1

Reachability analysis. Since the multiprocessor system is finite, we are able to automatically check reachability properties. Using the prototype tool described at the end of Section 12.3, we checked that the process *Mainframe* is deadlock-free.

12.6 Transient Analysis in a Simple Protocol

In this section we address a different kind of analysis. In contrast to the steady state analysis performed in the previous sections, we are now interested in the observation of the execution of a system until it satisfies a certain property. Simple examples of this kind of questions are "how long takes me in average to cook spaghetti alla carbonara" or "how many employees usually arrive at the office before the boss does". In both cases, the experiment has a clear beginning and a clear end. The way we proceed in this type of analysis is by repeating the same experiment many times in order to gather a significant amount of information from which we can derive an approximation of the average and variance as well as depicting histograms which would help to form an idea of the shape of the actual probability distribution function of the studied property answers.

As an example of transient analysis, we will study the root contention part in the protocol IEEE 1394 "FireWire". The IEEE 1394 high performance serial bus (IEEE Computer Society, 1996) is a standardised protocol that has been developed for the interconnection of digital multimedia equipment. The protocol has several layers. The physical layer, which is the lower, has several phases for which there is one that involves the election of the leader among several nodes. The leader election protocol proceeds in a deterministic way, but it is possible that at the end of this election procedure there are two remaining nodes that intend to signal each other as the leader (or *root*). This situation is called root contention. The root contention protocol is a probabilistic algorithm that has been formally verified

by Stoelinga and Vaandrager (1999) in a probabilistic fashion. They checked that exactly one node is eventually elected as a leader with probability one. In the following, we answer the question how long the resolution of the root contention takes by approximating the probability distribution of the time that takes to find a leader since the root contention is detected. We base our specification on (Stoelinga and Vaandrager, 1999) and work within the restrictions that ensure the correctness of the protocol.

Formal specification. In the root contention, exactly two *Node*s are involved. They communicate with each other via two unidirectional *Wire*s. The overall system can be described by

$$RootCont \;=\; (Node_0 \,||_{\varnothing}\, Node_1) \,||_{Sig}\, (Wire^0_{\mathsf{empty}} \,||_{\varnothing}\, Wire^1_{\mathsf{empty}})$$

where $Sig = \{send^i_s, receive^i_s \mid s \in \{\mathsf{req}, \mathsf{ack}\} \wedge i \in \{0,1\}\}$ is the set of signals they interchange. The signals are saved in a *Buffer* at the node's port. The *Process*ing part of the node can check the buffer at any time. Thus, for $i \in \{0,1\}$, we define

$$Node_i \;=\; Process_i \,||_{Chk}\, Buffer^i_{\mathsf{empty}}$$

with $Chk = \{check_s \mid s \in \{\mathsf{empty}, \mathsf{req}, \mathsf{ack}\}\}$.

The nodes are symmetric. Their behaviour can be described as follows. When a node detects a root contention, it chooses with equal probability whether to wait for a short or a long time. The short time is somewhere in the interval $[a_s, b_s]$ while the long time is somewhere in the interval $[a_l, b_l]$. It is assumed that $0 \le a_s \le b_s < a_l \le b_l$. We code this stochastic waiting time in the clock variables x_i, whose distribution we discuss later. After having waited for the indicated period, the node may have already received a request message ($check_{\mathsf{req}}$). If this is the case, the node acknowledges this to its contender ($send^i_{\mathsf{ack}}$) and declares itself the root ($root_i$). If instead the node did not receive any message ($check_{\mathsf{empty}}$), it sends a request message ($send^i_{\mathsf{req}}$) and waits for an answer. If the answer is an acknowledgment ($check_{\mathsf{ack}}$), it declares itself a child ($child_i$) and terminates. If the answer is also a request ($check_{\mathsf{req}}$), the node assumes there is still root contention and starts the process all over again.

$$\begin{aligned}
Process_i &= wait(x_i); (\; check_{\mathsf{empty}}; SendingReq_i \\
&\qquad + check_{\mathsf{req}}; SendingAck_i \;) \\
SendingReq_i &= send^i_{\mathsf{req}}; (\; check_{\mathsf{req}}; Process_i \\
&\qquad + check_{\mathsf{ack}}; child_i; \mathbf{0} \;) \\
SendingAck_i &= send^i_{\mathsf{ack}}; root_i; \mathbf{0}
\end{aligned}$$

The *Buffer* that accompanies the *Process* in a node is a one place buffer that can be overwritten. In addition, once its content is read, the buffer is marked as empty.

$$\begin{aligned}
Buffer^i_s \;=\;& check_s; Buffer^i_{\mathsf{empty}} \qquad (\text{for } s \in \{\mathsf{empty}, \mathsf{req}, \mathsf{ack}\}) \\
&+ receive^i_{\mathsf{req}}; Buffer^i_{\mathsf{req}} \\
&+ receive^i_{\mathsf{ack}}; Buffer^i_{\mathsf{ack}}
\end{aligned}$$

A *Wire* receives a signal on one end (actions $send_s^i$) and transmits it to the other end (actions $receive_s^{1-i}$). The velocity of the signal in the cable can reach a maximum of 198 *mtr/μsec*. We assume the transmission time to be distributed according to a random variable y_i. We discuss its probability distribution function below.

$$\begin{array}{lcll} Wire^i_{\mathsf{empty}} & = & send^i_{\mathsf{req}}; Wire^i_{\mathsf{req}} & \\ & & + send^i_{\mathsf{ack}}; Wire^i_{\mathsf{ack}} & \\ Wire^i_s & = & receive^{1-i}_s(y_i); Wire^i_{\mathsf{empty}} & (\text{for } s \in \{\mathsf{req}, \mathsf{ack}\}) \\ & & + send^i_{\mathsf{req}}; Wire^i_{\mathsf{req}} & \\ & & + send^i_{\mathsf{ack}}; Wire^i_{\mathsf{ack}} & \end{array}$$

The distribution of the clocks. The waiting time in a node i is determined by the clock x_i. With probability $\frac{1}{2}$, it should wait some time in the interval $[a_s, b_s]$, and the other half of the probability induces a waiting time in the interval $[a_l, b_l]$. We assume that the value of each interval can be chosen with a uniform distribution. Thus, provided that $0 \leq a_s \leq b_s < a_l \leq b_l$, the distribution of x_i, $i = 1, 0$, is given by the function

$$F_{x_i}(t) = \begin{cases} 0 & \text{if } \ t < a_s \\ \dfrac{1}{2} \cdot \dfrac{t - a_s}{b_s - a_s} & \text{if } \ a_s \leq t < b_s \\ \dfrac{1}{2} & \text{if } \ b_s \leq t < a_l \\ \dfrac{1}{2} \cdot \left(1 + \dfrac{t - a_l}{b_l - a_l}\right) & \text{if } \ a_l \leq t < b_l \\ 1 & \text{if } \ b_l \leq t \end{cases}$$

The transmission through a wire i takes a certain time which depends on the length of the cable and is described by a clock y_i. We assume that transmission takes place with a maximal velocity v and it can drop down to $v/2$, though the average is around $\frac{4}{5}v$. We suppose that the transmission time can be approximated by means of a *beta distribution* with parameters $\beta_1 = 2$ and $\beta_2 = 6$. Thus, given that m is the length of the cable, the probability density function of clock y_i, $i = 1, 0$, is given by

$$f_{y_i}(t) \ = \ \begin{cases} \dfrac{\left(\frac{v}{m}t - 1\right)\left(2 - \frac{v}{m}t\right)^5}{\beta(2,6)} & \text{if } \ \frac{m}{v} \leq t < 2\frac{m}{v} \\ 0 & \text{if } \ t < \frac{m}{v} \ \text{ or } \ 2\frac{m}{v} \leq t \end{cases}$$

where β is a particular function that is well known in statistics and probability theory (see, e.g. Devore, 1982; Jain, 1991; Shiryaev, 1996). As usual, the distribution function is defined as the integral of f_{y_i}

$$F_{y_i}(t) = \int_{\frac{m}{v}}^{t} f_{y_i}(x)dx$$

	a_s	b_s	a_l	b_l	m
IEEE 1394	0.24 μsec	0.26 μsec	0.57 μsec	0.60 μsec	< 15.345 mtr
IEEE 1394a	0.76 μsec	0.80 μsec	1.60 μsec	1.64 μsec	< 39.6 mtr

Table 12.2: Parameters for the root contention protocol

which, in this case, can be computed only numerically.

To guarantee the correctness of the protocol it is necessary that the maximum transmission delay is strictly smaller than a_s and $\frac{a_l - b_s}{2}$ (Stoelinga and Vaandrager, 1999). We can consider that the maximum transmission delay is the lowest upper bound of the support set of F_{y_i}, that is, $2m/v$ which is the distance the message may travel divided by the slowest speed. So,

$$2\,\frac{m}{v} \;<\; \min\left\{a_s, \frac{a_l - b_s}{2}\right\}$$

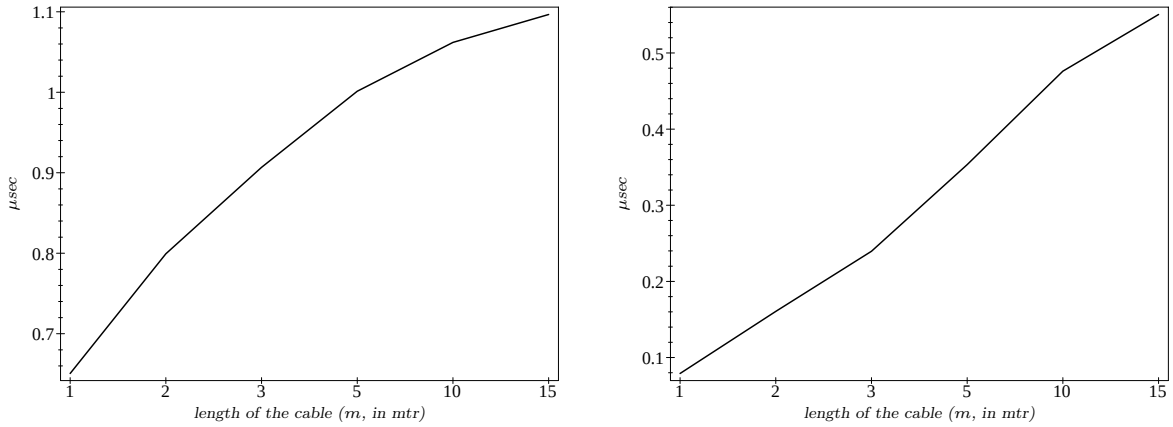
Assuming that $v = 198$ $mtr/\mu sec$ then $m < \min\{99a_s, 49.5(a_l - b_s)\}$. In order to proceed with the analysis of the protocol, we consider the values of a_s, b_s, a_l, and b_l as they are specified in the standard of the IEEE 1394 (IEEE Computer Society, 1996) and the draft of the IEEE 1394a (IEEE Computer Society, 1998). We report these values and the bound for m in Table 12.2.

Although we have respected the parameters imposed by the standards, we do not know the real shape of the distributions. We chose them rather arbitrarily. In fact, the standardisation of the protocol cannot specify a particular distribution functions. They would depend on the actual architecture of the whole system that is particular to each implementation.

Simulation results. It can be checked that the closed behaviour of the protocol is deterministic with probability 1. As a consequence no adversary is necessary to carry out the simulation.

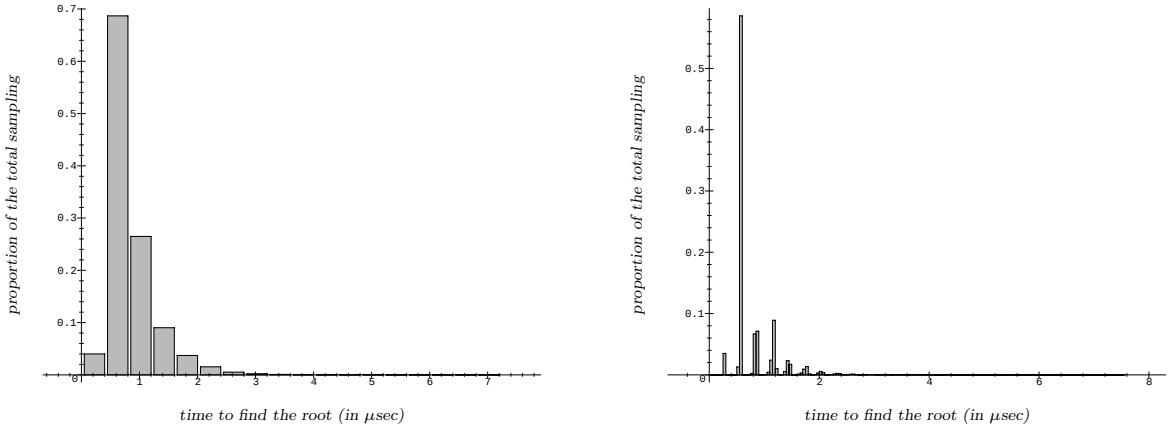
For each setting, we have run 5 series of 100000 simulations each starting with different seeds for the random number generator. In the case of the IEEE 1394, we let m, the length of the cable, range over 1, 2, 3, 5, 10, and 15 meters. The average and variance of each case are reported in Figure 12.8, where it can be observed their increase according m grows.

In Figure 12.9(a), we report a histogram to have an idea of the shape of the probability density function of the total time to find the root. In this case, we report on the protocol running with a cable of 2 mtr. The ith column in the plot reports the proportion of experiments that takes between $0.4 \cdot (i-1)$ and $0.4 \cdot i$ μsec to find the root. According to the picture, in the majority of the cases, a root is found within at most two "contention rounds" (counting the initial one, which is always present).



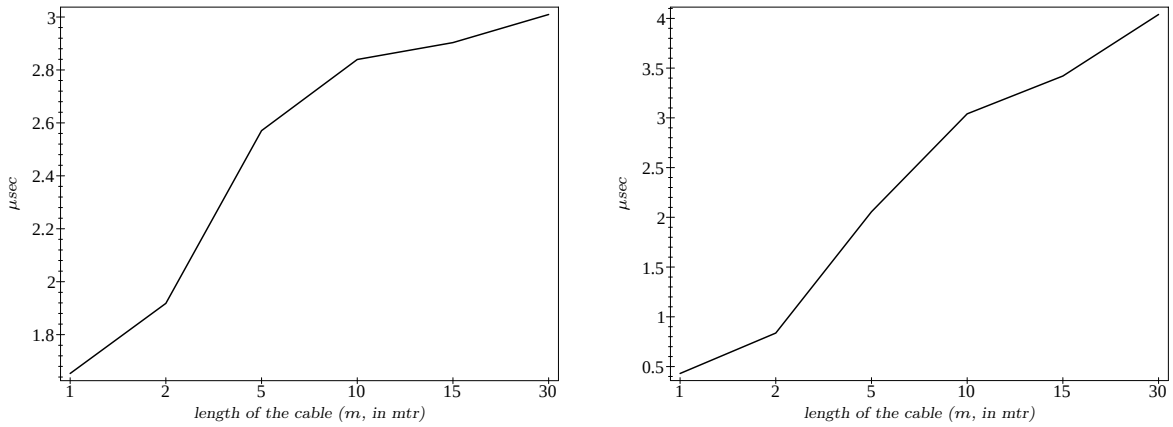
(a) Average (b) Variance

Figure 12.8: Time until resolution of root contention in the IEEE 1394



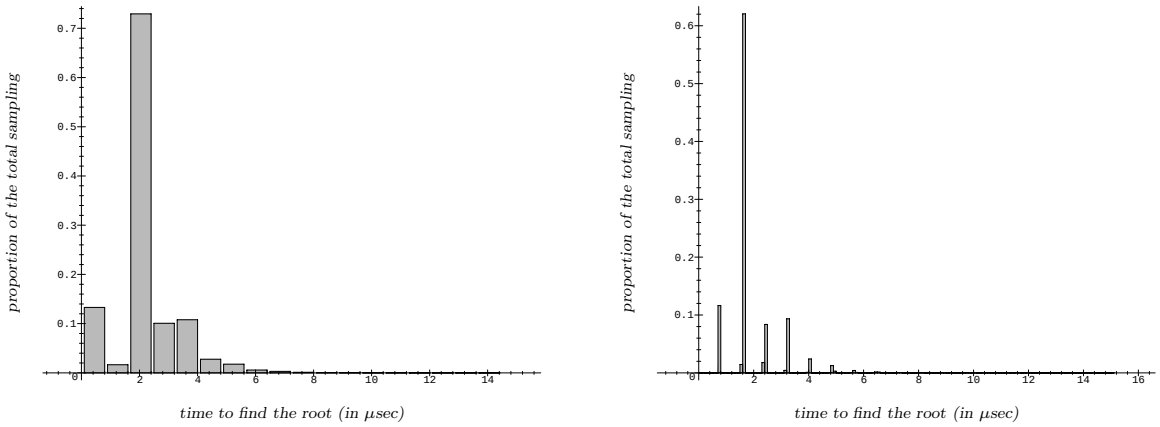
(a) Interval length: 0.4 μsec (b) Interval length: 0.05 μsec

Figure 12.9: Histogram for the IEEE 1394 with $m = 2$ *mtr*



(a) Average (b) Variance

Figure 12.10: Time until resolution of root contention in the IEEE 1394a



(a) Interval length: 0.8 μsec (b) Interval length: 0.1 μsec

Figure 12.11: Histogram for the IEEE 1394a with $m = 2$ mtr

Although the histogram of Figure 12.9(a) resembles a smooth curve, the actual probability density function is noticeably oscillating. This can be observed in Figure 12.9(b) where the columns are narrower, that is, they represent the proportion of experiments with respect to smaller intervals of only 0.05 μsec length.

In Figures 12.10 and 12.11 we give a similar report but for the case of the IEEE 1394a. In this case we let m range over 1, 2, 5, 10, 15, and 30 meters. Figure 12.11 reports the histograms for the case of the system communicated through a cable of 2 mtr.

Notice that the new version of the protocol shows a lower performance. The averages of the time until the resolution of the root contention are more than twice bigger in the IEEE 1394a for the same cable length. This is a consequence of the larger delays a_s, b_s, a_l, and b_l. On the other hand, the IEEE 1394a allows for the correct communication of longer distances.

12.7 Related Work

A preliminary attempt that constructs simulation models from process algebra has been given in (Katoen et al., 1996). For recursive processes, though, this approach yields infinite objects which makes this approach less suited for the use of efficient (regenerative) simulation techniques.

Another process algebra for discrete-event simulation has been presented by Harrison and Strulo (1995) and applied to a cache coherency protocol in (Field et al., 1998). The semantic objects are highly infinite. To simulate a specification it is translated into C++ while using some simulation libraries. Although their work is related to ours, we use a different process algebra, allow non-determinism and use the concept of adversaries for its resolution, and obtain (for most processes) finite stochastic automata. To our knowledge, no support for qualitative analysis is incorporated in (Harrison and Strulo, 1995; Field et al., 1998).

Other works that relate simulation models (or languages) to process algebra are (Pooley, 1995; Birtwistle and Tofts, 1996). In these works, the approach is different: rather than generating a simulation automatically from a process algebra specification (as we do), they use process algebra as a semantic model for simulation languages. In addition, these works do not take probabilistic timing into consideration; they only consider functional correctness.

12.8 Concluding Remarks

In this chapter we discussed the foundations and the feasibility of discrete event simulation as well as the verification and validation of ♤ specifications. We believe that the results are highly satisfactory for the following reasons:

- ♤ allows for a straightforward and smooth combination of qualitative and quantitative analysis. Although we have not intensively explored the practice of qualitative

analysis, the experience of Section 12.5 shows that exactly the same specification can be used to do both performance and validation analysis. In addition, the theoretical results of Section 12.1 state that very little effort is necessary to translate a ♤ specification into, for example, LOTOS, while preserving the safety properties. In so doing, CADP can be used to automatically model check the original specification.

- The importance of having discrete event simulation algorithm for ♤ allows us to fully concentrate on the specification of the system under consideration rather than on the appropriate scheduling of events as it would happen when general purpose languages like C or C++ are used instead. Moreover, the way we carried out the collection of quantitative information was by simple inspection of the occurrence of actions and their respective timing. In many cases, like when estimating throughput or the number of jobs in the system (see Sections 12.4 and 12.5), the information is gathered by simply counting the occurrence of particular actions. In other cases, the treatment should be slightly more careful. For instance, when estimating the waiting time or response time, it would be necessary to save some information about the occurrence time of certain actions.

- In particular, the root contention protocol of the IEEE 1394 studied in Section 12.6 is a real protocol. Though we did not model the execution time in the different parts of the protocol, in our opinion, the reported study is quite realistic and we can expect that the estimated averages are upper bounds of the actual values. (The consideration of the execution time would induce more variability, hence diminishing the chances of root contention if both nodes choose to wait the same amount of time!) The satisfactory study of a real protocol at this relatively early stage in the development of ♤ and its discrete event simulator are encouraging for future theoretical and applied development of this framework.

Chapter 13

Concluding Remarks

— *Probablemente ya sea hora de terminar —dijo el Doctor.*[†]

Bruno Gerardo Épinde, *Cita de los ritmos*. Wetten, 1999.

This chapter takes a retrospective view on the contributions of this dissertation. It summarises our achievements based on the aims and motivations discussed in Chapter 1 and provides a number of directions and ideas for future research.

13.1 Achievements

In this thesis we discussed algebras and automata for hard and soft real-time systems. In the first part we addressed the issue of hard real-time systems. Although we discussed some new topics in the area of timed automata, the major contribution of this part is the process algebra ♡. ♡ provides an *equational theory for timed automata*. The concept of algebra is foundational in mathematics. Therefore, by introducing ♡, we have proposed an alternative viewpoint for the understanding of timed automata and provided a hierarchical and structured way to model real-time systems. We gave an equational theory that is sound and complete for *symbolic bisimulation*, and extended it with some additional axioms preserving soundness for *timed bisimulation*. In addition, ♡ serves as a language for the hierarchical and structured specification of hard real-time systems. We discussed a small example in Chapter 7. We remark that it was shown that ♧, which is a traditional untimed process algebra, is a proper subalgebra of ♡. However, the major contribution of this first part lies in the paradigmatic framework it provides for the development of a new theory for the description and analysis of stochastic systems.

Most of the novel contributions of this thesis can be found in the second part, which deals with soft real-time systems. Inspired by existing models, we have defined in Chapter 8, a model to represent the semantics of stochastic processes whose probabilistic behaviour can be arbitrarily distributed.

[†] "Probably, it is already time to finish," said the Doctor.

In Chapter 9, we introduced the *stochastic automata model*. Stochastic automata are the result of combining ideas taken from timed automata and generalised semi-Markov processes (GSMP). As a result, we obtained a *symbolic* stochastic model that supports *compositionality* and enables the description of events whose timing are *arbitrarily distributed*.

In fact, the parallel composition of stochastic automata can be carried out in a straightforward manner as explained in Chapter 10. A consequence of parallel composition is that *non-determinism* comes into play. The stochastic automata model allows the description of non-determinism and this feature is one of the main differences with GSMPs. The behaviour represented by a GSMP can nevertheless be expressed by some stochastic automaton, as shown in Section 9.5.

We argued that graphic notations alone are not suitable for the description of large systems. As Holger Hermanns (1998) stated: "Nowadays, there is a growing awareness that if there is hope that industrial-size designs can be handled by a performance-estimation methodology, this methodology must be based on [...] compositionality". This is the aim of the process algebra ♤. We conceived ♤ as a formal framework for the specification, modelling and analysis of stochastic systems. Since the semantics of ♤ is given in terms of stochastic automata, events' timing can be arbitrarily distributed.

As a stochastic process algebra, ♤ allows for the *qualitative and quantitative analysis* of systems. We provided ♤ with several tools to study functional correctness. To begin with, several equivalence relations based on bisimulation were defined on the underlying semantics (see Section 9.4). In Chapter 11, we provided *equational theories* that allow for the syntactic analysis of ♤ specifications. We proved that they conservatively extend ♧. Finally, in Section 12.3, we gave the theoretic foundations for (untimed) reachability analysis of stochastic automata. We used this result to implement a deadlock detection algorithm based on partial-order methods.

The techniques for stochastic analysis of ♤ specifications were reported in Sections 12.1 and 12.2. Though non-determinism is of significant importance for the functional analysis of systems, it turns out to be inconvenient for quantitative analysis. In order to resolve it, we proposed the use of *probabilistic adversaries* or *schedulers*. We devised a *discrete event simulation* algorithm that generates random executions from a given ♤ specification where the non-determinism is solved according to a selected adversary. Discrete event simulation allows us to *estimate* the performance parameters that we are interested in. We have reported a prototypical tool that implements this algorithm, and used it in several case studies. The case studies include steady state analysis of queueing systems and a multiprocessor mainframe, and transient analysis of a part of the IEEE 1394 protocol.

13.2 Future Research Directions

Stochastic automata and ♤ open new research possibilities in an area mainly dominated by Markovian process algebras. In this dissertation, we introduced this framework and applied it successfully to several case studies. This encourages us to address new theoretical and

applied developments. We distinguish between three different global directions: functional analysis, performance and reliability analysis, and extensions of ⌂ expressiveness. We discuss these issues separately.

Functional Analysis. In Section 12.3, we discussed a very first approach to functional analysis of stochastic automata by disregarding the stochastic aspects. We showed that several correctness properties can be checked using this approach. These properties include safety properties such as absence of deadlock. It would be advantageous, however, to formally relate the stochastic automata model with models for the description of timed systems such as timed automata. Such a connection would enlarge considerably the spectrum of properties that could be verified. These properties include liveness and timing aspects. In addition, the advantage of a mapping onto timed automata extends to the potential use of existing tools like KRONOS (Bozga et al., 1998) and UPPAAL (Larsen et al., 1997). The variant of timed automata originally proposed by Sifakis and Yovine (1996) and further developed in (Bornot et al., 1998) treats the concept of invariant in a different way. Invariants are the key concept to model hard constraints in timed automata. By using so-called *deadlines* instead, synchronisation in parallel composition can be relaxed to model soft real-time constraints. This composition is comparable to ⌂ parallel composition. Therefore, a compositional translation of stochastic automata into timed automata with deadlines should be feasible.

In any case, the properties that can be checked using the techniques of Section 12.3 or using a translation into timed automata do *not* include probabilistic information. Knowing the probabilistic value of a property can be quite significant. In many cases, errors cannot be completely eliminated from a system due to unpredictable behaviour, for instance, due to the environment in which it interacts. Knowing the likelihood of such errors at an early stage of the system development will help to take further decisions in the design and implementation of such systems. Scientific work in this direction is scarce and the formal methods community has only recently started to address the issue. Most of the work on the verification of probabilistic systems treats discrete probabilities (e.g Hansson, 1994; Segala, 1995) and few considered timed automata extended with discrete probabilities (Jensen, 1996; Kwiatkowska et al., 1999). The case of continuous probabilities has been studied to a much smaller degree. Alur et al. (1991) gave a model checking algorithm for GSMPs, but the logic does not provide probabilistic information. Recently, verification algorithms for Markov decision processes augmented with time information (de Alfaro, 1997, 1998) and continuous time Markov chains (Baier et al., 1999) have been reported.

Performance and Reliability Analysis. There are two approaches to study system models. On the one hand, *analytical and numerical* approaches provide a precise mathematical answer to the problem of calculating performance and reliability parameters. However, this kind of treatment is restricted to some classes of stochastic processes as, for example, continuous time Markov chains. A first approach to numerically solve ⌂ specifications would be the identification of the Markovian subset. A possible way to proceed is to find a formal relation between ⌂ and some Markovian process algebra. These process

algebras have been thoroughly studied in the last decade. In this respect, Hermanns' interactive Markov chains (IMC) (1998) provide an adequate framework. Like ⌂, IMC separates the stochastic timing aspects from the discrete actions, though it is only a two way-split: roughly speaking, in IMC we write $(x)a;p$ instead of $\{\!|x|\!\}\,\{x\} \mapsto a;p$ where x is an exponentially distributed clock. The moment of setting a clock is irrelevant in IMC due to the memoryless property of exponential distributions of Markov chains. IMC also considers non-determinism and adopts the same policy for synchronisation as ⌂. The connection of ⌂ with Markov processes can also profit from the already existing wide arsenal of software tools for the analysis of continuous time Markov chains (see e.g. Couvillion et al., 1991; Stewart, 1994; Hermanns et al., 1998; Haverkort and Niemegeers, 1996). Numerical solutions are not restricted to Markov processes. Often, partial or approximate answers can be obtained in a more general setting. Therefore, another direction to pursue is the identification of other subsets of ⌂ that can be solved numerically, e.g., based on Clark's work on insensitive analysis (1999).

A technique that is complementary to analytical and numerical analysis is *simulation*. Simulation is applicable, in principle, to any class of stochastic processes. In particular, we have used *discrete event simulation* techniques in Chapter 12. The algorithm presented there can be improved considerably. For instance, the approach we have used to resolve non-determinism, although technically sound, requires a particular insight in the system model in order to select a suitable adversary (see also the next item). In addition, it is necessary to incorporate advanced simulation techniques in order to obtain results in an efficient manner, specially when it involves the simulation of events that occur with a very low frequency. For this reason, rare event simulation techniques should be investigated in the context of ⌂.

Extending Expressiveness. As it is now, ⌂ is a language with some few basic constructions that serves mainly theoretical purposes. The usefulness of a language as part of the system design process depends on how expressive it is and how easy to use. Therefore, ⌂ could be extended with some basic data types and data structures that allow the specification of complex and large systems in a concise and simple manner. In this direction, a value passing-based calculus, such as (Milner, 1989; Milner et al., 1992; Hennessy and Lin, 1995; Groote, 1997; Victor, 1998), could serve as a paradigmatic reference.

In addition, the problem of solving non-determinism (also addressed in the previous item) can be attacked from a syntactic point of view. Two essential operations in process algebras, and in ⌂ in particular, are the non-deterministic choice and the parallel composition. In order to solve non-determinism, these operations can be parameterised with priorities, probabilistic information, or schedulers. A study in a simple probabilistic setting has recently been reported in (D'Argenio et al., 1999a) which, in a way, proposes an outline of the work to be done in the context of ⌂.

An important point to address is *abstraction*. If we compare the queues of Examples 11.19 and 9.28 —which can be found in Figure 13.1— we notice a similitude. The only difference is the action *out* that is used for the sole purpose of synchronising the *Queue* and *Server* processes. By hiding action *out*, we can say that it has become an inter-

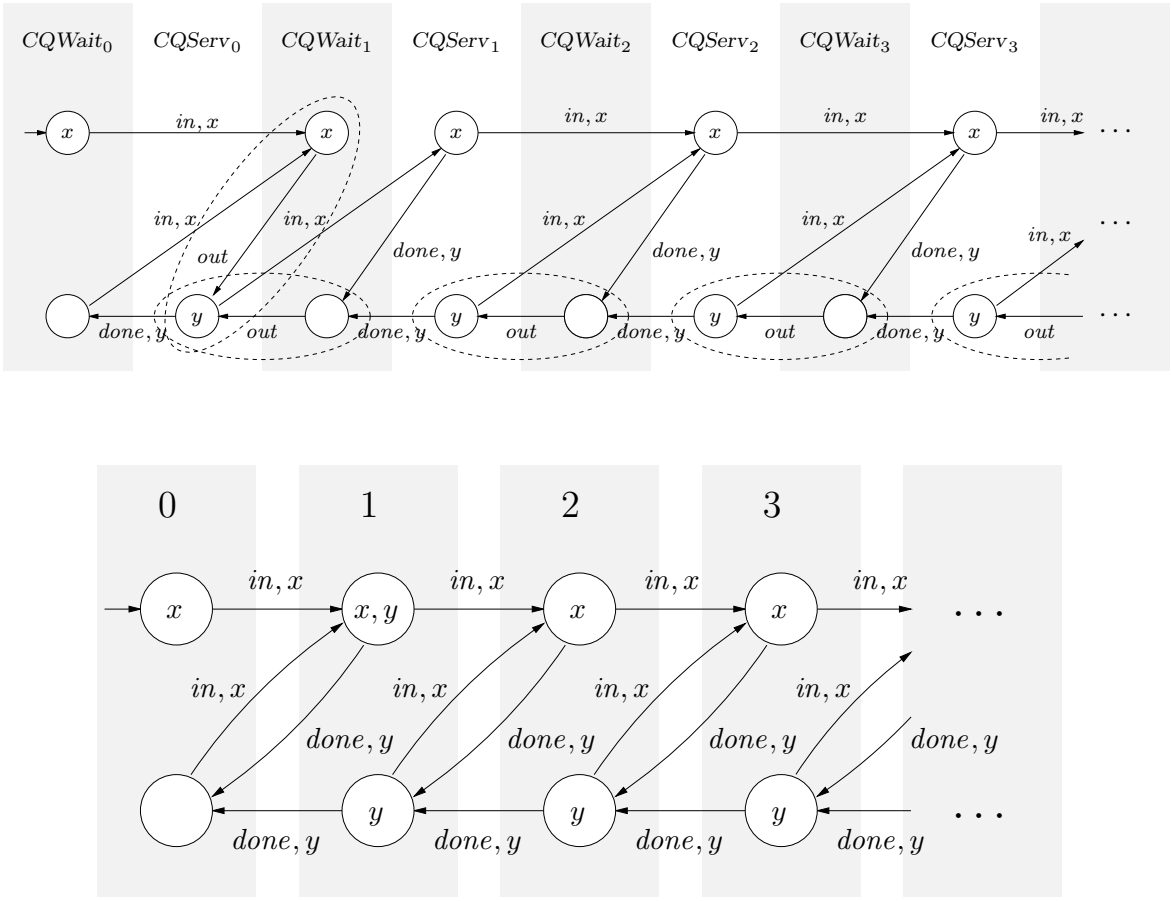


Figure 13.1: Two weakly bisimilar stochastic automata?

nal action that has no influence on the behaviour as understood by an external observer. Therefore, it would be interesting to be able to equate them. To do so, an appropriate notion of abstraction has to be developed. Based on this notion, equivalences such as *weak* or *branching bisimulations* could be defined. In this setting, the weak bisimulation defined on IMC (Hermanns et al., 1995; Hermanns, 1998) is a good candidate on which this research can be based.

Part Four

Technicalities

"D'oh!"

Homer Simpson. Springfield, 1999.
(Created by Matt Groening)

Appendix A

Proofs from Chapter 4

A.1 $\sim_{\&}$ is Transitive

The fact that $\sim_{\&}$ is transitive follows immediately from the following lemma.

Lemma A.1. *Let R_1 and R_2 be two symbolic bisimulations, then the relation*

$$\begin{aligned} R \;=\; \big\{ \langle s_1, s_3, SR\rangle \mid & \langle s_1, s_2, SR_1\rangle \in R_1 \wedge \langle s_2, s_3, SR_2\rangle \in R_2 \\ & \wedge \big(\langle C_1, C_3\rangle \in SR \iff \\ & \quad (\exists C_2, C_2'. \\ & \qquad (\langle C_1, C_2\rangle \in SR_1 \wedge \langle C_2', C_3\rangle \in SR_2 \\ & \qquad\qquad \wedge C_2 \cap FV(s_2) = C_2' \cap FV(s_2) \,) \\ & \qquad \vee (\langle C_1, C_2\rangle \in SR_1 \wedge C_2 \cap FV(s_2) = C_3 = \varnothing) \\ & \qquad \vee (\langle C_2, C_3\rangle \in SR_2 \wedge C_1 = C_2 \cap FV(s_2) = \varnothing) \,) \,\big) \\ & \wedge \langle \kappa(s_1), \kappa(s_3)\rangle \in SR \\ & \wedge \big(\langle C_1, C_3\rangle, \langle C_1', C_3'\rangle \in SR \implies (C_1 = C_1' \iff C_3 = C_3') \,\big) \\ & \wedge \text{for } i = 1, 3, \textstyle\bigcup\{C_i \mid \langle C_1, C_3\rangle \in SR\} \supseteq FV(s_i) \cup \kappa(s_i) \big\} \end{aligned}$$

is a symbolic bisimulation.

Proof. Let $\langle s_1, s_3, SR\rangle \in R$. Then there are $\langle s_1, s_2, SR_1\rangle \in R_1$ and $\langle s_2, s_3, SR_2\rangle \in R_2$ satisfying the above conditions. We prove that $\langle s_1, s_3, SR\rangle$ satisfies items 1–6 in Definition 4.16.

1. Suppose $\langle C_1, C_3\rangle, \langle C_1', C_3'\rangle \in SR$ and $(C_1 \cap C_1') \cup (C_3 \cap C_3') \neq \varnothing$. Assume $(C_1 \cap C_1') \neq \varnothing$. The proof follows in a similar way if $(C_3 \cap C_3') \neq \varnothing$. We can proceed as follows

$$\begin{aligned} & \langle C_1, C_3\rangle, \langle C_1', C_3'\rangle \in SR \text{ and } (C_1 \cap C_1') \neq \varnothing \\ & \implies && \text{(by def. of } SR\text{)} \\ & \quad \langle C_1, C_2\rangle, \langle C_1', C_2'\rangle \in SR_1 \;\text{ and }\; (C_1 \cap C_1') \cup (C_2 \cap C_2') \neq \varnothing \end{aligned}$$

$$\begin{array}{ll} \Longrightarrow & \text{(since } \langle s_1, s_2, SR_1\rangle \in R_1 \text{ and } R_1 \text{ is a symbolic bisimulation)} \\ \langle C_1, C_2\rangle = \langle C_1', C_2'\rangle & \\ \Longrightarrow & \text{(by def. of } SR\text{: } C_1 = C_1' \iff C_3 = C_3') \\ \langle C_1, C_3\rangle = \langle C_1', C_3'\rangle. & \end{array}$$

We remark that if $C_2 \cap FV(s_2) = C_2' \cap FV(s_2) \neq \varnothing$, the proof can be carried out alternatively without using condition $C_1 = C_1' \iff C_3 = C_3'$. In fact this condition needs to be explicit only to avoid ambiguity in the case that $C_2 \cap FV(s_2) = \varnothing$.

2. Immediate by the definition of R.

3. Immediate by the definition of R. It is worth to mention that $\langle \kappa(s_1), \kappa(s_2)\rangle \in SR_1$ and $\langle \kappa(s_2), \kappa(s_3)\rangle \in SR_2$ which, provided that $\kappa(s_2) \cap FV(s_2) \neq \varnothing$, implies $\langle \kappa(s_1), \kappa(s_3)\rangle \in SR$. The condition in the definition of R needs to be explicit otherwise the case that $\kappa(s_2) \cap FV(s_2) = \varnothing$ may induce this requirement to fail.

4. Define ξ_1 and ξ_2 such that for all $x, y, z \in \mathcal{C}$

$$\begin{array}{l} \xi_1 \circ \xi^1_{SR_1}(x) = \xi^1_{SR}(x), \\ \xi_2 \circ \xi^2_{SR_2}(z) = \xi^2_{SR}(z), \text{ and} \\ \xi_1 \circ \xi^2_{SR_1}(y) = \xi_2 \circ \xi^1_{SR_2}(y) \end{array} \tag{A.1}$$

Notice that if $\xi^1_{SR_1}(x) = \xi^2_{SR_1}(y)$ and $\xi^1_{SR_2}(y) = \xi^2_{SR_z}(z)$, then $\xi^1_{SR}(x) = \xi^2_{SR}(z)$. As a consequence ξ_1 and ξ_2 can always be defined.

It is not difficult to check that

$$\begin{array}{l} \xi_1 \circ \xi^2_{SR_1}(\iota(s_2)) \equiv \xi_2 \circ \xi^1_{SR_2}(\iota(s_2)), \\ \xi^1_{SR}(\iota(s_1)) \equiv \xi_1 \circ \xi^1_{SR_1}(\iota(s_1)), \text{ and} \\ \xi^2_{SR}(\iota(s_3)) \equiv \xi_2 \circ \xi^2_{SR_2}(\iota(s_3)). \end{array} \tag{A.2}$$

Because R_1 and R_2 are symbolic bisimulation, we have that

$$\models \xi^1_{SR_1}(\iota(s_1)) \Leftrightarrow \xi^2_{SR_1}(\iota(s_2)) \quad \text{and} \quad \models \xi^1_{SR_2}(\iota(s_2)) \Leftrightarrow \xi^2_{SR_2}(\iota(s_3)).$$

Since ξ_1 and ξ_2 are substitutions, we still have that

$$\begin{array}{l} \models \xi_1 \circ \xi^1_{SR_1}(\iota(s_1)) \Leftrightarrow \xi_1 \circ \xi^2_{SR_1}(\iota(s_2)) \quad \text{and} \\ \models \xi_2 \circ \xi^1_{SR_2}(\iota(s_2)) \Leftrightarrow \xi_2 \circ \xi^2_{SR_2}(\iota(s_3)). \end{array}$$

By the observation (A.2) it follows that

$$\models \xi^1_{SR}(\iota(s_1)) \Leftrightarrow \xi^2_{SR}(\iota(s_3)).$$

5. Assume $s_1 \xrightarrow{a,\phi_1} s_1'$. If $\models \neg(\xi^1_{SR}(\iota(s_1) \wedge \phi_1))$, the transfer property follows immediately. Otherwise, since $\langle s_1, s_2, SR_1\rangle \in R_1$, it holds that

(b_1) there are $s_2^{(1)}, \ldots, s_2^{(n)} \in \mathcal{S}$, $n \geq 1$ such that

- i_1. $s_2 \xrightarrow{a,\phi_2^{(i)}} s_2^{(i)}$ for all $i = 1, \ldots, n$;
- ii_1. $\models \xi^1_{SR_1}(\iota(s_1) \wedge \phi_1) \Rightarrow \bigvee_i \xi^2_{SR_1}(\iota(s_2) \wedge \phi_2^{(i)})$; and
- iii_1. for all i, there exists SR_1' such that $\langle s_1', s_2^{(i)}, SR_1' \rangle \in R_1$ and

$$\{\langle C_1 \cap FV(s_1'), C_2 \cap FV(s_2^{(i)})\rangle \mid \langle C_1, C_2\rangle \in SR_1'\} - \{\langle \varnothing, \varnothing\rangle\} = \{\langle C_1 \cap FV(s_1'), C_2 \cap FV(s_2^{(i)})\rangle \mid \langle C_1, C_2\rangle \in SR_1\} - \{\langle \varnothing, \varnothing\rangle\}.$$

Without lost of generality assume that it is never the case that $\models \neg(\xi^2_{SR_1}(\iota(s_2^{(i)}) \wedge \phi_2^{(i)}))$ for all $i = 1, \ldots, n$. By using the substitutions defined in (A.1), and noticing that ξ_1 and ξ_2 are bijections, it follows that $\models \neg(\xi^2_{SR_2}(\iota(s_2^{(i)}) \wedge \phi_2^{(i)}))$. From here and since $\langle s_2, s_3, SR_2\rangle \in R_2$, for each $i = 1, \ldots, n$,

(b_2) there are $s_3^{(i,1)}, \ldots, s_3^{(i,m_i)} \in \mathcal{S}$, $m_i \geq 1$ such that

- i_2. $s_3 \xrightarrow{a,\phi_3^{(i,j)}} s_3^{(i,j)}$ for all $j = 1, \ldots, m_i$;
- ii_2. $\models \xi^1_{SR_2}(\iota(s_2) \wedge \phi_2^{(i)}) \Rightarrow \bigvee_i \xi^2_{SR_2}(\iota(s_3) \wedge \phi_3^{(i,j)})$; and
- iii_2. for all j, there exists SR_2' such that $\langle s_2^{(i)}, s_3^{(i,j)}, SR_2'\rangle \in R_2$ and

$$\{\langle C_1 \cap FV(s_2^{(i)}), C_2 \cap FV(s_3^{(i,j)})\rangle \mid \langle C_1, C_2\rangle \in SR_2'\} - \{\langle \varnothing, \varnothing\rangle\} = \{\langle C_1 \cap FV(s_2^{(i)}), C_2 \cap FV(s_3^{(i,j)})\rangle \mid \langle C_1, C_2\rangle \in SR_2\} - \{\langle \varnothing, \varnothing\rangle\}.$$

Relating the previous statements we have that

(b) there are $s_3^{(i,j)} \in \mathcal{S}$ with $i = 1, \ldots, n$, $n \geq 1$, and $j = 1, \ldots, m_i$, $m_i \geq 1$.

i. $s_3 \xrightarrow{a,\phi_3^{(i,j)}} s_3^{(i,j)}$ for all $i = 1, \ldots, n$, $j = 1, \ldots, m_i$.

ii. Following a similar reasoning to item 4, and considering substitutions defined in (A.1), we have that

$$\models \xi^1_{SR}(\iota(s_1) \wedge \phi_1) \Rightarrow \bigvee_{i,j} \xi^2_{SR}(\iota(s_3) \wedge \phi_3^{(i,j)}).$$

iii. Choose SR' such that

$$\begin{aligned}
&\langle C_1, C_3\rangle \in SR' \iff \\
&\quad (\ \exists C_2, C_2'.\ (\ \langle C_1, C_2\rangle \in SR_1' \wedge \langle C_2', C_3\rangle \in SR_2' \\
&\qquad\qquad \wedge C_2 \cap FV(s_2) = C_2' \cap FV(s_2)\ ) \\
&\qquad\quad \vee (\langle C_1, C_2\rangle \in SR_1' \wedge C_2 \cap FV(s_2) = C_3 = \varnothing) \\
&\qquad\quad \vee (\langle C_2, C_3\rangle \in SR_2' \wedge C_1 = C_2 \cap FV(s_2) = \varnothing)\ ); \\
&\langle \kappa(s_1), \kappa(s_3)\rangle \in SR';\ \text{and} \\
&\langle C_1, C_3\rangle, \langle C_1', C_3'\rangle \in SR' \implies (C_1 = C_1' \iff C_3 = C_3')
\end{aligned}$$

with SR'_1 and SR'_2 being the sets in items iii_1 and iii_2, respectively.
First we prove that $\langle s'_1, s_3^{(i,j)}, SR' \rangle \in R$. To do so we only need to prove that

$$\bigcup \{C_1 \mid \langle C_1, C_3 \rangle \in SR'\} \supseteq FV(s'_1) \cup \kappa(s'_1); \text{ and} \tag{A.3}$$

$$\bigcup \{C_3 \mid \langle C_1, C_3 \rangle \in SR'\} \supseteq FV(s_3^{(i,j)}) \cup \kappa(s_3^{(i,j)}). \tag{A.4}$$

We prove (A.3). (A.4) follows in a similar way. Suppose $C'_2 \cap FV(s_2^{(i)}) \neq \varnothing$. Then

$$\begin{array}{ll}
\langle C'_1, C'_2 \rangle \in SR'_1 & \\
\Longrightarrow & (\text{by iii}_1) \\
\quad \exists \langle C_1, C_2 \rangle \in SR_1.\ C'_2 \cap FV(s_2^{(i)}) = C_2 \cap FV(s_2^{(i)}) & \\
\Longrightarrow & (\text{by def. of } SR) \\
\quad \exists \langle C^\star_2, C_3 \rangle \in SR_2.\ C'_2 \cap FV(s_2^{(i)}) = C^\star_2 \cap FV(s_2^{(i)}) & \\
\Longrightarrow & (\text{by iii}_2) \\
\quad \exists \langle C^\star_2, C_3 \rangle \in SR_2, \langle C''_2, C'_3 \rangle \in SR'_2. & \\
\qquad\qquad C^\star_2 \cap FV(s_2^{(i)}) = C''_2 \cap FV(s_2^{(i)}) & \\
\Longrightarrow & (\text{by def. of } SR') \\
\quad \exists C'_3.\ \langle C'_1, C'_3 \rangle \in SR'. &
\end{array}$$

If $C_2 \cap FV(s_2^{(i)}) = \varnothing$ sometimes the previous reasoning applies. Otherwise, it immediately follows that $\langle C'_1, \varnothing \rangle \in SR'_1$ implies $\langle C'_1, \varnothing \rangle \in SR'$.
This previous reasoning implies

$$\bigcup \{C_1 \mid \langle C_1, C_2 \rangle \in SR'_1\} \subseteq \bigcup \{C_1 \mid \langle C_1, C_3 \rangle \in SR'\}.$$

Since R_1 is a symbolic bisimulation, $FV(s'_1) \cup \kappa(s'_1) \subseteq \bigcup \{C_1 \mid \langle C_1, C_2 \rangle \in SR'_1\}$ from which immediately follows (A.3).
To conclude, we need to prove the forward compatibility property:

$$\begin{array}{l}
\{\langle C_1 \cap FV(s'_1), C_3 \cap FV(s_3^{(i,j)}) \rangle \mid \langle C_1, C_3 \rangle \in SR'\} - \{\langle \varnothing, \varnothing \rangle\} = \\
\qquad \{\langle C_1 \cap FV(s'_1), C_3 \cap FV(s_3^{(i,j)}) \rangle \mid \langle C_1, C_3 \rangle \in SR\} - \{\langle \varnothing, \varnothing \rangle\}.
\end{array}$$

We proceed as follows,

$$\begin{array}{ll}
\langle C'_1, C'_3 \rangle \in & \\
\qquad \{\langle C_1 \cap FV(s'_1), C_3 \cap FV(s_3^{(i,j)}) \rangle \mid \langle C_1, C_3 \rangle \in SR'\} - \{\langle \varnothing, \varnothing \rangle\} & \\
\Longleftrightarrow & (\text{by def. of } SR')
\end{array}$$

$$
\begin{aligned}
& \langle C_1', C_3' \rangle \neq \langle \varnothing, \varnothing \rangle \\
& \wedge\ \exists C_1, C_2, C_2^\star, C_3. \\
& \qquad (\quad (\langle C_1, C_2 \rangle \in SR_1' \wedge \langle C_2^\star, C_3 \rangle \in SR_2' \\
& \qquad\qquad\qquad \wedge\ C_2 \cap FV(s_2^{(i)}) = C_2^\star \cap FV(s_2)\,) \\
& \qquad\ \vee (\langle C_1, C_2 \rangle \in SR_1' \wedge C_2 \cap FV(s_2^{(i)}) = C_3 = \varnothing) \\
& \qquad\ \vee (\langle C_2^\star, C_3 \rangle \in SR_2' \wedge C_1 = C_2^\star \cap FV(s_2^{(i)}) = \varnothing) \quad) \\
& \qquad \wedge\ C_1' = C_1 \cap FV(s_1') \\
& \qquad \wedge\ C_3' = C_3 \cap FV(s_3^{(i,j)}) \\
\iff & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by iii}_1\text{ and iii}_2\text{)} \\
& \langle C_1', C_3' \rangle \neq \langle \varnothing, \varnothing \rangle \\
& \wedge\ \exists C_1, C_2, C_2^\star, C_3. \\
& \qquad (\quad (\langle C_1, C_2 \rangle \in SR_1 \wedge \langle C_2^\star, C_3 \rangle \in SR_2 \\
& \qquad\qquad\qquad \wedge\ C_2 \cap FV(s_2^{(i)}) = C_2^\star \cap FV(s_2)\,) \\
& \qquad\ \vee (\langle C_1, C_2 \rangle \in SR_1 \wedge C_2 \cap FV(s_2^{(i)}) = C_3 = \varnothing) \\
& \qquad\ \vee (\langle C_2^\star, C_3 \rangle \in SR_2 \wedge C_1 = C_2^\star \cap FV(s_2^{(i)}) = \varnothing) \quad) \\
& \qquad \wedge\ C_1' = C_1 \cap FV(s_1') \\
& \qquad \wedge\ C_3' = C_3 \cap FV(s_3^{(i,j)}) \\
\iff & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by def. of } SR\text{)} \\
& \langle C_1', C_3' \rangle \in \\
& \quad \{\langle C_1 \cap FV(s_1'), C_3 \cap FV(s_3^{(i,j)}) \rangle \mid \langle C_1, C_3 \rangle \in SR\} - \{\langle \varnothing, \varnothing \rangle\}.
\end{aligned}
$$

6. This case is symmetric to the previous one. *Q.E.D.*

A.2 Proof of Theorem 4.20

Lemma A.2. *Let $R_{\&}$ be a symbolic bisimulation. The relation*

$$
\begin{aligned}
R_t \stackrel{\text{def}}{=} \big\{ \langle (s_1, v_1), (s_2, v_2) \rangle \mid\ & \langle s_1, s_2, SR \rangle \in R_{\&} \\
& \wedge\ \forall \langle C_1, C_2 \rangle \in SR. \\
& \quad (\ \forall x, y \in C_1 \cap FV(s_1).\ \ v_1(x) = v_1(y) \\
& \quad \wedge \forall x, y \in C_2 \cap FV(s_2).\ \ v_2(x) = v_2(y) \\
& \quad \wedge \forall x \in C_1 \cap FV(s_1), y \in C_2 \cap FV(s_2).\ \ v_1(x) = v_2(y)\ )\ \big\}^{\mathsf{s}}
\end{aligned}
\tag{A.5}
$$

is a timed bisimulation.

Proof. First, notice that R_t is symmetry closed. We check that R_t satisfies the transfer properties in Definition 3.3. So, take $\langle (s_1, v_1), (s_2, v_2) \rangle \in R_t$. By definition of R_t, we have

that there is a $\langle s_1, s_2, SR\rangle \in R_{\&}$ or $\langle s_2, s_1, SR\rangle \in R_{\&}$ satisfying the conditions in (A.5). We will assume that $\langle s_1, s_2, SR\rangle \in R_{\&}$; the other case follows a similar prove. Before proceeding, we make an observation that will be of great help in the rest of the proof. For a given $d \in \mathbb{R}_{\geq 0}$, define v_{SR} such that

$$\forall x \in FV(s_1) \cup \kappa(s_1).\ (v_1[\kappa(s_1) \hookleftarrow 0] + d)(x) = v_{SR}(\xi^1_{SR}(x)) \tag{A.6}$$
$$\forall x \in FV(s_2) \cup \kappa(s_2).\ (v_2[\kappa(s_2) \hookleftarrow 0] + d)(x) = v_{SR}(\xi^2_{SR}(x)). \tag{A.7}$$

It is not difficult to check that v_{SR} is indeed well defined. We proceed by checking the transfer properties.

1. Suppose $(s_1, v_1) \xrightarrow{a(d)} (s_1', v_1')$. By Definition 4.6

$$s_1 \xrightarrow{a,\phi} s_1', \quad \models (v_1[\kappa(s_1) \hookleftarrow 0] + d)(\phi \wedge \iota(s_1)), \quad \text{and} \quad v_1' = v_1[\kappa(s_1) \hookleftarrow 0] + d. \tag{A.8}$$

Since $\langle s_1, s_2, SR\rangle \in R_{\&}$ then either

(a) $\models \neg(\xi^1_{SR}(\iota(s_1) \wedge \phi_1))$, or

(b) there are $s_2^{(1)}, \ldots, s_2^{(n)} \in \mathcal{S}$, $n \geq 1$ such that

i. $s_2 \xrightarrow{a,\phi_2^{(i)}} s_2^{(i)}$ for all $i = 1, \ldots, n$;

ii. $\models \xi^1_{SR}(\iota(s_1) \wedge \phi_1) \Rightarrow \bigvee_i \xi^2_{SR}(\iota(s_2) \wedge \phi_2^{(i)})$; and

iii. for all i, there exists SR' such that $\langle s_1', s_2^{(i)}, SR'\rangle \in R$ and

$$\{\langle C_1 \cap FV(s_1'), C_2 \cap FV(s_2^{(i)})\rangle \mid \langle C_1, C_2\rangle \in SR'\} - \{\langle \varnothing, \varnothing\rangle\} = \{\langle C_1 \cap FV(s_1'), C_2 \cap FV(s_2^{(i)})\rangle \mid \langle C_1, C_2\rangle \in SR\} - \{\langle \varnothing, \varnothing\rangle\}.$$

Because of (A.8) and (A.6), it follows that (a) does not hold and hence item (b) must hold. Because of ii., (A.6), and (A.7)

$$\models (v_1[\kappa(s_1) \hookleftarrow 0] + d)(\iota(s_1) \wedge \phi_1) \Rightarrow \bigvee_i (v_2[\kappa(s_2) \hookleftarrow 0] + d)(\iota(s_2) \wedge \phi_2^{(i)}).$$

Because of (A.8), there must be a particular i such that

$$\models (v_2[\kappa(s_2) \hookleftarrow 0] + d)(\iota(s_2) \wedge \phi_2^{(i)}).$$

So, because of item i. and Definition 4.6,

$$(s_2, v_2) \xrightarrow{a(d)} (s_2^{(i)}, v_2[\kappa(s_2) \hookleftarrow 0] + d).$$

It remains to show that $\langle (s_1', v_1[\kappa(s_1) \hookleftarrow 0] + d), (s_2^{(i)}, v_2[\kappa(s_2) \hookleftarrow 0] + d)\rangle \in R_t$. But this reduces to check that SR' in iii. satisfies properties in (A.5). Let $\langle C_1', C_2'\rangle \in SR'$.

We only prove the case in which $x \in C_1' \cap FV(s_1')$ and $y \in C_2' \cap FV(s_2^{(i)})$. The other cases follow in a similar way.

$$
\begin{array}{ll}
& x \in C_1' \cap FV(s_1') \quad \wedge \quad y \in C_2' \cap FV(s_2^{(i)}) \\
\Longrightarrow & \text{(by item iii.)} \\
& \exists \langle C_1, C_2 \rangle \in SR.\ x \in C_1 \cap FV(s_1') \quad \wedge \quad y \in C_2 \cap FV(s_2^{(i)}) \\
\Longrightarrow & \text{(because } FV(s_1') \subseteq FV(s_1) \cup \kappa(s_1) \text{ and } FV(s_2^{(i)}) \subseteq FV(s_2) \cup \kappa(s_2)) \\
& \exists \langle C_1, C_2 \rangle \in SR.\ x \in C_1 \cap (FV(s_1) \cup \kappa(s_1)) \quad \wedge \quad y \in C_2 \cap FV(s_2)(\cup \kappa(s_2)) \\
\Longrightarrow & \text{(because } \langle \kappa(s_1), \kappa(s_2) \rangle \in SR) \\
& \exists \langle C_1, C_2 \rangle \in SR.\ (x \in C_1 \cap FV(s_1) \quad \wedge \quad y \in C_2 \cap FV(s_2)) \\
& \qquad\qquad \vee \quad (x \in C_1 \cap \kappa(s_1) \quad \wedge \quad y \in C_2 \cap \kappa(s_2)) \\
\Longrightarrow & \text{(by (A.5) and def. of } v[C \hookleftarrow 0] \text{ and } v + d) \\
& (v_1[\kappa(s_1) \hookleftarrow 0] + d)(x) = (v_2[\kappa(s_2) \hookleftarrow 0] + d)(y).
\end{array}
$$

2. We only prove that $\mathcal{U}_d(s_1, v_1)$ implies $\mathcal{U}_d(s_2, v_2)$.

$$
\begin{array}{ll}
& \mathcal{U}_d(s_1, v_1) \\
\Longrightarrow & \text{(by Definition 4.6)} \\
& \models (v_1[\kappa(s_1) \hookleftarrow 0] + d)(\iota(s_1)) \qquad (*) \\
\Longrightarrow & \text{(because } \langle s_1, s_2, SR \rangle \in R_{\&}) \\
& \models \xi^1_{SR}(\iota(s_1)) \Leftrightarrow \xi^2_{SR}(\iota(s_2)) \\
\Longrightarrow & \text{(because of (A.6) and (A.7))} \\
& \models (v_1[\kappa(s_1) \hookleftarrow 0] + d)(\iota(s_1)) \Leftrightarrow (v_2[\kappa(s_2) \hookleftarrow 0] + d)(\iota(s_2)) \\
\Longrightarrow & \text{(because of (*))} \\
& \models (v_2[\kappa(s_2) \hookleftarrow 0] + d)(\iota(s_2)) \\
\Longrightarrow & \text{(by Definition 4.6)} \\
& \mathcal{U}_d(s_2, v_2).
\end{array}
$$

Q.E.D.

Theorem 4.20. *Let s_1 and s_2 be two locations in a timed automaton TA. If $s_1 \sim_{\&} s_2$ then $s_1 \sim_t s_2$.*

Proof. The proof follows from Lemma A.2. It only remains to notice that if *SR* satisfies conditions (a)–(c) in Definition 4.16, then any $v_1 = v_2$ satisfies conditions in R_t. *Q.E.D.*

Appendix B

Proofs from Chapter 5

B.1 Alternative Definitions

Tables B.1, B.2, and B.3 give definitions for fv, κ, and ι in terms of proof systems instead of sets defined as fixed points (see Chapter 5). In particular, the rules of κ and ι are quite helpful to prove that the structured operational semantics of ♡ is in *path* format. As a matter of fact, notice that we can observe $x \in \kappa(\_)$ and $\psi = \iota(\_)$ as predicates on ♡ terms.

$$\frac{x \in var(\phi)}{x \in fv(\phi \mapsto p)} \qquad \frac{x \in var(\psi)}{x \in fv(\psi \gg p)} \qquad \frac{x \in fv(p) - C}{x \in fv(\{\!|C|\!\}\, p)} \qquad \frac{x \in fv(p) \quad X = p}{x \in fv(X)}$$

$$\frac{x \in fv(p) \quad \mathsf{op} \in \{a;\_, \phi \mapsto \_, \psi \gg \_, \_[f]\}}{x \in fv(\mathsf{op}(p))} \qquad \frac{x \in fv(p) \quad \oplus \in \{+, ||_A, \between_A, |_A\}}{x \in fv(p \oplus q) \quad x \in fv(q \oplus p)}$$

Table B.1: Alternative definition of fv in ♡

$$\frac{x \in C}{x \in \kappa(\{\!|C|\!\}\, p)} \qquad \frac{x \in \kappa(p) \quad \mathsf{op} \in \{\phi \mapsto \_, \{\!|C|\!\}\, \_, \psi \gg \_, \_[f]\}}{x \in \kappa(\mathsf{op}(p))}$$

$$\frac{x \in \kappa(p) \quad X = p}{x \in \kappa(X)} \qquad \frac{x \in \kappa(p) \quad \oplus \in \{+, ||_A, \between_A, |_A\}}{x \in \kappa(p \oplus q) \quad x \in \kappa(q \oplus p)}$$

Table B.2: Alternative definition of κ in ♡

$$\mathbf{tt} = \iota(\mathbf{0}) \qquad \mathbf{tt} = \iota(a;p) \qquad \frac{\psi' = \iota(p)}{\psi \wedge \psi' = \iota(\psi \gg p)}$$

$$\frac{\psi = \iota(p) \quad \psi' = \iota(q)}{\psi \vee \psi' = \iota(p+q)} \qquad \frac{\psi = \iota(p) \quad X = p}{\psi = \iota(X)}$$

$$\frac{\psi = \iota(p) \quad \mathsf{op} \in \{\phi \mapsto \_, \{\!|C|\!\}\,\_, \_[f]\}}{\psi = \iota(\mathsf{op}(p))} \qquad \frac{\psi = \iota(p) \quad \psi' = \iota(q) \quad \oplus \in \{||_A, \between_A, |_A\}}{\psi \wedge \psi' = \iota(p \oplus q)}$$

Table B.3: Alternative definition of ι in $\heartsuit$

B.2 Proofs of Theorems 5.16 and 5.17

Lemma B.1. *Let $p_1, p_2 \in \heartsuit$ and let ξ_1, ξ_2, ξ_1', and ξ_2' be substitutions such that for all $x \in fv(p_1)$ and $y \in fv(p_2)$, $\xi_1(x) = \xi_2(y) \implies \xi_1'(x) = \xi_2'(y)$. Then, $\xi_1(p_1) \simeq_\alpha \xi_2(p_2) \implies \xi_1'(p_1) \simeq_\alpha \xi_2'(p_2)$.*

Proof. The proof follows by induction in the depth of the proof tree of $\xi_1(p_1) \simeq_\alpha \xi_2(p_2)$ considering each syntactic case separately. We only give some few paradigmatic cases.

Case $\{\!|C|\!\}\,p$. In the following, we assume $C_1', C_2', C^\star \subseteq \mathcal{C}$ are sets of fresh clock names.

$$\begin{aligned}
&\xi_1(\{\!|C_1|\!\}\,p_1) \simeq_\alpha \xi_2(\{\!|C_2|\!\}\,p_2) \\
&\implies && \text{(by Def. 5.8, substitution)} \\
&\quad \{\!|C_1'|\!\}\,\xi_1\xi_{[C_1'/C_1]}(p_1) \simeq_\alpha \{\!|C_2'|\!\}\,\xi_2\xi_{[C_2'/C_2]}(p_2) \\
&\implies && \text{(by Def. 5.9, } \simeq_\alpha\text{)} \\
&\quad \xi_{[C^\star/C_1']}(\xi_1\xi_{[C_1'/C_1]}(p_1)) \simeq_\alpha \xi_{[C^\star/C_2']}\xi_2(\xi_{[C_2'/C_2]}(p_2)) \\
&\implies && \text{(by composition)} \\
&\quad \xi_1\xi_{[C^\star/C_1]}(p_1) \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}(p_2) \\
&\implies && \text{(by induction)} \\
&\quad \xi_1'\xi_{[C^\star/C_1]}(p_1) \simeq_\alpha \xi_2'\xi_{[C^\star/C_2]}(p_2) \\
&\implies && \text{(by composition)} \\
&\quad \xi_{[C^\star/C_1']}(\xi_1'(\xi_{[C_1'/C_1]}(p_1))) \simeq_\alpha \xi_{[C^\star/C_2']}(\xi_2'(\xi_{[C_2'/C_2]}(p_2))) \\
&\implies && \text{(by Def. 5.9, } \simeq_\alpha\text{)} \\
&\quad \{\!|C_1'|\!\}\,\xi_1'\xi_{[C_1'/C_1]}(p_1) \simeq_\alpha \{\!|C_2'|\!\}\,\xi_2'\xi_{[C_2'/C_2]}(p_2) \\
&\implies && \text{(by Def. 5.8, substitution)} \\
&\quad \xi_1'(\{\!|C_1|\!\}\,p_1) \simeq_\alpha \xi_2'(\{\!|C_2|\!\}\,p_2)
\end{aligned}$$

Case $\psi \gg p$.

$$
\begin{array}{ll}
\xi_1(\psi \gg p_1) \simeq_\alpha \xi_2(\psi \gg p_2) & \\
\Longrightarrow & \text{(by Def. 5.8, substitution)} \\
\quad \xi_1(\psi) \gg \xi_1(p_1) \simeq_\alpha \xi_2(\psi) \gg \xi_2(p_2) & \\
\Longrightarrow & \text{(by Def. 5.9, } \simeq_\alpha\text{)} \\
\quad \xi_1(\psi) = \xi_2(\psi) \text{ and } \xi_1(p_1) \simeq_\alpha \xi_2(p_2) & \\
\Longrightarrow & \text{(by condition on } \xi\text{'s and induction)} \\
\quad \xi'_1(\psi) = \xi'_2(\psi) \text{ and } \xi'_1(p_1) \simeq_\alpha \xi'_2(p_2) & \\
\Longrightarrow & \text{(by Def. 5.9, } \simeq_\alpha\text{)} \\
\quad \xi'_1(\psi) \gg \xi'_1(p_1) \simeq_\alpha \xi'_2(\psi) \gg \xi'_2(p_2) & \\
\Longrightarrow & \text{(by Def. 5.8, substitution)} \\
\quad \xi'_1(\psi \gg p_1) \simeq_\alpha \xi'_2(\psi \gg p_2) &
\end{array}
$$

Case $p \,||_A\, q$.

$$
\begin{array}{ll}
\xi_1(p_1 \,||_A\, q_1) \simeq_\alpha \xi_2(p_2 \,||_A\, q_2) & \\
\Longrightarrow & \text{(by Def. 5.8, substitution)} \\
\quad \xi_1(p_1) \,||_A\, \xi_1(q_1) \simeq_\alpha \xi_2(p_2) \,||_A\, \xi_2(q_2) & \\
\Longrightarrow & \text{(by Def. 5.9, } \simeq_\alpha\text{)} \\
\quad \xi_1(p_1) \simeq_\alpha \xi_2(p_2) \text{ and } \xi_1(q_1) \simeq_\alpha \xi_2(q_2) & \\
\Longrightarrow & \text{(by induction)} \\
\quad \xi'_1(p_1) \simeq_\alpha \xi'_2(p_2) \text{ and } \xi'_1(q_1) \simeq_\alpha \xi'_2(q_2) & \\
\Longrightarrow & \text{(by Def. 5.9, } \simeq_\alpha\text{)} \\
\quad \xi'_1(p_1) \,||_A\, \xi'_1(q_1) \simeq_\alpha \xi'_2(p_2) \,||_A\, \xi'_2(q_2) & \\
\Longrightarrow & \text{(by Def. 5.8, substitution)} \\
\quad \xi'_1(p_1 \,||_A\, q_1) \simeq_\alpha \xi'_2(p_2 \,||_A\, q_2) &
\end{array}
$$

Q.E.D.

Lemma B.2. *Let* $p_1, p_2 \in \heartsuit$*, let* ξ_1 *and* ξ_2 *be two substitutions, and let* $x^\star \in \mathcal{C}$ *be a fresh clock name. If* $\xi_1(p_1) \simeq_\alpha \xi_2(p_2)$ *then* $\xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\overline{\mathsf{ck}}(p_1)) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\overline{\mathsf{ck}}(p_2))$.

Proof. The proof follows by induction on the depth of the proof tree of $\xi_1(p_1) \simeq_\alpha \xi_2(p_2)$ considering each syntactic case separately. We only give case $\{\!|C|\!\}\, p$ which is the most difficult.

Case $\{\!|C|\!\}\, p$. In the following, we assume $C'_1, C'_2, C^\star \subseteq \mathcal{C}$ are sets of fresh clock names.

$$
\begin{array}{ll}
\xi_1(\{\!|C_1|\!\}\, p_1) \simeq_\alpha \xi_2(\{\!|C_2|\!\}\, p_2) & \\
\Longrightarrow & \text{(by Def. 5.8 (substitution), Def. 5.9 } (\simeq_\alpha)\text{, and composition)} \\
\quad \xi_1\xi_{[C^\star/C_1]}(p_1) \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}(p_2) &
\end{array}
$$

$$\Longrightarrow \qquad \text{(by induction)}$$
$$\xi_1\xi_{[C^\star/C_1]}\xi_{\{x^\star/\kappa(p_1)\}}(\overline{\mathsf{ck}}(p_1)) \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}\xi_{\{x^\star/\kappa(p_2)\}}(\overline{\mathsf{ck}}(p_2))$$
$$\Longrightarrow \qquad \text{(by Lemma B.1)}$$
$$\xi_1\xi_{\{x^\star/C_1\cup\kappa(p_1)\}}(\overline{\mathsf{ck}}(p_1)) \simeq_\alpha \xi_2\xi_{\{x^\star/C_2\cup\kappa(p_2)\}}(\overline{\mathsf{ck}}(p_2))$$
$$\Longrightarrow \qquad \text{(by Def 5.4 } (\overline{\mathsf{ck}})\text{, and Def. of } \kappa)$$
$$\xi_1\xi_{\{x^\star/\kappa(\{\!|C_1|\!\}\, p_1)\}}(\overline{\mathsf{ck}}(\{\!|C_1|\!\}\, p_1)) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(\{\!|C_2|\!\}\, p_2)\}}(\overline{\mathsf{ck}}(\{\!|C_2|\!\}\, p_2))$$

Q.E.D.

Lemma B.3. *Let $p_1, p_2 \in \heartsuit$ such that none of them has local conflict of variables. Let ξ_1 and ξ_2 be two substitutions such that $\xi_1(p) \simeq_\alpha \xi_2(p_2)$. Consider $\longrightarrow$ to be derived using rules in Table 5.3 or rule* **alpha** *(page 58). Then*

(*a*) $p_1 \xrightarrow{a,\phi_1} p_1'$ *implies that there are* $p_2' \in \heartsuit$ *and* $\phi_2 \in \Phi$*, such that*

$$p_2 \xrightarrow{a,\phi_2} p_2',$$
$$\xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\phi_1) = \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\phi_2), \quad \textit{and}$$
$$\xi_1\xi_{\{x^\star/\kappa(p_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(p_2');$$

(*b*) $p_2 \xrightarrow{a,\phi_2} p_2'$ *implies that there are* $p_1' \in \heartsuit$ *and* $\phi_1 \in \Phi$*, such that*

$$p_1 \xrightarrow{a,\phi_1} p_1',$$
$$\xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\phi_1) = \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\phi_2), \quad \textit{and}$$
$$\xi_1\xi_{\{x^\star/\kappa(p_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(p_2');$$

(*c*) $\xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\iota(p_1)) = \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\iota(p_2))$.

Proof. We first prove property (*a*); it will need of (*c*) in some few cases. Property (*b*) follows from (*a*) by symmetry of $\simeq_\alpha$. The proof of property (*c*) is the last we report. In both proofs for (*a*) and (*c*) we proceed by induction on the depth of the proof tree of $\longrightarrow$, doing analysis by syntactic case.

Case (*a*). Cases $\mathbf{0}$ and $a;p$ are straightforward. We omit the proofs of cases $\psi \gg p$, $p[f]$, and X since they follow similar reasoning to the one of $\phi \mapsto p$. We also omit subcases $p \between_A q$ and $p \mid_A q$ since they are particular cases of $p \mid\mid_A q$.

Subcase $\phi \mapsto p$.

$$\xi_1(\phi_1 \mapsto p_1) \simeq_\alpha \xi_2(\phi_2 \mapsto p_2) \quad \wedge \quad \phi_1 \mapsto p_1 \xrightarrow{a,\phi_1\wedge\phi_1'} p_1'$$
$$\Longrightarrow \qquad \text{(by Def. 5.8 (substitution), Def. 5.9 } (\simeq_\alpha)\text{, and def. of } \longrightarrow)$$
$$\xi_1(\phi_1) = \xi_2(\phi_2) \quad \wedge \quad \xi_1(p_1) \simeq_\alpha \xi_2(p_2) \quad \wedge \quad p_1 \xrightarrow{a,\phi_1'} p_1'$$

$\Longrightarrow$ (by induction)

$\xi_1(\phi_1) = \xi_2(\phi_2) \quad \wedge \quad p_2 \xrightarrow{a,\phi_2'} p_2'$
$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\phi_1') = \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\phi_2')$
$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(p_2')$

$\Longrightarrow$ (since $\neg\mathsf{cv}(p_1)$ and $\neg\mathsf{cv}(p_2)$, applying def. of κ and def. of $\rightarrow$)

$\phi_2 \mapsto p_2 \xrightarrow{a,\phi_2\wedge\phi_2'} p_2'$
$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(\phi_1\mapsto p_1)\}}(\phi_1 \wedge \phi_1') = \xi_2\xi_{\{x^\star/\kappa(\phi_2\mapsto p_2)\}}(\phi_2 \wedge \phi_2')$
$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(\phi_1\mapsto p_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(\phi_2\mapsto p_2)\}}(p_2')$

Subcase $\{\!|C|\!\}\, p$. In the following we assume that $C^\star \subseteq \mathcal{C}$ is a set of fresh clock names.

$\xi_1(\{\!|C_1|\!\}\, p_1) \simeq_\alpha \xi_2(\{\!|C_2|\!\}\, p_2) \quad \wedge \quad \{\!|C_1|\!\}\, p_1 \xrightarrow{a,\phi_1} p_1'$

$\Longrightarrow$ (by Def. 5.8 (substitution), Def. 5.9 ($\simeq_\alpha$), composition, and def. of $\rightarrow$)

$\xi_1\xi_{[C^\star/C_1]}(p_1) \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}(p_2) \quad \wedge \quad p_1 \xrightarrow{a,\phi_1} p_1'$

$\Longrightarrow$ (by induction)

$p_2 \xrightarrow{a,\phi_2} p_2'$
$\wedge \quad \xi_1\xi_{[C^\star/C_1]}\xi_{\{x^\star/\kappa(p_1)\}}(\phi_1) = \xi_2\xi_{[C^\star/C_2]}\xi_{\{x^\star/\kappa(p_2)\}}(\phi_2)$
$\wedge \quad \xi_1\xi_{[C^\star/C_1]}\xi_{\{x^\star/\kappa(p_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}\xi_{\{x^\star/\kappa(p_2)\}}(p_2')$

$\Longrightarrow$ (by Lemma B.1)

$p_2 \xrightarrow{a,\phi_2} p_2'$
$\wedge \quad \xi_1\xi_{\{x^\star/C_1\cup\kappa(p_1)\}}(\phi_1) = \xi_2\xi_{\{x^\star/C_2\cup\kappa(p_2)\}}(\phi_2)$
$\wedge \quad \xi_1\xi_{\{x^\star/C_1\cup\kappa(p_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/C_2\cup\kappa(p_2)\}}(p_2')$

$\Longrightarrow$ (by def. of κ and def. of $\rightarrow$)

$\{\!|C_2|\!\}\, p_2 \xrightarrow{a,\phi_2} p_2'$
$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(\{\!|C_1|\!\}\, p_1)\}}(\phi_1) = \xi_2\xi_{\{x^\star/\kappa(\{\!|C_2|\!\}\, p_2)\}}(\phi_2)$
$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(\{\!|C_1|\!\}\, p_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(\{\!|C_2|\!\}\, p_2)\}}(p_2')$

Subcase $p + q$.

$p_1 + q_1 \xrightarrow{a,\psi_1\wedge\phi_1} p_1'$

$\Longrightarrow$ (by def. of $\rightarrow$)

$(p_1 \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad \psi_1 = \iota(p_1))$ (*)
$\vee \quad (q_1 \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad \psi_1 = \iota(q_1))$ (**)

Assume (*) is the case. Case (**) proceeds in a similar way. Then

$$\xi_1(p_1 + q_1) \simeq_\alpha \xi_2(p_2 + q_2) \quad \wedge \quad p_1 \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad \psi_1 = \iota(p_1)$$
$$\Longrightarrow \qquad \text{(by Def. 5.8 (substitution), and Def. 5.9 } (\simeq_\alpha))$$
$$\xi_1(p_1) \simeq_\alpha \xi_2(p_2) \quad \wedge \quad p_1 \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad \psi_1 = \iota(p_1)$$
$$\Longrightarrow \qquad \text{(by induction and item (c) of this lemma)}$$
$$p_2 \xrightarrow{a,\phi_2} p_2'$$
$$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\phi_1) = \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\phi_2)$$
$$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(p_2')$$
$$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\iota(p_1)) = \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\iota(p_2))$$
$$\Longrightarrow \qquad \text{(since } \neg\mathsf{cv}(p_1 + q_1) \text{ and } \neg\mathsf{cv}(p_2 + q_2), \text{ applying def. of } \kappa)$$
$$p_2 \xrightarrow{a,\phi_2} p_2'$$
$$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1+q_1)\}}(\phi_1) = \xi_2\xi_{\{x^\star/\kappa(p_2+q_2)\}}(\phi_2)$$
$$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1+q_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2+q_2)\}}(p_2')$$
$$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1+q_1)\}}(\iota(p_1)) = \xi_2\xi_{\{x^\star/\kappa(p_2+q_2)\}}(\iota(p_2))$$
$$\Longrightarrow \qquad \text{(by def. of } \longrightarrow \text{ and calculations, taking } \psi_2 = \iota(p_2))$$
$$p_2 + q_2 \xrightarrow{a,\psi_2\wedge\phi_2} p_2'$$
$$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1+q_1)\}}(\psi_1 \wedge \phi_1) = \xi_2\xi_{\{x^\star/\kappa(p_2+q_2)\}}(\psi_2 \wedge \phi_2)$$
$$\wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1+q_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2+q_2)\}}(p_2')$$

Subcase $p \,||_A\, q$.

$$p_1 \,||_A\, q_1 \xrightarrow{a,\phi_1} p_1' \,||_A\, q_1'$$
$$\Longrightarrow \qquad \text{(by def. of } \longrightarrow)$$
$$(p_1 \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad a \notin A \quad \wedge \quad q_1' = \overline{\mathsf{ck}}(q_1')) \qquad (*)$$
$$\vee \quad (q_1 \xrightarrow{a,\phi_1} q_1' \quad \wedge \quad a \notin A \quad \wedge \quad p_1' = \overline{\mathsf{ck}}(p_1')) \qquad (**)$$
$$\vee \quad (p_1 \xrightarrow{a,\phi_1'} p_1' \quad \wedge \quad q_1 \xrightarrow{a,\phi_1''} q_1' \quad \wedge \quad \phi_1 \equiv \phi_1' \wedge \phi_1'' \quad \wedge \quad a \in A) \qquad (***)$$

Assume (*) is the case. Case (**) proceeds in a similar way and case (***) follows the same lines but it is simpler. Then

$$\xi_1(p_1 \,||_A\, q_1) \simeq_\alpha \xi_2(p_2 \,||_A\, q_2) \quad \wedge \quad p_1 \xrightarrow{a,\phi_1} p_1'$$
$$\Longrightarrow \qquad \text{(by Def. 5.8 (substitution), and Def. 5.9 } (\simeq_\alpha))$$
$$\xi_1(p_1) \simeq_\alpha \xi_2(p_2) \quad \wedge \quad \xi_1(q_1) \simeq_\alpha \xi_2(q_2) \quad \wedge \quad p_1 \xrightarrow{a,\phi_1} p_1'$$
$$\Longrightarrow \qquad \text{(by Lemma B.2 and induction)}$$

$$
\begin{array}{ll}
& \xi_1\xi_{\{x^\star/\kappa(q_1)\}}(\overline{\mathsf{ck}}(q_1)) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(q_2)\}}(\overline{\mathsf{ck}}(q_2)) \\
\wedge & p_2 \xrightarrow{a,\phi_2} p_2' \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\phi_1) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\phi_2) \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(p_2') \\
\Longrightarrow & \qquad \text{(since } \neg\mathsf{cv}(p_1 \,||_A\, q_1) \text{ and } \neg\mathsf{cv}(p_2 \,||_A\, q_2)\text{, applying def. of } \kappa) \\
& \xi_1\xi_{\{x^\star/\kappa(p_1||_A q_1)\}}(\overline{\mathsf{ck}}(q_1)) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2||_A q_2)\}}(\overline{\mathsf{ck}}(q_2)) \\
\wedge & p_2 \xrightarrow{a,\phi_2} p_2' \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1||_A q_1)\}}(\phi_1) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2||_A q_2)\}}(\phi_2) \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1||_A q_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2||_A q_2)\}}(p_2') \\
\Longrightarrow & \qquad \text{(by Def. 5.9 } (\simeq_\alpha)\text{, and Def. 5.8 (substitution), and def. of } \rightarrow) \\
& p_2 \,||_A\, q_2 \xrightarrow{a,\phi_2} p_2' \,||_A\, \overline{\mathsf{ck}}(q_2) \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1||_A q_1)\}}(\phi_1) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2||_A q_2)\}}(\phi_2) \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1||_A q_1)\}}(p_1' \,||_A\, \overline{\mathsf{ck}}(q_1)) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2||_A q_2)\}}(p_2' \,||_A\, \overline{\mathsf{ck}}(q_2))
\end{array}
$$

Subcase **alpha**. Let $p_1 \xrightarrow{a,\phi_1} p_1'$ and suppose it was obtained by applying rule **alpha**. Then, for some p_1'', $p_1 \xrightarrow{a,\phi_1} p_1''$, $p_1'' \simeq_\alpha p_1'$, and $\neg\mathsf{cv}(p_1')$. Hence, we proceed,

$$
\begin{array}{ll}
& \xi_1(p_1) \simeq_\alpha \xi_2(p_2) \quad \wedge \quad p_1 \xrightarrow{a,\phi_1} p_1'' \quad \wedge \quad p_1'' \simeq_\alpha p_1' \\
\Longrightarrow & \qquad \text{(by induction)} \\
& p_2 \xrightarrow{a,\phi_2} p_2' \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\phi_1) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\phi_2) \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(p_1'') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(p_2') \\
\wedge & p_1'' \simeq_\alpha p_1' \\
\Longrightarrow & \qquad \text{(by Lemma B.1)} \\
& p_2 \xrightarrow{a,\phi_2} p_2' \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\phi_1) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\phi_2) \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(p_1'') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(p_2') \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(p_1'') \simeq_\alpha \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(p_1') \\
\Longrightarrow & \qquad \text{(by transitivity of } \simeq_\alpha) \\
& p_2 \xrightarrow{a,\phi_2} p_2' \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\phi_1) \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\phi_2) \\
\wedge & \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(p_1') \simeq_\alpha \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(p_2')
\end{array}
$$

Case (c). We only give the proof for cases $\psi \gg p$, $\{\!\{C\}\!\}\, p$, and $p \,||_A\, q$, since cases $\mathbf{0}$ and $a; p$ are straightforward and for the other operators the reasoning is similar.

Subcase $\psi \gg p$.

$$
\begin{array}{ll}
& \xi_1(\psi_1 \gg p_1) \simeq_\alpha \xi_2(\psi_2 \gg p_2) \\
\Longrightarrow & \text{(by Def. 5.8 (substitution) and Def. 5.9 } (\simeq_\alpha)) \\
& \xi_1(\psi_1) = \xi_2(\psi_2) \quad \wedge \quad \xi_1(p_1) \simeq_\alpha \xi_2(p_2) \\
\Longrightarrow & \text{(by induction)} \\
& \xi_1(\psi_1) = \xi_2(\psi_2) \quad \wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\iota(p_1)) = \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\iota(p_2)) \\
\Longrightarrow & \text{(since } \neg\mathsf{cv}(\psi_1 \gg p_1) \text{ and } \neg\mathsf{cv}(\psi_2 \gg p_2), \text{ applying def. of } \kappa) \\
& \xi_1\xi_{\{x^\star/\kappa(\psi_1 \gg p_1)\}}(\psi_1) = \xi_2\xi_{\{x^\star/\kappa(\psi_2 \gg p_2)\}}(\psi_2) \\
& \wedge \quad \xi_1\xi_{\{x^\star/\kappa(\psi_1 \gg p_1)\}}(\iota(p_1)) = \xi_2\xi_{\{x^\star/\kappa(\psi_2 \gg p_2)\}}(\iota(p_2)) \\
\Longrightarrow & \text{(by calculations and def. of } \iota) \\
& \xi_1\xi_{\{x^\star/\kappa(\psi_1 \gg p_1)\}}(\iota(\psi_1 \gg p_1)) = \xi_2\xi_{\{x^\star/\kappa(\psi_2 \gg p_2)\}}(\iota(\psi_2 \gg p_2))
\end{array}
$$

Subcase $\{\!|C|\!\}\, p$. In the following we assume that $C^\star \subseteq \mathcal{C}$ is a set of fresh clock names.

$$
\begin{array}{ll}
& \xi_1(\{\!|C_1|\!\}\, p_1) \simeq_\alpha \xi_2(\{\!|C_2|\!\}\, p_2) \\
\Longrightarrow & \text{(by Def. 5.8 (substitution), Def. 5.9 } (\simeq_\alpha), \text{ and composition)} \\
& \xi_1\xi_{[C^\star/C_1]}(p_1) \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}(p_2) \\
\Longrightarrow & \text{(by induction)} \\
& \xi_1\xi_{[C^\star/C_1]}\xi_{\{x^\star/\kappa(p_1)\}}(\iota(p_1)) = \xi_2\xi_{[C^\star/C_2]}\xi_{\{x^\star/\kappa(p_2)\}}(\iota(p_2)) \\
\Longrightarrow & \text{(by Lemma B.1)} \\
& \xi_1\xi_{\{x^\star/C_1 \cup \kappa(p_1)\}}(\iota(p_1)) = \xi_2\xi_{\{x^\star/C_2 \cup \kappa(p_2)\}}(\iota(p_2)) \\
\Longrightarrow & \text{(by def. of } \kappa \text{ and } \iota) \\
& \xi_1\xi_{\{x^\star/\kappa(\{\!|C_1|\!\}\, p_1)\}}(\iota(\{\!|C_1|\!\}\, p_1)) = \xi_2\xi_{\{x^\star/\kappa(\{\!|C_2|\!\}\, p_2)\}}(\iota(\{\!|C_2|\!\}\, p_2))
\end{array}
$$

Subcase $p \,||_A\, q$.

$$
\begin{array}{ll}
& \xi_1(p_1 \,||_A\, q_1) \simeq_\alpha \xi_2(p_2 \,||_A\, q_2) \\
\Longrightarrow & \text{(by Def. 5.8 (substitution) and Def. 5.9 } (\simeq_\alpha)) \\
& \xi_1(p_1) \simeq_\alpha \xi_2(p_2) \quad \wedge \quad \xi_1(q_1) \simeq_\alpha \xi_2(q_2) \\
\Longrightarrow & \text{(by induction)} \\
& \xi_1\xi_{\{x^\star/\kappa(p_1)\}}(\iota(p_1)) = \xi_2\xi_{\{x^\star/\kappa(p_2)\}}(\iota(p_2)) \\
& \wedge \quad \xi_1\xi_{\{x^\star/\kappa(q_1)\}}(\iota(q_1)) = \xi_2\xi_{\{x^\star/\kappa(q_2)\}}(\iota(q_2)) \\
\Longrightarrow & \text{(since } \neg\mathsf{cv}(p_1 \,||_A\, q_1) \text{ and } \neg\mathsf{cv}(p_2 \,||_A\, q_2), \text{ applying def. of } \kappa) \\
& \xi_1\xi_{\{x^\star/\kappa(p_1 ||_A q_1)\}}(\iota(p_1)) = \xi_2\xi_{\{x^\star/\kappa(p_2 ||_A q_2)\}}(\iota(p_2)) \\
& \wedge \quad \xi_1\xi_{\{x^\star/\kappa(p_1 ||_A q_1)\}}(\iota(q_1)) = \xi_2\xi_{\{x^\star/\kappa(p_2 ||_A q_2)\}}(\iota(q_2)) \\
\Longrightarrow & \text{(by calculations and def. of } \iota) \\
& \xi_1\xi_{\{x^\star/\kappa(p_1 ||_A q_1)\}}(\iota(p_1 \,||_A\, q_1)) = \xi_2\xi_{\{x^\star/\kappa(p_2 ||_A q_2)\}}(\iota(p_2 \,||_A\, q_2)) \qquad \textit{Q.E.D.}
\end{array}
$$

Lemma B.4. *Let $p \in \heartsuit^\alpha$. Then $FV(p) \subseteq fv(p)$.*

Proof sketch. By induction on the depth of the proof tree of $\longrightarrow$ and ι, the following can be proved

(*a*) $p \xrightarrow{a,\phi} p'$ and $x \in (var(\phi) \cup fv(p')) - \kappa(p)$ then $x \in fv(p)$, and

(*b*) $x \in var(\iota(p)) - \kappa(p)$ then $x \in fv(p)$.

From here, it follows that

$$fv(p) \supseteq \Big(\{x \mid p \xrightarrow{a,\phi} p' \wedge x \in var(\phi) \cup fv(p')\} \cup var(\iota(p))\Big) - \kappa(p).$$

Since $FV(p)$ is the smallest set satisfying the same inclusion (see Definition 4.14), and hence also equality, the lemma follows. *Q.E.D.*

Theorem 5.16. *Let $p, q \in \heartsuit^\alpha$; so, none of them has local conflict of variables and moreover they are α-conflict free. If $p \simeq_\alpha q$ then $p \sim_\& q$ where p and q should be interpreted here as states in $TA(\heartsuit^\alpha)$.*

Proof. Let $R_\&$ be the least relation satisfying the following rules

$$\frac{\begin{array}{c} p_1 \simeq_\alpha p_2 \qquad \neg\mathsf{cv}(p_1) \qquad \neg\mathsf{cv}(p_2) \\ SR = \{\langle \kappa(p_1), \kappa(p_2)\rangle\} \cup \{\langle \{x\} - \kappa(p_1), \{x\} - \kappa(p_2)\rangle \mid x \in fv(p_1) \cup fv(p_2)\} \end{array}}{\langle p_1, p_2, SR\rangle \in R_\&} \tag{B.1}$$

$$\frac{\begin{array}{c} \langle p_1, p_2, SR\rangle \in R_\& \qquad p_1 \xrightarrow{a,\phi_1}{}' p_1' \qquad p_2 \xrightarrow{a,\phi_2}{}' p_2' \\ \xi^1_{SR}(\phi_1) = \xi^2_{SR}(\phi_2) \qquad \xi^1_{SR}(p_1') \simeq_\alpha \xi^2_{SR}(p_2') \\ SR' = \{\langle \kappa(p_1'), \kappa(p_2')\rangle\} \cup \{\langle C_1 \cap fv(p_1'), C_2 \cap fv(p_2')\rangle \mid \langle C_1, C_2\rangle \in SR\} \end{array}}{\langle p_1', p_2', SR'\rangle \in R_\&} \tag{B.2}$$

where ξ^i_{SR} are as in Definition 4.15. Notice that tuples in $R_\&$ obtained from rule (B.1) satisfy properties (a), (b), and (c) in Definition 4.16. So, by proving that $R_\&$ is a symbolic bisimulation, the theorem follows.

First, we show the following claim.

Claim B.5. *$\langle p_1, p_2, SR\rangle \in R_\&$ implies $\xi^1_{SR}(p_1) \simeq_\alpha \xi^2_{SR}(p_2)$.*

Proof. Suppose $\langle p_1, p_2, SR\rangle \in R_\&$ is derived using rule (B.1). Then our claim follows by Lemma B.1. If it is derived using rule (B.2), there exists $\langle p_1', p_2', SR'\rangle \in R_\&$ such that $p_1' \xrightarrow{a,\phi_1}{}' p_1$, $p_2' \xrightarrow{a,\phi_2}{}' p_2$, $\xi^1_{SR'}(p_1) \simeq_\alpha \xi^2_{SR'}(p_2)$, and

$$SR = \{\langle \kappa(p_1), \kappa(p_2)\rangle\} \cup \{\langle C_1 \cap fv(p_1), C_2 \cap fv(p_2)\rangle \mid \langle C_1, C_2\rangle \in SR'\}.$$

So, we have that

$$
\begin{array}{l}
x \in fv(p_1) \quad \wedge \quad y \in fv(p_2) \quad \wedge \quad \xi^1_{SR'}(x) = \xi^2_{SR'}(y) \\
\Longrightarrow \hfill \text{(by Def. 4.15 } (\xi^i_{SR'})) \\
\quad x \in fv(p_1) \quad \wedge \quad y \in fv(p_2) \quad \wedge (\exists \langle C_1, C_2 \rangle \in SR'.\ x \in C_1 \wedge y \in C_2) \\
\Longrightarrow \\
\quad \exists \langle C_1, C_2 \rangle \in SR'.\ x \in C_1 \cap fv(p_1) \wedge y \in C_2 \cap fv(p_2) \\
\Longrightarrow \hfill \text{(by Def. 4.15 } (\xi^i_{SR}) \text{, since } \langle C_1 \cap fv(p_1), C_2 \cap fv(p_2) \rangle \in SR) \\
\quad \xi^1_{SR}(x) = \xi^2_{SR}(y)
\end{array}
$$

From this, and using Lemma B.1, it follows that $\xi^1_{SR'}(p_1) \simeq_\alpha \xi^2_{SR'}(p_2)$ implies $\xi^1_{SR}(p_1) \simeq_\alpha \xi^2_{SR}(p_2)$. *Q.E.D. (Claim)*

In the following we prove that $R_\&$ satisfies the transfer properties of Definition 4.16. So suppose $\langle p_1, p_2, SR \rangle \in R_\&$

1. This property follows straightforwardly by induction on the depth of the proof of $\langle p_1, p_2, SR \rangle \in R_\&$.

2. By induction on the depth of the proof of $\langle p_1, p_2, SR \rangle \in R_\&$, it can be proven that, for $i = 1, 2$,

   $$\bigcup \{ C_i \mid \langle C_1, C_2 \rangle \in SR \} \supseteq fv(p_i) \cup \kappa(p_i).$$

   By Lemma B.4, the property follows.

3. Immediate, by definition of $R_\&$

4. By Lemma B.3.(c), since $\xi^i_{SR} \xi_{\{\xi^i_{SR}(x_i)/\kappa(p_i)\}} = \xi^i_{SR}$ for any $x_i \in \kappa(p_i)$.

5. Suppose $p_1 \xrightarrow{a,\phi_1}{}' p'_1$, then $p_1 \xrightarrow{a,\phi_1} p'_1$. By Claim B.5 $\xi^1_{SR}(p_1) \simeq_\alpha \xi^2_{SR}(p_2)$. Hence, by Lemma B.3.($a$), there are $p'_2 \in \heartsuit$ and $\phi_2 \in \Phi$, such that

   $$
   \begin{array}{l}
   p_2 \xrightarrow{a,\phi_2} p''_2, \\
   \xi^1_{SR} \xi_{\{x^\star/\kappa(p_1)\}}(\phi_1) = \xi^2_{SR} \xi_{\{x^\star/\kappa(p_2)\}}(\phi_2), \quad \text{and} \\
   \xi^1_{SR} \xi_{\{x^\star/\kappa(p_1)\}}(p'_1) \simeq_\alpha \xi^2_{SR} \xi_{\{x^\star/\kappa(p_2)\}}(p''_2).
   \end{array}
   $$

   Since p_2 is α-conflict free, there is a p'_2 such that $\neg \mathsf{cv}(p'_2)$, $p''_2 \simeq_\alpha p'_2$, and $p_2 \xrightarrow{a,\phi_2} p'_2$. By Lemma B.1, $\xi^2_{SR} \xi_{\{x^\star/\kappa(p_2)\}}(p''_2) \simeq_\alpha \xi^2_{SR} \xi_{\{x^\star/\kappa(p_2)\}}(p'_2)$ Choose $x^\star = \xi^i_{SR}(x_i)$ for any $i = 1, 2$, $x_i \in \kappa(p_i)$. Then

   $$p_2 \xrightarrow{a,\phi_2}{}' p'_2, \quad \xi^1_{SR}(\phi_1) = \xi^2_{SR}(\phi_2), \quad \text{and} \quad \xi^1_{SR}(p'_1) \simeq_\alpha \xi^2_{SR}(p'_2).$$

Thus $\models \xi^1_{SR}(\iota(p_1) \wedge \phi_1) \Rightarrow \xi^2_{SR}(\iota(p_2) \wedge \phi_2)$. Take SR' as in rule (B.2). Then $\langle p'_1, p'_2, SR' \rangle \in R_{\&}$. It remains to prove that

$$\begin{aligned}\{\langle C_1 \cap FV(p'_1), C_2 \cap FV(p'_2)\rangle \mid \langle C_1, C_2\rangle \in SR'\} - \{\langle \varnothing, \varnothing\rangle\} = \\ \{\langle C_1 \cap FV(p'_1), C_2 \cap FV(p'_2)\rangle \mid \langle C_1, C_2\rangle \in SR\} - \{\langle \varnothing, \varnothing\rangle\}.\end{aligned} \tag{B.3}$$

By definition of SR', it holds that

$$\begin{aligned}\{\langle C_1 \cap fv(p'_1), C_2 \cap fv(p'_2)\rangle \mid \langle C_1, C_2\rangle \in SR'\} - \{\langle \varnothing, \varnothing\rangle\} = \\ \{\langle C_1 \cap fv(p'_1), C_2 \cap fv(p'_2)\rangle \mid \langle C_1, C_2\rangle \in SR\} - \{\langle \varnothing, \varnothing\rangle\}.\end{aligned}$$

By Lemma B.4, the forward compatibility property (B.3) follows.

6. This case is analogous to the previous one. *Q.E.D.*

Theorem 5.17. *Let $p \in \heartsuit^{\mathsf{cf}}$; so, p is conflict free. Then, the rooted timed automata $(TA(\heartsuit^{\mathsf{cf}}), p)$ and $(TA(\heartsuit^{\alpha}), p)$ are symbolically bisimilar.*

Proof. Similar to the proof of Theorem 5.16 above. *Q.E.D.*

B.3 Proof of Theorem 5.24

We give the following lemmas without proof.

Lemma B.6. *If $p \xrightarrow{a,\phi}{}' p'$ then $fv(p') \subseteq fv(p) \cup \kappa(p)$. As a consequence, for all $v, v' \in \mathbf{V}$, $(p, v) \xrightarrow{a(d)} (p', v')$ implies $fv(p') \subseteq fv(p) \cup \kappa(p)$. Here, $\longrightarrow$ is the transition in $TTS(TA(\heartsuit^{\alpha}))$.*

Lemma B.7. *If $p \simeq_\alpha q$ then $p \sim_t q$ when they are interpreted in $TTS(\heartsuit^{bc})$.*

The proof of Lemma B.6 follow by induction on the depth of the proof tree. Lemma B.7 is proved in (D'Argenio and Brinksma, 1996a).

Theorem 5.24. *For every α-conflict free $p \in \heartsuit^{br}$, and for every closed valuation v_0, $\langle p, v_0\rangle \sim_t (p', v_0)$, where $p' \simeq_\alpha p$ and $\neg\mathsf{cv}(p)$. Here, $\langle p, v_0\rangle$ is a state in the timed transition system of the direct semantics $TTS(\heartsuit^{br})$, and (p', v_0) is a state in the "two steps" semantics $TTS(TA(\heartsuit^{\alpha}))$.*

Proof. We state that

$$R \stackrel{\text{def}}{=} \left\{\langle \langle p, v_1\rangle, (p, v_2)\rangle \mid p \in \heartsuit^{bc} \wedge \neg\mathsf{cv}(p) \wedge v_1 \upharpoonright fv(p) = v_2 \upharpoonright fv(p)\right\}^{\mathsf{s}},$$

with $v \upharpoonright C \stackrel{\text{def}}{=} v \cap (C \times \mathbb{R}_{\geq 0})$, is a timed bisimulation up to $\sim_t$. This suffices to prove the theorem.

We prove each transfer property in Definition 3.5 by induction on the depth of the proof tree by considering each syntactic case separately. Besides, we observe that symmetric cases can be proved following a converse reasoning.

Case 1.

Subcase **0**. Trivial

Subcase $a; p$. On the one hand, by Definition 5.23

$$\langle a; p, v_1\rangle \xrightarrow{a(d)} \langle p, v_1 + d\rangle.$$

On the other hand,

$$\begin{array}{ll}
a; p \xrightarrow{a,\mathbf{tt}} p & \text{(by Table 5.3)}\\
\Longrightarrow & \text{(by rule } \mathbf{alpha}\text{)}\\
\quad a; p \xrightarrow{a,\mathbf{tt}}{}' q \quad\wedge\quad p \simeq_\alpha q \quad\wedge\quad \neg\mathsf{cv}(q) & \\
\Longrightarrow & \text{(by Def. 4.6)}\\
\quad (a; p, v_2) \xrightarrow{a(d)} (q, v_2 + d) \quad\wedge\quad p \simeq_\alpha q \quad\wedge\quad \neg\mathsf{cv}(q) &
\end{array}$$

By Lemma B.7, $\langle p, v_1 + d\rangle \sim_t \langle q, v_1 + d\rangle$. Since $fv(a; p) = fv(p) = fv(q)$, we have that $(v_1 + d) \restriction fv(q) = (v_2 + d) \restriction fv(q)$. As a consequence $\langle\langle p, v_1 + d\rangle, (q, v_2 + d)\rangle \in (\sim_t \cup R)^*$.

Subcase $\phi \mapsto p$.

$$\begin{array}{ll}
\langle \phi \mapsto p, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle & \\
\Longrightarrow & \text{(by Def. 5.23)}\\
\quad \langle p, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle \quad\wedge\quad \models (v_1 + d)(\phi) & \\
\Longrightarrow & \text{(by induction)}\\
\quad (p, v_2) \xrightarrow{a(d)} (q, v_2') \quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \quad\wedge\quad \models (v_1 + d)(\phi) & \\
\Longrightarrow & \text{(by Def. 4.6, } v_2' = v_2[\kappa(p) \leftarrowtail 0] + d\text{)}\\
\quad p \xrightarrow{a,\phi'}{}' q \quad\wedge\quad \models v_2'(\phi' \wedge \iota(p)) & \\
\quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \quad\wedge\quad \models (v_1 + d)(\phi) & \\
\Longrightarrow & \text{(by rule } \mathbf{alpha}\text{)}\\
\quad p \xrightarrow{a,\phi'} q' \quad\wedge\quad q \simeq_\alpha q' \quad\wedge\quad \neg\mathsf{cv}(q) \quad\wedge\quad \models v_2'(\phi' \wedge \iota(p)) & \\
\quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \quad\wedge\quad \models (v_1 + d)(\phi) & \\
\Longrightarrow & \text{(by Table 5.3 and } v_1 \restriction fv(p) = v_2 \restriction fv(p)\text{)}\\
\quad \phi \mapsto p \xrightarrow{a,\phi\wedge\phi'} q' \quad\wedge\quad q \simeq_\alpha q' \quad\wedge\quad \neg\mathsf{cv}(q) \quad\wedge\quad \models v_2'(\phi' \wedge \iota(p)) & \\
\quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \quad\wedge\quad \models (v_2 + d)(\phi) & \\
\Longrightarrow & \text{(by rule } \mathbf{alpha}\text{, } \neg\mathsf{cv}(p)\text{, and calculations)}\\
\quad \phi \mapsto p \xrightarrow{a,\phi\wedge\phi'}{}' q \quad\wedge\quad \models v_2'(\phi \wedge \phi' \wedge \iota(p)) & \\
\quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* &
\end{array}$$

$$\Longrightarrow \qquad \text{(by Def. 4.6 and } v_2' = v_2[\kappa(\phi \mapsto p) \leftarrowtail 0] + d)$$

$$(\phi \mapsto p, v_2) \xrightarrow{a(d)} (q, v_2') \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$

Subcase $p_1 + p_2$.

$$\langle p_1 + p_2, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle$$

$$\Longrightarrow \qquad \text{(by Def. 5.23)}$$

$$\langle p_1, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle \qquad (*)$$

$$\vee \quad \langle p_2, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle \qquad (**)$$

Suppose (*) is the case. Case (**) can be proved in the same way.

$$\langle p_1, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle$$

$$\Longrightarrow \qquad \text{(by induction)}$$

$$(p_1, v_2) \xrightarrow{a(d)} (q, v_2') \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$

$$\Longrightarrow \qquad \text{(by Def. 4.6, } v_2' = v_2[\kappa(p) \leftarrowtail 0] + d)$$

$$p_1 \xrightarrow{a,\phi}{}' q \quad \wedge \quad \models v_2'(\phi \wedge \iota(p_1)) \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$

$$\Longrightarrow \qquad \text{(by rule } \textbf{alpha})$$

$$p_1 \xrightarrow{a,\phi} q' \quad \wedge \quad q \simeq_\alpha q' \quad \wedge \quad \neg \mathsf{cv}(q)$$
$$\wedge \quad \models v_2'(\phi \wedge \iota(p_1)) \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$

$$\Longrightarrow \qquad \text{(by Table 5.3)}$$

$$p_1 + p_2 \xrightarrow{a,\phi\wedge\iota(p_1)} q' \quad \wedge \quad q \simeq_\alpha q' \quad \wedge \quad \neg \mathsf{cv}(q)$$
$$\wedge \quad \models v_2'(\phi \wedge \iota(p_1)) \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$

$$\Longrightarrow \qquad \text{(by rule } \textbf{alpha})$$

$$p_1 + p_2 \xrightarrow{a,\phi\wedge\iota(p_1)}{}' q$$
$$\wedge \quad \models v_2'(\phi \wedge \iota(p_1)) \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$

$$\Longrightarrow \qquad \left(\begin{array}{c}\text{since } \iota(p_1 + p_2) = \iota(p_1) \vee \iota(p_2) \text{ and } v_2' \restriction fv(q) = v_2'' \restriction fv(q) \text{ because} \\ \neg \mathsf{cv}(p_1 + p_2) \text{ and Lemma B.6, with } v_2'' = v_2[\kappa(p_1 + p_2) \leftarrowtail 0] + d\end{array}\right)$$

$$p_1 + p_2 \xrightarrow{a,\phi\wedge\iota(p_1)}{}' q$$
$$\wedge \quad \models v_2''(\phi \wedge \iota(p_1) \wedge \iota(p_1 + p_2)) \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2'')\rangle \in (\sim_t \cup R)^*$$

$$\Longrightarrow \qquad \text{(by Def. 4.6)}$$

$$(p_1 + p_2, v_2) \xrightarrow{a(d)} (q, v_2'') \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2'')\rangle \in (\sim_t \cup R)^*$$

Subcase $\psi \gg p$.

$$\langle \psi \gg p, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle$$

$$\Longrightarrow \qquad \text{(by Def. 5.23)}$$

$$\langle p, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle \quad\wedge\quad \models (v_1 + d)(\psi)$$
$\Longrightarrow$ (by induction)
$$(p, v_2) \xrightarrow{a(d)} (q, v_2') \quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \quad\wedge\quad \models (v_1 + d)(\psi)$$
$\Longrightarrow$ (by Def. 4.6, $v_2' = v_2[\kappa(p) \mapsto 0] + d$)
$$p \xrightarrow{a,\phi}{}' q \quad\wedge\quad \models v_2'(\phi \wedge \iota(p))$$
$$\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \quad\wedge\quad \models (v_1 + d)(\psi)$$
$\Longrightarrow$ (by rule **alpha**)
$$p \xrightarrow{a,\phi} q' \quad\wedge\quad q \simeq_\alpha q' \quad\wedge\quad \neg\mathsf{cv}(q) \quad\wedge\quad \models v_2'(\phi \wedge \iota(p))$$
$$\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \quad\wedge\quad \models (v_1 + d)(\psi)$$
$\Longrightarrow$ (by Table 5.3 and $v_1 \restriction fv(p) = v_2 \restriction fv(p)$)
$$\psi \gg p \xrightarrow{a,\phi} q' \quad\wedge\quad q \simeq_\alpha q' \quad\wedge\quad \neg\mathsf{cv}(q) \quad\wedge\quad \models v_2'(\phi \wedge \iota(p))$$
$$\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \quad\wedge\quad \models (v_2 + d)(\psi)$$
$\Longrightarrow$ (by rule **alpha**, $\neg\mathsf{cv}(p)$, and calculations)
$$\psi \gg p \xrightarrow{a,\phi}{}' q \quad\wedge\quad \models v_2'(\phi \wedge \iota(\psi \gg p)) \quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$
$\Longrightarrow$ (by Def. 4.6 and $v_2' = v_2[\kappa(\psi \gg p) \mapsto 0] + d$)
$$(\psi \gg p, v_2) \xrightarrow{a(d)} (q, v_2') \quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$

Subcase $\{\!|C|\!\}\, p$.

$$\langle \{\!|C|\!\}\, p, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle$$
$\Longrightarrow$ (by Def. 5.23)
$$\langle p, v_1[C \mapsto 0]\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle$$
$\Longrightarrow$ (by induction)
$$(p, v_2[C \mapsto 0]) \xrightarrow{a(d)} (q, v_2') \quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$
$\Longrightarrow$ (by Def. 4.6, $v_2' = v_2[C \mapsto 0][\kappa(p) \mapsto 0] + d$)
$$p \xrightarrow{a,\phi}{}' q \quad\wedge\quad \models v_2'(\phi \wedge \iota(p)) \quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$
$\Longrightarrow$ (by rule **alpha**)
$$p \xrightarrow{a,\phi} q' \quad\wedge\quad q \simeq_\alpha q' \quad\wedge\quad \neg\mathsf{cv}(q)$$
$$\wedge\quad \models v_2'(\phi \wedge \iota(p)) \quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$
$\Longrightarrow$ (by Table 5.3)
$$\{\!|C|\!\}\, p \xrightarrow{a,\phi} q' \quad\wedge\quad q \simeq_\alpha q' \quad\wedge\quad \neg\mathsf{cv}(q)$$
$$\wedge\quad \models v_2'(\phi \wedge \iota(\{\!|C|\!\}\, p)) \quad\wedge\quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*$$
$\Longrightarrow$ (by rule **alpha**)

$$
\begin{array}{ll}
& \{\!|C|\!\}\, p \xrightarrow{a,\phi}{}' q \\
\wedge & \models v_2'(\phi \wedge \iota(\{\!|C|\!\}\, p)) \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \\
\Longrightarrow & \qquad\qquad \text{(by Def. 4.6 and } v_2' = v_2[\kappa(\{\!|C|\!\}\, p) \mapsto 0] + d) \\
& (\{\!|C|\!\}\, p, v_2) \xrightarrow{a(d)} (q, v_2') \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*
\end{array}
$$

Subcase X.

$$
\begin{array}{ll}
& \langle X, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle \\
\Longrightarrow & \qquad\qquad \text{(by Def. 5.23)} \\
& \langle p, v_1\rangle \xrightarrow{a(d)} \langle p', v_1'\rangle \\
\Longrightarrow & \qquad\qquad \text{(by induction)} \\
& (p, v_2) \xrightarrow{a(d)} (q, v_2') \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \\
\Longrightarrow & \qquad\qquad \text{(by Def. 4.6, } v_2' = v_2[\kappa(p) \mapsto 0] + d) \\
& p \xrightarrow{a,\phi}{}' q \quad \wedge \quad \models v_2'(\phi \wedge \iota(p)) \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \\
\Longrightarrow & \qquad\qquad \text{(by rule } \textbf{alpha}) \\
& p \xrightarrow{a,\phi} q' \quad \wedge \quad q \simeq_\alpha q' \quad \wedge \quad \neg \mathsf{cv}(q) \\
\wedge & \models v_2'(\phi \wedge \iota(p)) \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \\
\Longrightarrow & \qquad\qquad \text{(by Table 5.3)} \\
& X \xrightarrow{a,\phi} q' \quad \wedge \quad q \simeq_\alpha q' \quad \wedge \quad \neg \mathsf{cv}(q) \\
\wedge & \models v_2'(\phi \wedge \iota(X)) \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \\
\Longrightarrow & \qquad\qquad \text{(by rule } \textbf{alpha}) \\
& X \xrightarrow{a,\phi}{}' q \quad \wedge \quad \models v_2'(\phi \wedge \iota(X)) \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^* \\
\Longrightarrow & \qquad\qquad \text{(by Def. 4.6 and } v_2' = v_2[\kappa(X) \mapsto 0] + d) \\
& (X, v_2) \xrightarrow{a(d)} (q, v_2') \quad \wedge \quad \langle\langle p', v_1'\rangle, (q, v_2')\rangle \in (\sim_t \cup R)^*
\end{array}
$$

Case 2.

Subcases $\mathbf{0}$ *and* $a; p$. Straightforward.

Subcase $\phi \mapsto p$.

$$
\begin{array}{ll}
& \mathcal{U}_d\langle \phi \mapsto p, v_1\rangle \\
\Longleftrightarrow & \qquad\qquad \text{(by Def. 5.23)} \\
& \mathcal{U}_d\langle p, v_1\rangle \\
\Longleftrightarrow & \qquad\qquad \text{(by induction)} \\
& \mathcal{U}_d(p, v_2) \\
\Longleftrightarrow & \qquad\qquad \text{(by Def. 4.6)} \\
& \models (v_2[\kappa(p) \mapsto 0] + d)(\iota(p))
\end{array}
$$

$\iff$ (by Table 5.3)

$\models (v_2[\kappa(\phi \mapsto p) \leftarrowtail 0] + d)(\iota(\phi \mapsto p))$

$\iff$ (by Def. 4.6)

$\mathcal{U}_d(\phi \mapsto p, v_2)$

Subcase $p_1 + p_2$.

$\mathcal{U}_d\langle p_1 + p_2, v_1\rangle$

$\iff$ (by Def. 5.23)

$\mathcal{U}_d\langle p_1, v_1\rangle \quad \vee \quad \mathcal{U}_d\langle p_2, v_1\rangle$

$\iff$ (by induction)

$\mathcal{U}_d(p_1, v_2) \quad \vee \quad \mathcal{U}_d(p_2, v_2)$

$\iff$ (by Def. 4.6)

$\models (v_2[\kappa(p_1) \leftarrowtail 0] + d)(\iota(p_1)) \quad \vee \quad \models (v_2[\kappa(p_2) \leftarrowtail 0] + d)(\iota(p_2))$

$\iff$ (since $\neg\mathsf{cv}(p_1 + p_2)$ and logic)

$\models (v_2[\kappa(p_1) \cup \kappa(p_2) \leftarrowtail 0] + d)(\iota(p_1) \vee \iota(p_2))$

$\iff$ (by Table 5.3)

$\models (v_2[\kappa(p_1 + p_2) \leftarrowtail 0] + d)(\iota(p_1 + p_2))$

$\iff$ (by Def. 4.6)

$\mathcal{U}_d(\phi \mapsto p, v_2)$

Subcase $\psi \gg p$.

$\mathcal{U}_d\langle \psi \gg p, v_1\rangle$

$\iff$ (by Def. 5.23)

$\mathcal{U}_d\langle p, v_1\rangle \quad \wedge \quad \models (v_1 + d)(\psi)$

$\iff$ (by induction)

$\mathcal{U}_d(p, v_2) \quad \wedge \quad \models (v_1 + d)(\psi)$

$\iff$ (by Def. 4.6)

$\models (v_2[\kappa(p) \leftarrowtail 0] + d)(\iota(p)) \quad \wedge \quad \models (v_1 + d)(\psi)$

$\iff$ (since $v_1 \restriction fv(\psi \gg p) = v_2 \restriction fv(\psi \gg p)$ and $\neg\mathsf{cv}(\psi \gg p)$)

$\models (v_2[\kappa(p) \leftarrowtail 0] + d)(\iota(p)) \quad \wedge \quad \models (v_2[\kappa(p) \leftarrowtail 0] + d)(\psi)$

$\iff$ (by Table 5.3)

$\models (v_2[\kappa(\psi \gg p) \leftarrowtail 0] + d)(\iota(\psi \gg p))$

$\iff$ (by Def. 4.6)

$\mathcal{U}_d(\psi \gg p, v_2)$

Subcase $\{\!|C|\!\}\, p$.

$$
\begin{array}{ll}
& \mathcal{U}_d\langle \{\!|C|\!\}\, p, v_1\rangle \\
\iff & \text{(by Def. 5.23)} \\
& \mathcal{U}_d\langle p, v_1[C \leftarrowtail 0]\rangle \\
\iff & \text{(by induction)} \\
& \mathcal{U}_d(p, v_2[C \leftarrowtail 0]) \\
\iff & \text{(by Def. 4.6)} \\
& \models (v_2[C \leftarrowtail 0][\kappa(p) \leftarrowtail 0] + d)(\iota(p)) \\
\iff & \text{(by Table 5.3)} \\
& \models (v_2[\kappa(\{\!|C|\!\}\, p) \leftarrowtail 0] + d)(\iota(\{\!|C|\!\}\, p)) \\
\iff & \text{(by Def. 4.6)} \\
& \mathcal{U}_d(\{\!|C|\!\}\, p, v_2)
\end{array}
$$

Subcase X.

$$
\begin{array}{ll}
& \mathcal{U}_d\langle X, v_1\rangle \\
\iff & \text{(by Def. 5.23)} \\
& \mathcal{U}_d\langle p, v_1\rangle \\
\iff & \text{(by induction)} \\
& \mathcal{U}_d(p, v_2) \\
\iff & \text{(by Def. 4.6)} \\
& \models (v_2[\kappa(p) \leftarrowtail 0] + d)(\iota(p)) \\
\iff & \text{(by Table 5.3)} \\
& \models (v_2[\kappa(X) \leftarrowtail 0] + d)(\iota(X)) \\
\iff & \text{(by Def. 4.6)} \\
& \mathcal{U}_d(X, v_2)
\end{array}
$$

Q.E.D.

B.4 Proof of Theorems 5.25 and 5.22

Theorem 5.25. *Let $p, q \in \heartsuit^{br}$, such that $p \sim_t q$. Let $\mathbf{C}[\ ]$ be a context built using operators $\mathbf{0}$, $a;\_$, $\phi \mapsto \_$, $\psi \gg \_$, $\{\!|C|\!\}\, \_$, $+$, and process variables. Then, it holds that $\mathbf{C}[p] \sim_t \mathbf{C}[q]$. Moreover, $\sim_t$ is also a congruence for recursion in $\heartsuit^{br}$ (see Theorem 2.13 for an appropriate formulation).*

Proof. Given any valuation $v \in \mathbf{V}$, define

$$
\begin{aligned}
R_v^{\mathrm{p}} &\stackrel{\text{def}}{=} \{\langle\langle a; p, v\rangle, \langle a; q, v\rangle\rangle \mid p \sim_t q\},\\
R_v^{\mathrm{g}} &\stackrel{\text{def}}{=} \{\langle\langle \phi \mapsto p, v\rangle, \langle \phi \mapsto q, v\rangle\rangle \mid p \sim_t q\},\\
R_v^{\mathrm{s}} &\stackrel{\text{def}}{=} \{\langle\langle p + p', v\rangle, \langle q + q', v\rangle\rangle \mid p \sim_t q \wedge p' \sim_t q'\},\\
\\
R_v^{\mathrm{cs}} &\stackrel{\text{def}}{=} \{\langle\langle \{\!|C|\!\}\, p, v\rangle, \langle \{\!|C|\!\}\, q, v\rangle\rangle \mid p \sim_t q\}, \text{ and}\\
R_v^{\mathrm{i}} &\stackrel{\text{def}}{=} \{\langle\langle \psi \gg p, v\rangle, \langle \psi \gg q, v\rangle\rangle \mid p \sim_t q\}.
\end{aligned}
$$

The proof that these relations are timed bisimulations up to $\sim_t$ is standard.

For the case of recursion we only prove the case with only one process variable. The case with several variables follows similarly. Let p and q be two terms in $\heartsuit^{br}$ containing at most variable $X \in \mathcal{V}$ such that for all $r \in \heartsuit^{br}$, $p[r/X] \sim_t q[r/X]$, that is, for all valuations $v \in \mathbf{V}$, $\langle p[r/X], v\rangle \sim_t \langle q[r/X], v\rangle$. Let Y and Z defined by

$$Y = p[Y/X] \qquad\qquad Z = q[Z/X]$$

It is not hard to prove that for any $v \in \mathbf{V}$

$$R_v^{\mathrm{rec}} \stackrel{\text{def}}{=} \{\langle\langle r[Y/X], v\rangle, \langle r[Z/X], v\rangle\rangle \mid r \in \heartsuit^{br} \text{ containing at most the variable } X\}^{\mathsf{s}}$$

is a timed bisimulations up to $\sim_t$. As a consequence $Y \sim_t Z$ by taking the term r to be the variable X.

The relations given above suffice to prove the theorem. *Q.E.D.*

Theorem 5.22. *Let p and q be two α-conflict-free $\heartsuit$ terms such that $p \sim_t q$. Let $\mathbf{C}[\ ]$ be a $\heartsuit$ context such that $\mathbf{C}[p]$ and $\mathbf{C}[q]$ are α-conflict-free. Then, it holds that $\mathbf{C}[p] \sim_t \mathbf{C}[q]$.*

Proof. For the basic operation we proceed as for the previous theorem: given any valuation $v \in \mathbf{V}$, define

$$
\begin{aligned}
R_v^{\mathrm{p}} &\stackrel{\text{def}}{=} \{\langle (a; p, v), (a; q, v)\rangle \mid p \sim_t q \wedge \neg \mathsf{cv}(a; p) \wedge \neg \mathsf{cv}(a; q)\},\\
R_v^{\mathrm{g}} &\stackrel{\text{def}}{=} \{\langle (\phi \mapsto p, v), (\phi \mapsto q, v)\rangle \mid p \sim_t q \wedge \neg \mathsf{cv}(\phi \mapsto p) \wedge \neg \mathsf{cv}(\phi \mapsto q)\},\\
R_v^{\mathrm{s}} &\stackrel{\text{def}}{=} \{\langle (p + p', v), (q + q', v)\rangle \mid p \sim_t q \wedge p' \sim_t q' \wedge \neg \mathsf{cv}(p + p') \wedge \neg \mathsf{cv}(q + p')\},\\
R_v^{\mathrm{cs}} &\stackrel{\text{def}}{=} \{\langle (\{\!|C|\!\}\, p, v), (\{\!|C|\!\}\, q, v)\rangle \mid p \sim_t q \wedge \neg \mathsf{cv}(\{\!|C|\!\}\, p) \wedge \neg \mathsf{cv}(\{\!|C|\!\}\, q)\}, \text{ and}\\
R_v^{\mathrm{i}} &\stackrel{\text{def}}{=} \{\langle (\psi \gg p, v), (\psi \gg q, v)\rangle \mid p \sim_t q \wedge \neg \mathsf{cv}(\psi \gg p) \wedge \neg \mathsf{cv}(\psi \gg q)\}.
\end{aligned}
$$

The proof that these relations are timed bisimulations up to $\sim_t$ is not difficult. However, the proof in this case is more involved than the one for Theorem 5.25. So, we prove as an example that relation R_v^{cs} is a timed bisimulation up to $\sim_t$. We check the transfer properties according to Definition 3.5.

Case 1.

$$
\begin{array}{ll}
(\{\!|C|\!\}\, p, v) \xrightarrow{a(d)} (p', v_1) & \\
\Longrightarrow & \text{(by Def. 4.6 and } v_1 = v[\kappa(\{\!|C|\!\}\, p) \leftarrowtail 0] + d) \\
\quad \{\!|C|\!\}\, p \xrightarrow{a,\phi}{}' p' \quad \wedge \quad \models v_1(\phi \wedge \iota(\{\!|C|\!\}\, p)) & \\
\Longrightarrow & \text{(by rule } \textbf{alpha}) \\
\quad \{\!|C|\!\}\, p \xrightarrow{a,\phi} p'' \quad \wedge \quad p' \simeq_\alpha p'' \quad \wedge \quad \neg \mathsf{cv}(p') \quad \wedge \quad \models v_1(\phi \wedge \iota(\{\!|C|\!\}\, p)) & \\
\Longrightarrow & \text{(by Table 5.3)} \\
\quad p \xrightarrow{a,\phi} p'' \quad \wedge \quad p' \simeq_\alpha p'' \quad \wedge \quad \neg \mathsf{cv}(p') \wedge \quad \models v_1(\phi \wedge \iota(p)) & \\
\Longrightarrow & \text{(by rule } \textbf{alpha}) \\
\quad p \xrightarrow{a,\phi}{}' p' \quad \wedge \quad \models v_1(\phi \wedge \iota(p)) & \\
\Longrightarrow & \text{(by Def. 4.6, } v_1 = v[C \leftarrowtail 0][\kappa(p) \leftarrowtail 0] + d) \\
\quad (p, v[C \leftarrowtail 0]) \xrightarrow{a(d)} (p', v_1) & \\
\Longrightarrow & \text{(since } p \sim_t q) \\
\quad (q, v[C \leftarrowtail 0]) \xrightarrow{a(d)} (q', v_2) \quad \wedge \quad \langle (p', v_1'), (q, v_2') \rangle \in \sim_t & \\
\Longrightarrow & \text{(by Def. 4.6, } v_2 = v[C \leftarrowtail 0][\kappa(q) \leftarrowtail 0] + d, \text{ and } \sim_t \subseteq (\sim_t \cup R_v^{\mathrm{cs}})^*) \\
\quad q \xrightarrow{a,\phi'}{}' q' \quad \wedge \quad \models v_2(\phi' \wedge \iota(q)) \quad \wedge \quad \langle (p', v_1'), (q, v_2') \rangle \in (\sim_t \cup R_v^{\mathrm{cs}})^* & \\
\Longrightarrow & \text{(by rule } \textbf{alpha}) \\
\quad q \xrightarrow{a,\phi'} q'' \quad \wedge \quad q' \simeq_\alpha q'' \quad \wedge \quad \neg \mathsf{cv}(q') & \\
\quad \wedge \quad \models v_2(\phi' \wedge \iota(q)) \quad \wedge \quad \langle (p', v_1'), (q, v_2') \rangle \in (\sim_t \cup R_v^{\mathrm{cs}})^* & \\
\Longrightarrow & \text{(by Table 5.3)} \\
\quad \{\!|C|\!\}\, q \xrightarrow{a,\phi'} q'' \quad \wedge \quad q' \simeq_\alpha q'' \quad \wedge \quad \neg \mathsf{cv}(q') & \\
\quad \wedge \quad \models v_2(\phi' \wedge \iota(\{\!|C|\!\}\, q)) \quad \wedge \quad \langle (p', v_1'), (q, v_2') \rangle \in (\sim_t \cup R_v^{\mathrm{cs}})^* & \\
\Longrightarrow & \text{(by rule } \textbf{alpha}) \\
\quad \{\!|C|\!\}\, q \xrightarrow{a,\phi'}{}' q' \quad \wedge \quad \models v_2(\phi' \wedge \iota(\{\!|C|\!\}\, q)) \quad \wedge \quad \langle (p', v_1'), (q, v_2') \rangle \in (\sim_t \cup R_v^{\mathrm{cs}})^* & \\
\Longrightarrow & \text{(by Def. 4.6 and } v_2 = v[\kappa(\{\!|C|\!\}\, q) \leftarrowtail 0] + d) \\
\quad (\{\!|C|\!\}\, q, v) \xrightarrow{a(d)} (q', v_2) \quad \wedge \quad \langle (p', v_1'), (q, v_2') \rangle \in (\sim_t \cup R_v^{\mathrm{cs}})^* &
\end{array}
$$

Case 2.

$$
\begin{array}{ll}
\mathcal{U}_d(\{\!|C|\!\}\, p, v) & \\
\Longrightarrow & \text{(by Def. 4.6)} \\
\quad \models (v[\kappa(\{\!|C|\!\}\, p) \leftarrowtail 0] + d)(\iota(\{\!|C|\!\}\, p)) & \\
\Longrightarrow & \text{(by Table 5.3)} \\
\quad \models (v[C \leftarrowtail 0][\kappa(p) \leftarrowtail 0] + d)(\iota(p)) &
\end{array}
$$

$$
\begin{array}{ll}
\Longrightarrow & \text{(by Def. 4.6)} \\
\quad \mathcal{U}_d(p, v[C \leftarrowtail 0]) & \\
\Longrightarrow & \text{(since } p \sim_t q\text{)} \\
\quad \mathcal{U}_d(q, v[C \leftarrowtail 0]) & \\
\Longrightarrow & \text{(by Def. 4.6)} \\
\quad \models (v[C \leftarrowtail 0][\kappa(q) \leftarrowtail 0] + d)(\iota(q)) & \\
\Longrightarrow & \text{(by Table 5.3)} \\
\quad \models (v[\kappa(\{\!|C|\!\}\, q) \leftarrowtail 0] + d)(\iota(\{\!|C|\!\}\, q)) & \\
\Longrightarrow & \text{(by Def. 4.6)} \\
\quad \mathcal{U}_d(\{\!|C|\!\}\, q, v) &
\end{array}
$$

Let R be a timed bisimulations. For the renaming operator, we define

$$R^{\mathrm{rn}} \stackrel{\text{def}}{=} \{\langle (p[f], v_1), (q[f], v_2)\rangle \mid \langle (p, v_1), (q, v_2)\rangle \in R\}.$$

It is not difficult to prove that R^{rn} is a timed bisimulation.

For the different parallel compositions we define

$$
\begin{array}{rll}
R^{\mathrm{m}} \stackrel{\text{def}}{=} & \{\langle (p_1 \,||_A\, q, v_1), (p_2 \,||_A\, q, v_2)\rangle \mid \langle (p_1, \hat{v}_1), (p_2, \hat{v}_2)\rangle \in \sim_t & \\
& \qquad \wedge \quad v_1 \restriction fv(p_1) = \hat{v}_1 \restriction fv(p_1) & \\
& \qquad \wedge \quad v_2 \restriction fv(p_2) = \hat{v}_2 \restriction fv(p_2) & \\
& \qquad \wedge \quad v_1 \restriction fv(q) = v_2 \restriction fv(q) & \\
& \qquad \wedge \quad \neg \mathsf{cv}(p_1 \,||_A\, q) \ \wedge\ \neg \mathsf{cv}(p_2 \,||_A\, q) \quad \} & \\
\cup & \{\langle (\overline{\mathsf{ck}}(p_1) \,||_A\, q, v_1), (\overline{\mathsf{ck}}(p_2) \,||_A\, q, v_2)\rangle \mid \langle (p_1, \hat{v}_1), (p_2, \hat{v}_2)\rangle \in \sim_t & \\
& \qquad \wedge \quad v_1 \restriction fv(p_1) = (\hat{v}_1[\kappa(p_1) \leftarrowtail 0] + d) \restriction fv(p_1) & \\
& \qquad \wedge \quad v_2 \restriction fv(p_2) = (\hat{v}_2[\kappa(p_2) \leftarrowtail 0] + d) \restriction fv(p_2) & \\
& \qquad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad v_1 \restriction fv(q) = v_2 \restriction fv(q) & \\
& \qquad \wedge \quad \neg \mathsf{cv}(\overline{\mathsf{ck}}(p_1) \,||_A\, q) \ \wedge\ \neg \mathsf{cv}(\overline{\mathsf{ck}}(p_2) \,||_A\, q) \quad \} &
\end{array}
$$

$$
\begin{array}{rll}
R^{\mathrm{lm}} \stackrel{\text{def}}{=} & \{\langle (p_1 \,\llbracket_A\, q, v_1), (p_2 \,\llbracket_A\, q, v_2)\rangle \mid \langle (p_1, \hat{v}_1), (p_2, \hat{v}_2)\rangle \in \sim_t & \\
& \qquad \wedge \quad v_1 \restriction fv(p_1) = \hat{v}_1 \restriction fv(p_1) & \\
& \qquad \wedge \quad v_2 \restriction fv(p_2) = \hat{v}_2 \restriction fv(p_2) & \\
& \qquad \wedge \quad \hat{v}_1 \restriction fv(q) = \hat{v}_2 \restriction fv(q) & \\
& \qquad \wedge \quad \neg \mathsf{cv}(p_1 \,\llbracket_A\, q) \ \wedge\ \neg \mathsf{cv}(p_2 \,\llbracket_A\, q) \quad \} \ \cup \ R^{\mathrm{m}} &
\end{array}
$$

$$
\begin{array}{rcl}
R^{\mathrm{cm}} & \stackrel{\mathrm{def}}{=} & \{\langle (p_1 \mid_A q, v_1), (p_2 \mid_A q, v_2)\rangle \mid \langle (p_1, \hat{v}_1), (p_2, \hat{v}_2)\rangle \in \sim_t \\
& & \quad \wedge \quad v_1 \restriction fv(p_1) = \hat{v}_1 \restriction fv(p_1) \\
& & \quad \wedge \quad v_2 \restriction fv(p_2) = \hat{v}_2 \restriction fv(p_2) \\
& & \quad \wedge \quad \hat{v}_1 \restriction fv(q) = \hat{v}_2 \restriction fv(q) \\
& & \quad \wedge \quad \neg \mathsf{cv}(p_1 \mid_A q) \ \wedge \ \neg \mathsf{cv}(p_2 \mid_A q) \quad \} \ \cup \ R^{\mathrm{m}}.
\end{array}
$$

The relations for the commutative situation are defined in the same way. We claim that all this relations are timed bisimulations up to $\sim_t$. We only prove that R^{m} is a timed bisimulation up to $\sim_t$. Proofs for the other relations follows in the same way. The following claim will be used in many occasions along the proof.

Claim B.7.a. *$p \xrightarrow{a,\phi} p'$ implies $fv(p') \subseteq fv(p) \cup \kappa(p)$.*

The proof follows straightforwardly by induction on the depth of the proof tree. We proceed now to check the transfer properties for each case in R^{m}.

Case $\langle (p_1 \mid\mid_A q, v_1), (p_2 \mid\mid_A q, v_2)\rangle \in R^{\mathrm{m}}$.

Subcase 1.

$$
\begin{array}{ll}
& (p_1 \mid\mid_A q, v_1) \xrightarrow{a(d)} (r, v_1') \\
\Longrightarrow & \qquad \text{(by Def. 4.6 and } v_1' = v_1[\kappa(p_1 \mid\mid_A q) \leftarrowtail 0] + d) \\
& p_1 \mid\mid_A q \xrightarrow{a,\phi_1}{}' r \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(p_1 \mid\mid_A q)) \\
\Longrightarrow & \qquad \text{(by rule } \textbf{alpha}) \\
& p_1 \mid\mid_A q \xrightarrow{a,\phi_1} r' \quad \wedge \quad r \simeq_\alpha r' \quad \wedge \quad \neg \mathsf{cv}(r) \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(p_1 \mid\mid_A q)) \\
\Longrightarrow & \qquad \text{(by Table 5.3)} \\
& (\ (p_1 \xrightarrow{a,\phi_1} p_1' \ \wedge \ r' \equiv p_1' \mid\mid_A \overline{\mathsf{ck}}(q)) \qquad (*) \\
& \ \vee \ (q \xrightarrow{a,\phi_1} q' \ \wedge \ r' \equiv \overline{\mathsf{ck}}(p_1) \mid\mid_A q') \qquad (**) \\
& \ \vee \ (p_1 \xrightarrow{a,\phi_1'} p_1' \ \wedge \ q \xrightarrow{a,\phi_1''} q' \ \wedge \ r' \equiv p_1' \mid\mid_A q' \ \wedge \ \phi_1 \equiv \phi_1' \wedge \phi_1'') \ ) \qquad (***) \\
& \wedge \ r \simeq_\alpha r' \quad \wedge \quad \neg \mathsf{cv}(r) \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(p_1) \wedge \iota(q))
\end{array}
$$

We proceed with each subcase separately.

Sub-subcase (*)

$$
\begin{array}{ll}
& p_1 \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad r \simeq_\alpha p_1' \mid\mid_A \overline{\mathsf{ck}}(q) \quad \wedge \quad \neg \mathsf{cv}(r) \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(p_1) \wedge \iota(q)) \\
\Longrightarrow & \qquad \text{(by logic, Def. 5.9 } (\simeq_\alpha) \text{ with } r \equiv p_1'' \mid\mid_A q' \text{ , and Def. 5.4 } (\mathsf{cv})) \\
& p_1 \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(p_1)) \quad \wedge \quad p_1'' \simeq_\alpha p_1' \quad \wedge \quad \neg \mathsf{cv}(p_1'') \quad \wedge \quad \models v_1'(\iota(q)) \\
\Longrightarrow & \qquad \text{(by rule } \textbf{alpha}) \\
& p_1 \xrightarrow{a,\phi_1}{}' p_1'' \quad \wedge \quad \models v_1'(\phi \wedge \iota(p_1)) \quad \wedge \quad \models v_1'(\iota(q))
\end{array}
$$

$$\Longrightarrow \quad \left(\begin{array}{r} \text{since } v_1 \restriction fv(p_1) = \hat{v}_1 \restriction fv(p_1),\ \hat{v}_1' = \hat{v}_1[\kappa(p_1) \hookleftarrow 0] + d, \\ v_1 \restriction fv(q) = v_2 \restriction fv(q),\ v_2' = v_2[\kappa(p_2 \mid\mid_A q) \hookleftarrow 0] + d, \text{ and } \neg\mathsf{cv}(p_2 \mid\mid_A q) \end{array}\right)$$

$$p_1 \xrightarrow{a,\phi_1}{}' p_1'' \quad \wedge \quad \models \hat{v}_1'(\phi_1 \wedge \iota(p_1)) \quad \wedge \quad \models v_2'(\iota(q))$$

$$\Longrightarrow \qquad \text{(by Def. 4.6)}$$

$$(p_1, \hat{v}_1) \xrightarrow{a(d)} (p_1'', \hat{v}_1') \quad \wedge \quad \models v_2'(\iota(q))$$

$$\Longrightarrow \qquad \text{(since } \langle (p_1, \hat{v}_1), (p_2, \hat{v}_2)\rangle \in \sim_t\text{)}$$

$$(p_2, \hat{v}_2) \xrightarrow{a(d)} (p_2''', \hat{v}_2') \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$$

$$\Longrightarrow \qquad \text{(by Def. 4.6, } \hat{v}_2' = \hat{v}_2[\kappa(p_2) \hookleftarrow 0] + d\text{)}$$

$$p_2 \xrightarrow{a,\phi_2}{}' p_2''' \quad \wedge \quad \models \hat{v}_2'(\phi_2 \wedge \iota(p_2))$$
$$\wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$$

$$\Longrightarrow \qquad \text{(by rule } \textbf{alpha}\text{)}$$

$$p_2 \xrightarrow{a,\phi_2} p_2' \quad \wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models \hat{v}_2'(\phi_2 \wedge \iota(p_2))$$
$$\wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$$

$$\Longrightarrow \qquad \text{(by Table 5.3)}$$

$$p_2 \mid\mid_A q \xrightarrow{a,\phi_2} p_2' \mid\mid_A \overline{\mathsf{ck}}(q) \quad \wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models \hat{v}_2'(\phi_2 \wedge \iota(p_2))$$
$$\wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$$

$$\Longrightarrow \qquad \text{(by rule } \textbf{alpha}\text{)}$$

$$p_2 \mid\mid_A q \xrightarrow{a,\phi_2}{}' p_2'' \mid\mid_A q'' \quad \wedge \quad p_2'' \mid\mid_A q'' \simeq_\alpha p_2' \mid\mid_A \overline{\mathsf{ck}}(q) \quad \wedge \quad \neg\mathsf{cv}(p_2'' \mid\mid_A q'') \quad \wedge$$
$$p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models \hat{v}_2'(\phi_2 \wedge \iota(p_2)) \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$$

$$\Longrightarrow \qquad \text{(since } v_2 \restriction fv(p_2) = \hat{v}_2 \restriction fv(p_2), \neg\mathsf{cv}(p_2 \mid\mid_A q), \text{ logic, and Table 5.3)}$$

$$p_2 \mid\mid_A q \xrightarrow{a,\phi_2}{}' p_2'' \mid\mid_A q'' \quad \wedge \quad p_2'' \mid\mid_A q'' \simeq_\alpha p_2' \mid\mid_A \overline{\mathsf{ck}}(q) \quad \wedge \quad \neg\mathsf{cv}(p_2'' \mid\mid_A q'')$$
$$\wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models v_2'(\phi_2 \wedge \iota(p_2 \mid\mid_A q)) \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t$$

$$\Longrightarrow \qquad \text{(by Def. 4.6, Def. 5.9 } (\simeq_\alpha)\text{)}$$

$$(p_2 \mid\mid_A q, v_2) \xrightarrow{a(d)} (p_2'' \mid\mid_A q'', v_2') \quad \wedge \quad p_2'' \simeq_\alpha p_2''' \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t$$

$$\Longrightarrow \qquad \text{(by Theorems 5.16 and 4.20)}$$

$$(p_2 \mid\mid_A q, v_2) \xrightarrow{a(d)} (p_2'' \mid\mid_A q'', v_2') \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2'', \hat{v}_2')\rangle \in \sim_t \qquad (\dagger)$$

Since $p_1'' \mid\mid_A q' \simeq_\alpha p_1' \mid\mid_A \overline{\mathsf{ck}}(q)$ and $p_2'' \mid\mid_A q'' \simeq_\alpha p_2' \mid\mid_A \overline{\mathsf{ck}}(q)$, $q' \simeq_\alpha \overline{\mathsf{ck}}(q) \simeq_\alpha q''$. Thus, $fv(q') = fv(\overline{\mathsf{ck}}(q)) = fv(q'')$ and $\kappa(q') = \kappa(q'') = \kappa(\overline{\mathsf{ck}}(q)) = \varnothing$. As a consequence $p_1'' \mid\mid_A q' \simeq_\alpha p_1'' \mid\mid_A \overline{\mathsf{ck}}(q)$, $p_2'' \mid\mid_A q'' \simeq_\alpha p_2'' \mid\mid_A \overline{\mathsf{ck}}(q)$, $\neg\mathsf{cv}(p_1'' \mid\mid_A \overline{\mathsf{ck}}(q))$, and $\neg\mathsf{cv}(p_2'' \mid\mid_A \overline{\mathsf{ck}}(q))$. Together with the fact that $fv(p_1'') \subseteq fv(p_1) \cup \kappa(p_1)$, $fv(p_2'') \subseteq fv(p_2) \cup \kappa(p_2)$, and $fv(\overline{\mathsf{ck}}(q)) \subseteq fv(q) \cup \kappa(q)$, we have,

$$(\dagger) \implies$$

$$(p_2 \mid\mid_A q, v_2) \xrightarrow{a(d)} (p_2'' \mid\mid_A q'', v_2') \quad \wedge \quad \langle (p_1'' \mid\mid_A \overline{\mathsf{ck}}(q), v_1'), (p_2'' \mid\mid_A \overline{\mathsf{ck}}(q), v_2')\rangle \in R^{\mathrm{m}}$$

$$\Longrightarrow \qquad \text{(by Theorems 5.16 and 4.20)}$$

$$(p_2 \,||_A\, q, v_2) \xrightarrow{a(d)} (p_2'' \,||_A\, q'', v_2') \quad \wedge \quad \langle (p_1'' \,||_A\, q', v_1'), (p_2'' \,||_A\, q'', v_2') \rangle \in (\sim_t \cup R^{\mathrm{m}})^*$$

Sub-subcase (**)

$$q \xrightarrow{a,\phi_1} q' \quad \wedge \quad r \simeq_\alpha \overline{\mathsf{ck}}(p_1) \,||_A\, q' \quad \wedge \quad \neg\mathsf{cv}(r) \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(p_1) \wedge \iota(q))$$
$$\Longrightarrow \qquad \text{(by logic)}$$
$$q \xrightarrow{a,\phi_1} q' \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(q)) \quad \wedge \quad \models v_1'(\iota(p_1))$$
$$\Longrightarrow \qquad \left(\begin{array}{c} \text{since } v_1 \restriction fv(p_1) = \hat{v}_1 \restriction fv(p_1),\ \hat{v}_1' = \hat{v}_1[\kappa(p_1) \leftarrow\!\shortmid 0] + d, \\ v_1 \restriction fv(q) = v_2 \restriction fv(q), \text{ and } \neg\mathsf{cv}(p_2 \,||_A\, q) \end{array} \right)$$
$$q \xrightarrow{a,\phi_1} q' \quad \wedge \quad \models v_2'(\phi_1 \wedge \iota(q)) \quad \wedge \quad \models \hat{v}_1'(\iota(p_1))$$
$$\Longrightarrow \qquad \text{(by Def. 4.6)}$$
$$q \xrightarrow{a,\phi_1} q' \quad \wedge \quad \models v_2'(\phi_1 \wedge \iota(q)) \quad \wedge \quad \mathcal{U}_d(p_1, \hat{v}_1)$$
$$\Longrightarrow \qquad \text{(since } \langle (p_1, \hat{v}_1), (p_2, \hat{v}_2) \rangle \in \sim_t)$$
$$q \xrightarrow{a,\phi_1} q' \quad \wedge \quad \models v_2'(\phi_1 \wedge \iota(q)) \quad \wedge \quad \mathcal{U}_d(p_2, \hat{v}_2)$$
$$\Longrightarrow \qquad \text{(by Def. 4.6, } \hat{v}_2' = \hat{v}_2[\kappa(p_2) \leftarrow\!\shortmid 0] + d)$$
$$q \xrightarrow{a,\phi_1} q' \quad \wedge \quad \models v_2'(\phi_1 \wedge \iota(q)) \quad \wedge \quad \models \hat{v}_2'(\iota(p_2))$$
$$\Longrightarrow \qquad \text{(by Table 5.3)}$$
$$p_2 \,||_A\, q \xrightarrow{a,\phi_1} \overline{\mathsf{ck}}(p_2) \,||_A\, q' \quad \wedge \quad \models v_2'(\phi_1 \wedge \iota(q)) \quad \wedge \quad \models \hat{v}_2'(\iota(p_2))$$
$$\Longrightarrow \qquad \text{(by rule } \textbf{alpha})$$
$$p_2 \,||_A\, q \xrightarrow{a,\phi_1}{}' p_2'' \,||_A\, q''' \quad \wedge \quad p_2'' \,||_A\, q''' \simeq_\alpha \overline{\mathsf{ck}}(p_2) \,||_A\, q' \quad \wedge \quad \neg\mathsf{cv}(p_2'' \,||_A\, q''')$$
$$\wedge \quad \models v_2'(\phi_1 \wedge \iota(q)) \quad \wedge \quad \models \hat{v}_2'(\iota(p_2))$$
$$\Longrightarrow \qquad \left(\begin{array}{c} \text{since } v_2 \restriction fv(p_2) = \hat{v}_2 \restriction fv(p_2),\ \neg\mathsf{cv}(p_2 \,||_A\, q), \\ v_2' = v_2[\kappa(p_2 \,||_A\, q) \leftarrow\!\shortmid 0] + d, \text{ logic, and Table 5.3} \end{array} \right)$$
$$p_2 \,||_A\, q \xrightarrow{a,\phi_1}{}' p_2'' \,||_A\, q''' \quad \wedge \quad \models v_2'(\phi_1 \wedge \iota(p_2 \,||_A\, q))$$
$$\Longrightarrow \qquad \text{(by Def. 4.6)}$$
$$(p_2 \,||_A\, q, v_2) \xrightarrow{a(d)} (p_2'' \,||_A\, q''', v_2') \qquad \text{(B.4)}$$

Choose $q^\star$ such that $q^\star \simeq_\alpha q'$ and $\kappa(q^\star)$ is a set of fresh variables not appearing in any of the processes above. Thus $\kappa(q^\star) \cap fv(\overline{\mathsf{ck}}(p_1)) = \kappa(q^\star) \cap fv(\overline{\mathsf{ck}}(p_2)) = \varnothing$. As a consequence, $r \equiv p_1'' \,||_A\, q'' \simeq_\alpha \overline{\mathsf{ck}}(p_1) \,||_A\, q' \simeq_\alpha \overline{\mathsf{ck}}(p_1) \,||_A\, q^\star$, $p_2'' \,||_A\, q''' \simeq_\alpha \overline{\mathsf{ck}}(p_2) \,||_A\, q' \simeq_\alpha \overline{\mathsf{ck}}(p_2) \,||_A\, q^\star$, $\neg\mathsf{cv}(\overline{\mathsf{ck}}(p_1) \,||_A\, q^\star)$, and $\neg\mathsf{cv}(\overline{\mathsf{ck}}(p_2) \,||_A\, q^\star)$. By Theorem 5.16,

$$p_1'' \,||_A\, q'' \sim_{\&} \overline{\mathsf{ck}}(p_1) \,||_A\, q^\star \quad \text{and} \quad p_2'' \,||_A\, q''' \sim_{\&} \overline{\mathsf{ck}}(p_2) \,||_A\, q^\star. \qquad \text{(B.5)}$$

Moreover, since $v_1 \restriction fv(p_1) = \hat{v}_1 \restriction fv(p_1)$, $v_2 \restriction fv(p_2) = \hat{v}_2 \restriction fv(p_2)$, $fv(p_1') \subseteq fv(p_1) \cup \kappa(p_1)$,

$fv(p_2') \subseteq fv(p_2) \cup \kappa(p_2)$, and $fv(q^\star) = fv(q') \subseteq fv(q) \cup \kappa(q)$, then

$$\begin{array}{l} v_1' \restriction fv(p_1') = (\hat{v}_1[\kappa(p_1) \leftarrow\!\shortmid 0] + d) \restriction fv(p_1'), \\ v_2' \restriction fv(p_2') = (\hat{v}_2[\kappa(p_2) \leftarrow\!\shortmid 0] + d) \restriction fv(p_2'), \quad \text{and} \\ v_1' \restriction fv(q^\star) = v_2' \restriction fv(q^\star). \end{array} \tag{B.6}$$

Since $\langle (p_1, \hat{v}_1), (p_2, \hat{v}_2) \rangle \in \sim_t$, by (B.4), (B.5), (B.6), and Theorem 4.20, finally we have,

$$(p_2 \,||_A\, q, v_2) \xrightarrow{a(d)} (p_2'' \,||_A\, q''', v_2') \quad \wedge \quad \langle (p_1'' \,||_A\, q'', v_1'), (p_2'' \,||_A\, q''', v_2') \rangle \in (\sim_t \cup R^{\mathrm{m}})^*.$$

Sub-subcase (***)

$$\begin{array}{l} p_1 \xrightarrow{a,\phi_1'} p_1' \quad \wedge \quad q \xrightarrow{a,\phi_1''} q' \quad \wedge \quad r \simeq_\alpha p_1' \,||_A\, q' \quad \wedge \quad \neg \mathsf{cv}(r) \\ \quad \wedge \quad \models v_1'(\phi_1' \wedge \phi_1'' \wedge \iota(p_1) \wedge \iota(q)) \\ \Longrightarrow \qquad \text{(by logic, Def. 5.9 } (\simeq_\alpha) \text{ with } r \equiv p_1'' \,||_A\, q'' \text{ , and Def. 5.4 } (\mathsf{cv})) \\ \quad p_1 \xrightarrow{a,\phi_1'} p_1' \quad \wedge \quad \models v_1'(\phi_1' \wedge \iota(p_1)) \quad \wedge \quad p_1'' \simeq_\alpha p_1' \quad \wedge \quad \neg \mathsf{cv}(p_1'') \\ \quad \wedge \quad q \xrightarrow{a,\phi_1''} q' \quad \wedge \quad \models v_1'(\phi_1'' \wedge \iota(q)) \wedge \quad q'' \simeq_\alpha q' \quad \wedge \quad \neg \mathsf{cv}(q'') \\ \Longrightarrow \qquad \text{(by rule } \textbf{alpha}) \\ \quad p_1 \xrightarrow{a,\phi_1'}{}' p_1'' \quad \wedge \quad \models v_1'(\phi_1' \wedge \iota(p_1)) \quad \wedge \quad q \xrightarrow{a,\phi_1''} q' \quad \wedge \quad \models v_1'(\phi_1'' \wedge \iota(q)) \\ \Longrightarrow \qquad \left(\begin{array}{r} \text{since } v_1 \restriction fv(p_1) = \hat{v}_1 \restriction fv(p_1),\ \hat{v}_1' = \hat{v}_1[\kappa(p_1) \leftarrow\!\shortmid 0] + d, \\ v_1 \restriction fv(q) = v_2 \restriction fv(q),\ v_2' = v_2[\kappa(p_2 \,||_A\, q) \leftarrow\!\shortmid 0] + d, \text{ and } \neg \mathsf{cv}(p_1 \,||_A\, q) \end{array} \right) \\ \quad p_1 \xrightarrow{a,\phi_1'}{}' p_1'' \quad \wedge \quad \models \hat{v}_1'(\phi_1' \wedge \iota(p_1)) \quad \wedge \quad q \xrightarrow{a,\phi_1''} q' \quad \wedge \quad \models v_2'(\phi_1'' \wedge \iota(q)) \\ \Longrightarrow \qquad \text{(by Def. 4.6)} \\ \quad (p_1, \hat{v}_1) \xrightarrow{a(d)} (p_1'', \hat{v}_1') \quad \wedge \quad q \xrightarrow{a,\phi_1''} q' \quad \wedge \quad \models v_2'(\phi_1'' \wedge \iota(q)) \\ \Longrightarrow \qquad \text{(since } \langle (p_1, \hat{v}_1), (p_2, \hat{v}_2) \rangle \in \sim_t) \\ \quad (p_2, \hat{v}_2) \xrightarrow{a(d)} (p_2''', \hat{v}_2') \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2') \rangle \in \sim_t \\ \quad \wedge \quad q \xrightarrow{a,\phi_1''} q' \quad \wedge \quad \models v_2'(\phi_1'' \wedge \iota(q)) \\ \Longrightarrow \qquad \text{(by Def. 4.6, } \hat{v}_2' = \hat{v}_2[\kappa(p_2) \leftarrow\!\shortmid 0] + d) \\ \quad p_2 \xrightarrow{a,\phi_2'}{}' p_2''' \quad \wedge \quad \models \hat{v}_2'(\phi_2' \wedge \iota(p_2)) \\ \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2') \rangle \in \sim_t \quad \wedge \quad q \xrightarrow{a,\phi_1''} q' \quad \wedge \quad \models v_2'(\phi_1'' \wedge \iota(q)) \\ \Longrightarrow \qquad \text{(by rule } \textbf{alpha}) \\ \quad p_2 \xrightarrow{a,\phi_2'} p_2' \quad \wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models \hat{v}_2'(\phi_2' \wedge \iota(p_2)) \\ \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2') \rangle \in \sim_t \quad \wedge \quad q \xrightarrow{a,\phi_1''} q' \quad \wedge \quad \models v_2'(\phi_1'' \wedge \iota(q)) \end{array}$$

$\Longrightarrow$ (by Table 5.3, $\phi_2 \equiv \phi_2' \wedge \phi_1''$)

$p_2 \,||_A\, q \xrightarrow{a,\phi_2} p_2' \,||_A\, q' \quad \wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models \hat{v}_2'(\phi_2' \wedge \iota(p_2))$
$\wedge \quad \models v_2'(\phi_1'' \wedge \iota(q)) \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2') \rangle \in \sim_t$

$\Longrightarrow$ (by rule **alpha**)

$p_2 \,||_A\, q \xrightarrow{a,\phi_2}{}' p_2'' \,||_A\, q''' \quad \wedge \quad p_2'' \,||_A\, q''' \simeq_\alpha p_2' \,||_A\, q' \quad \wedge \quad \neg \mathsf{cv}(p_2'' \,||_A\, q''')$
$\wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models \hat{v}_2'(\phi_2' \wedge \iota(p_2))$
$\wedge \quad \models v_2'(\phi_1'' \wedge \iota(q)) \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2') \rangle \in \sim_t$

$\Longrightarrow$ (since $v_2 \restriction fv(p_2) = \hat{v}_2 \restriction fv(p_2)$, $\neg \mathsf{cv}(p_2 \,||_A\, q)$, logic and Table 5.3)

$p_2 \,||_A\, q \xrightarrow{a,\phi_2}{}' p_2'' \,||_A\, q''' \quad \wedge \quad p_2'' \,||_A\, q''' \simeq_\alpha p_2' \,||_A\, q' \quad \wedge \quad \neg \mathsf{cv}(p_2'' \,||_A\, q''')$
$\wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models v_2'(\phi_2 \wedge \iota(p_2 \,||_A\, q)) \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2') \rangle \in \sim_t$

$\Longrightarrow$ (by Def. 4.6, Def. 5.9 ($\simeq_\alpha$))

$(p_2 \,||_A\, q, v_2) \xrightarrow{a(d)} (p_2'' \,||_A\, q''', v_2') \quad \wedge \quad p_2'' \simeq_\alpha p_2'''$
$\wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2') \rangle \in \sim_t$

$\Longrightarrow$ (by Theorems 5.16 and 4.20)

$$(p_2 \,||_A\, q, v_2) \xrightarrow{a(d)} (p_2'' \,||_A\, q''', v_2') \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2'', \hat{v}_2') \rangle \in \sim_t \tag{B.7}$$

As before, we can choose $q^\star$ such that $p_1'' \,||_A\, q'' \simeq_\alpha p_1' \,||_A\, q' \simeq_\alpha p_1'' \,||_A\, q^\star$, $p_2'' \,||_A\, q''' \simeq_\alpha p_2' \,||_A\, q''' \simeq_\alpha p_2'' \,||_A\, q^\star$, $\neg \mathsf{cv}(p_1'' \,||_A\, q^\star)$, and $\neg \mathsf{cv}(p_2'' \,||_A\, q^\star)$. By Theorem 5.16,

$$p_1'' \,||_A\, q'' \sim_\& p_1'' \,||_A\, q^\star \quad \text{and} \quad p_2'' \,||_A\, q''' \sim_\& p_2'' \,||_A\, q^\star. \tag{B.8}$$

Moreover, since $v_1 \restriction fv(p_1) = \hat{v}_1 \restriction fv(p_1)$, $v_2 \restriction fv(p_2) = \hat{v}_2 \restriction fv(p_2)$, $fv(p_1'') \subseteq fv(p_1) \cup \kappa(p_1)$, $fv(p_2'') \subseteq fv(p_2) \cup \kappa(p_2)$, and $fv(q^\star) = fv(q') \subseteq fv(q) \cup \kappa(q)$, then

$$\begin{aligned} &v_1' \restriction fv(p_1'') = \hat{v}_1' \restriction fv(p_1''), \\ &v_2' \restriction fv(p_2'') = \hat{v}_2' \restriction fv(p_2''), \quad \text{and} \\ &v_1' \restriction fv(q^\star) = v_2' \restriction fv(q^\star). \end{aligned} \tag{B.9}$$

Since $\langle (p_1'', \hat{v}_1'), (p_2'', \hat{v}_2') \rangle \in \sim_t$, by (B.7), (B.8), (B.9), and Theorem 4.20, finally we have,

$$(p_2 \,||_A\, q, v_2) \xrightarrow{a(d)} (p_2'' \,||_A\, q''', v_2') \quad \wedge \quad \langle (p_1'' \,||_A\, q'', v_1'), (p_2'' \,||_A\, q''', v_2') \rangle \in (\sim_t \cup R^{\mathrm{m}})^*.$$

Subcase 2.

$\mathcal{U}_d(p_1 \,||_A\, q, v_1)$

$\Longrightarrow$ (by Def. 4.6 and $v_1' = v_1[\kappa(p_1 \,||_A\, q) \mapsto 0] + d$)

$\models v_1'(\iota(p_1 \,||_A\, q))$

$\Longrightarrow$ (by Table 5.3)

$$
\begin{array}{l}
\models v_1'(\iota(p_1)) \quad \wedge \quad \models v_1'(\iota(q)) \\
\Longrightarrow \qquad \left(\begin{array}{r} \text{since } v_1 \restriction fv(p_1) = \hat{v}_1 \restriction fv(p_1),\ \hat{v}_1' = \hat{v}_1[\kappa(p_1) \hookleftarrow 0] + d, \\ v_1 \restriction fv(q) = v_2 \restriction fv(q),\ v_2' = v_2[\kappa(p_2 \mid\mid_A q) \hookleftarrow 0] + d,\ \text{and } \neg \mathsf{cv}(p_2 \mid\mid_A q) \end{array} \right) \\
\models \hat{v}_1'(\iota(p_1)) \quad \wedge \quad \models v_2'(\iota(q)) \\
\Longrightarrow \qquad \text{(by Def. 4.6)} \\
\mathcal{U}_d(p_1, \hat{v}_1) \quad \wedge \quad \models v_2'(\iota(q)) \\
\Longrightarrow \qquad \text{(since } \langle (p_1, \hat{v}_1), (p_2, \hat{v}_2) \rangle \in \sim_t \text{)} \\
\mathcal{U}_d(p_2, \hat{v}_2) \quad \wedge \quad \models v_2'(\iota(q)) \\
\Longrightarrow \qquad \text{(by Def. 4.6 and } \hat{v}_2' = \hat{v}_2[\kappa(p_2) \hookleftarrow 0] + d \text{)} \\
\models \hat{v}_2'(\iota(p_2)) \quad \wedge \quad \models v_2'(\iota(q)) \\
\Longrightarrow \qquad \text{(since } v_2 \restriction fv(p_2) = \hat{v}_2 \restriction fv(p_2), \text{ and } \neg \mathsf{cv}(p_2 \mid\mid_A q) \text{)} \\
\models v_2'(\iota(p_2)) \quad \wedge \quad \models v_2'(\iota(q)) \\
\Longrightarrow \qquad \text{(by Table 5.3)} \\
\models v_2'(\iota(p_2 \mid\mid_A q)) \\
\Longrightarrow \qquad \text{(by Def. 4.6)} \\
\mathcal{U}_d(p_2 \mid\mid_A q, v_2)
\end{array}
$$

Case $\langle (\overline{\mathsf{ck}}(p_1) \mid\mid_A q, v_1), (\overline{\mathsf{ck}}(p_2) \mid\mid_A q, v_2) \rangle \in R^{\mathrm{m}}$.

Subcase 1.

$$
\begin{array}{l}
(\overline{\mathsf{ck}}(p_1) \mid\mid_A q, v_1) \xrightarrow{a(d)} (r, v_1') \\
\Longrightarrow \qquad \text{(by Def. 4.6 and } v_1' = v_1[\kappa(\overline{\mathsf{ck}}(p_1) \mid\mid_A q) \hookleftarrow 0] + d \text{)} \\
\overline{\mathsf{ck}}(p_1) \mid\mid_A q \xrightarrow{a,\phi_1}{}' r \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(\overline{\mathsf{ck}}(p_1) \mid\mid_A q)) \\
\Longrightarrow \qquad \text{(by rule } \textbf{alpha}\text{)} \\
\overline{\mathsf{ck}}(p_1) \mid\mid_A q \xrightarrow{a,\phi_1} r' \quad \wedge \quad r \simeq_\alpha r' \quad \wedge \quad \neg \mathsf{cv}(r) \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(\overline{\mathsf{ck}}(p_1) \mid\mid_A q)) \\
\Longrightarrow \qquad \text{(by Table 5.3)} \\
(\ (\overline{\mathsf{ck}}(p_1) \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad r' \equiv p_1' \mid\mid_A \overline{\mathsf{ck}}(q)) \qquad (*) \\
\quad \vee \quad (q \xrightarrow{a,\phi_1} q' \quad \wedge \quad r' \equiv \overline{\mathsf{ck}}(p_1) \mid\mid_A q') \qquad (**) \\
\quad \vee \quad (\overline{\mathsf{ck}}(p_1) \xrightarrow{a,\phi_1'} p_1' \quad \wedge \quad q \xrightarrow{a,\phi_1''} q' \quad \wedge \quad r' \equiv p_1' \mid\mid_A q' \quad \wedge \quad \phi_1 \equiv \phi_1' \wedge \phi_1'')\ ) \quad (***) \\
\wedge \quad r \simeq_\alpha r' \quad \wedge \quad \neg \mathsf{cv}(r) \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(\overline{\mathsf{ck}}(p_1)) \wedge \iota(q))
\end{array}
$$

Since the proof proceeds in a similar way to the previous case, we only show the proof for case (*).

Sub-subcase (*)

$$
\begin{array}{l}
\overline{\mathsf{ck}}(p_1) \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad r \simeq_\alpha p_1' \mid\mid_A \overline{\mathsf{ck}}(q) \\
\quad \wedge \quad \neg \mathsf{cv}(r) \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(\overline{\mathsf{ck}}(p_1)) \wedge \iota(q))
\end{array}
$$

$\Longrightarrow$ (by rules (5.1), page 52)

$p_1 \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad r \simeq_\alpha p_1' \,||_A\, \overline{\mathsf{ck}}(q) \quad \wedge \quad \neg\mathsf{cv}(r) \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(p_1) \wedge \iota(q))$

$\Longrightarrow$ (by logic, Def. 5.9 ($\simeq_\alpha$) with $r \equiv p_1'' \,||_A\, q'$, and Def. 5.4 (cv))

$p_1 \xrightarrow{a,\phi_1} p_1' \quad \wedge \quad \models v_1'(\phi_1 \wedge \iota(p_1)) \quad \wedge \quad p_1'' \simeq_\alpha p_1' \quad \wedge \quad \neg\mathsf{cv}(p_1'') \quad \wedge \quad \models v_1'(\iota(q))$

$\Longrightarrow$ (by rule **alpha**)

$p_1 \xrightarrow{a,\phi_1}{}' p_1'' \quad \wedge \quad \models v_1'(\phi \wedge \iota(p_1)) \quad \wedge \quad \models v_1'(\iota(q))$

$\Longrightarrow$ $\left(\begin{array}{c}\text{since } v_1 \restriction fv(p_1) = (\hat{v}_1[\kappa(p_1) \hookleftarrow 0] + d') \restriction fv(p_1),\\ \hat{v}_1' = \hat{v}_1[\kappa(p_1) \hookleftarrow 0] + d' + d,\ v_1 \restriction fv(q) = v_2 \restriction fv(q),\\ v_2' = v_2[\kappa(\overline{\mathsf{ck}}(p_2) \,||_A\, q) \hookleftarrow 0] + d, \text{ and } \neg\mathsf{cv}(\overline{\mathsf{ck}}(p_2) \,||_A\, q)\end{array}\right)$

$p_1 \xrightarrow{a,\phi_1}{}' p_1'' \quad \wedge \quad \models \hat{v}_1'(\phi_1 \wedge \iota(p_1)) \quad \wedge \quad \models v_2'(\iota(q))$

$\Longrightarrow$ (by Def. 4.6)

$(p_1, \hat{v}_1) \xrightarrow{a(d'+d)} (p_1'', \hat{v}_1') \quad \wedge \quad \models v_2'(\iota(q))$

$\Longrightarrow$ (since $\langle (p_1, \hat{v}_1), (p_2, \hat{v}_2)\rangle \in \sim_t$)

$(p_2, \hat{v}_2) \xrightarrow{a(d'+d)} (p_2''', \hat{v}_2') \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$

$\Longrightarrow$ (by Def. 4.6, $\hat{v}_2' = \hat{v}_2[\kappa(p_2) \hookleftarrow 0] + d' + d$)

$p_2 \xrightarrow{a,\phi_2}{}' p_2''' \quad \wedge \quad \models \hat{v}_2'(\phi_2 \wedge \iota(p_2))$
$\wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$

$\Longrightarrow$ (by rule **alpha**)

$p_2 \xrightarrow{a,\phi_2} p_2' \quad \wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models \hat{v}_2'(\phi_2 \wedge \iota(p_2))$
$\wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$

$\Longrightarrow$ (by rules (5.1), page 52)

$\overline{\mathsf{ck}}(p_2) \xrightarrow{a,\phi_2} p_2' \quad \wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models \hat{v}_2'(\phi_2 \wedge \iota(\overline{\mathsf{ck}}(p_2)))$
$\wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$

$\Longrightarrow$ (by Table 5.3)

$\overline{\mathsf{ck}}(p_2) \,||_A\, q \xrightarrow{a,\phi_2} p_2' \,||_A\, \overline{\mathsf{ck}}(q) \quad \wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models \hat{v}_2'(\phi_2 \wedge \iota(\overline{\mathsf{ck}}(p_2)))$
$\wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$

$\Longrightarrow$ (by rule **alpha**)

$\overline{\mathsf{ck}}(p_2) \,||_A\, q \xrightarrow{a,\phi_2}{}' p_2'' \,||_A\, q'' \quad \wedge \quad p_2'' \,||_A\, q'' \simeq_\alpha p_2' \,||_A\, \overline{\mathsf{ck}}(q) \quad \wedge \quad \neg\mathsf{cv}(p_2'' \,||_A\, q'') \quad \wedge$
$p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models \hat{v}_2'(\phi_2 \wedge \iota(\overline{\mathsf{ck}}(p_2))) \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t \quad \wedge \quad \models v_2'(\iota(q))$

$\Longrightarrow$ $\left(\begin{array}{c}\text{since } v_2 \restriction fv(p_2) = (\hat{v}_2[\kappa(p_2) \hookleftarrow 0] + d) \restriction fv(p_2),\\ \neg\mathsf{cv}(\overline{\mathsf{ck}}(p_2) \,||_A\, q), \text{ logic, and Table 5.3}\end{array}\right)$

$\overline{\mathsf{ck}}(p_2) \,||_A\, q \xrightarrow{a,\phi_2}{}' p_2'' \,||_A\, q'' \quad \wedge \quad p_2'' \,||_A\, q'' \simeq_\alpha p_2' \,||_A\, \overline{\mathsf{ck}}(q)$
$\wedge \quad p_2''' \simeq_\alpha p_2' \quad \wedge \quad \models v_2'(\phi_2 \wedge \iota(\overline{\mathsf{ck}}(p_2) \,||_A\, q)) \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2')\rangle \in \sim_t$

$$\Longrightarrow \qquad \text{(by Def. 4.6, Def. 5.9 } (\simeq_\alpha))$$

$$(\overline{\mathsf{ck}}(p_2) \,||_A\, q, v_2) \xrightarrow{a(d)} (p_2'' \,||_A\, q'', v_2') \quad \wedge \quad p_2'' \simeq_\alpha p_2''' \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2''', \hat{v}_2') \rangle \in \sim_t$$

$$\Longrightarrow \qquad \text{(by Theorems 5.16 and 4.20)}$$

$$(\overline{\mathsf{ck}}(p_2) \,||_A\, q, v_2) \xrightarrow{a(d)} (p_2'' \,||_A\, q'', v_2') \quad \wedge \quad \langle (p_1'', \hat{v}_1'), (p_2'', \hat{v}_2') \rangle \in \sim_t \qquad (\dagger\dagger)$$

Since $p_1'' \,||_A\, q' \simeq_\alpha p_1' \,||_A\, \overline{\mathsf{ck}}(q)$ and $p_2'' \,||_A\, q'' \simeq_\alpha p_2' \,||_A\, \overline{\mathsf{ck}}(q)$, $q' \simeq_\alpha \overline{\mathsf{ck}}(q) \simeq_\alpha q''$. Thus, $fv(q') = fv(\overline{\mathsf{ck}}(q)) = fv(q'')$ and $\kappa(q') = \kappa(q'') = \kappa(\overline{\mathsf{ck}}(q)) = \varnothing$. As a consequence $p_1'' \,||_A\, q' \simeq_\alpha p_1'' \,||_A\, \overline{\mathsf{ck}}(q)$, $p_2'' \,||_A\, q'' \simeq_\alpha p_2'' \,||_A\, \overline{\mathsf{ck}}(q)$, $\neg\mathsf{cv}(p_1'' \,||_A\, \overline{\mathsf{ck}}(q))$, and $\neg\mathsf{cv}(p_2'' \,||_A\, \overline{\mathsf{ck}}(q))$. Together with the fact that $fv(p_1'') \subseteq fv(p_1) \cup \kappa(p_1)$, $fv(p_2'') \subseteq fv(p_2) \cup \kappa(p_2)$, and $fv(\overline{\mathsf{ck}}(q)) \subseteq fv(q) \cup \kappa(q)$, we have,

$$(\dagger\dagger) \Longrightarrow$$

$$(\overline{\mathsf{ck}}(p_2) \,||_A\, q, v_2) \xrightarrow{a(d)} (p_2'' \,||_A\, q'', v_2') \;\wedge\; \langle (p_1'' \,||_A\, \overline{\mathsf{ck}}(q), v_1'), (p_2'' \,||_A\, \overline{\mathsf{ck}}(q), v_2') \rangle \in R^{\mathrm{m}}$$

$$\Longrightarrow \qquad \text{(by Theorems 5.16 and 4.20)}$$

$$(\overline{\mathsf{ck}}(p_2) \,||_A\, q, v_2) \xrightarrow{a(d)} (p_2'' \,||_A\, q'', v_2') \;\wedge\; \langle (p_1'' \,||_A\, q', v_1'), (p_2'' \,||_A\, q'', v_2') \rangle \in (\sim_t \cup R^{\mathrm{m}})^*$$

Subcase 2. It follows a similar reasoning to the second transfer property of the previous case.

Since $\simeq_\alpha$ is a congruence, the relations above suffice to prove the theorem. *Q.E.D.*

Appendix C

Proofs from Chapter 6

C.1 Proof of Theorem 6.6

Theorem 6.6. *For every term $p \in \heartsuit^b$ there is a non-redundant basic term q such that $p = q$ can be proved by means of the axiom system $ax(\heartsuit^b) - \mathbf{D}$ and α-conversion.*

Proof. We proceed by structural induction.

Case $\mathbf{0}$.

$$\mathbf{0} \overset{\text{CR1,I1}}{=} \{\!|x|\!\}\, \mathbf{tt} \gg\!\!> \mathbf{0}$$

Case $a; p$. Take a fresh variable x and, by induction hypothesis, assume p is a non-redundant basic term. Then

$$a; p \overset{\text{G1}}{=} \mathbf{tt} \mapsto a; p \overset{\text{CR1,I1}}{=} \{\!|x|\!\}\, \mathbf{tt} \gg\!\!> (\mathbf{tt} \mapsto a; p)$$

Case $\phi \mapsto p$. By induction assume

$$p = \{\!|x|\!\}\, \psi \gg\!\!> \left(\sum_{i \in I} \phi_i \mapsto a_i; p_i \right)$$

where each p_i is already a non-redundant basic term. Moreover, by α-conversion, take x such that $x \notin var(\phi)$. Then

$$\begin{aligned} \phi \mapsto p \quad &\overset{\text{G4,G3}}{=} \quad \{\!|x|\!\}\, \psi \gg\!\!> \left(\phi \mapsto \sum_{i \in I} \phi_i \mapsto a_i; p_i \right) \\ &\overset{\text{G5,G2}}{=} \quad \{\!|x|\!\}\, \psi \gg\!\!> \left(\sum_{i \in I} (\phi \wedge \phi_i) \mapsto a_i; p_i \right) \end{aligned}$$

$$\begin{aligned}
&\overset{\text{Prop. 6.2.5}}{=} && \{\!|x|\!\}\,\psi \gg\!\!> \psi \mapsto \left(\sum_{i\in I}(\phi\wedge\phi_i)\mapsto a_i;p_i\right)\\
&\overset{\text{G5,G2}}{=} && \{\!|x|\!\}\,\psi \gg\!\!> \left(\sum_{i\in I}(\psi\wedge\phi\wedge\phi_i)\mapsto a_i;p_i\right)\\
&\overset{\text{L1,Stp,Prop. 6.2.4}}{=} && \{\!|x|\!\}\,\psi \gg\!\!> \left(\sum_{i\in I-K}(\psi\wedge\phi\wedge\phi_i)\mapsto a_i;p_i\right)
\end{aligned}$$

where $K \overset{\text{def}}{=} \{i\in I \mid \models (\psi\wedge\phi\wedge\phi_i)\Leftrightarrow \mathbf{ff}\}$.

Case $p+q$. By induction assume

$$p = \{\!|x|\!\}\,\psi \gg\!\!> \left(\sum_{i\in I}\phi_i\mapsto a_i;p_i\right) \quad\text{and}\quad q = \{\!|y|\!\}\,\psi' \gg\!\!> \left(\sum_{j\in J}\phi'_j\mapsto b_j;q_j\right)$$

where each p_i and q_j are already non-redundant basic terms. Moreover, by α-conversion, we can consider $x=y$. Then

$$\begin{aligned}
p+q &\overset{\text{ind.}}{=} \{\!|x|\!\}\,\psi \gg\!\!> \left(\sum_{i\in I}\phi_i\mapsto a_i;p_i\right) + \{\!|x|\!\}\,\psi' \gg\!\!> \left(\sum_{j\in J}\phi'_j\mapsto b_j;q_j\right)\\
&\overset{\text{CR4,I4}}{=} \{\!|x|\!\}\,\left(\sum_{i\in I}\psi \gg\!\!> (\phi_i\mapsto a_i;p_i) + \sum_{j\in J}\psi' \gg\!\!> (\phi'_j\mapsto b_j;q_j)\right)\\
&\overset{\text{A1,A2,Prop. 6.2.1}}{=} \{\!|x|\!\}\,\left(\sum_{i\in I}(\psi \gg\!\!> (\phi_i\mapsto a_i;p_i) + \psi' \gg\!\!> (\phi'_1\mapsto b_1;q_1))\right.\\
&\qquad\left. + \sum_{j\in J}(\psi' \gg\!\!> (\phi'_j\mapsto b_j;q_j) + \psi \gg\!\!> (\phi_1\mapsto a_1;p_1))\right)\\
&\overset{\text{I5}}{=} \{\!|x|\!\}\,\left(\sum_{i\in I}(\psi\vee\psi') \gg\!\!> ((\psi\wedge\phi_i)\mapsto a_i;p_i + (\psi'\wedge\phi'_1)\mapsto b_1;q_1)\right.\\
&\qquad\left. + \sum_{j\in J}(\psi\vee\psi') \gg\!\!> ((\psi'\wedge\phi'_j)\mapsto b_j;q_j + (\psi\wedge\phi_1)\mapsto a_1;p_1)\right)\\
&\overset{\text{I4}}{=} \{\!|x|\!\}\,(\psi\vee\psi') \gg\!\!> \left(\sum_{i\in I}((\psi\wedge\phi_i)\mapsto a_i;p_i + (\psi'\wedge\phi'_1)\mapsto b_1;q_1)\right.\\
&\qquad\left. + \sum_{j\in J}((\psi'\wedge\phi'_j)\mapsto b_j;q_j + (\psi\wedge\phi_1)\mapsto a_1;p_1)\right)\\
&\overset{\text{A1,A2,Prop. 6.2.1}}{=} \{\!|x|\!\}\,(\psi\vee\psi') \gg\!\!> \left(\sum_{i\in I}(\psi\wedge\phi_i)\mapsto a_i;p_i + \sum_{j\in J}(\psi'\wedge\phi'_j)\mapsto b_j;q_j\right)
\end{aligned}$$

Notice that the guards in all the summands, in conjunction with the invariant, are still satisfiable.

Case $\{\!|C|\!\}\, p$. By induction hypothesis assume

$$p = \{\!|x|\!\}\, \psi \gg \left(\sum_{i \in I} \phi_i \mapsto a_i; p_i \right)$$

where each p_i is already a non-redundant basic term. Then

$$\begin{aligned}
\{\!|C|\!\}\, p &\overset{\text{ind.}}{=} \{\!|C|\!\}\, \{\!|x|\!\}\, \psi \gg \left(\sum_{i \in I} \phi_i \mapsto a_i; p_i \right) \\
&\overset{\text{CR3}}{=} \{\!|C \cup \{x\}|\!\}\, \psi \gg \left(\sum_{i \in I} \phi_i \mapsto a_i; p_i \right) \\
&\overset{\text{CR2}}{=} \{\!|x|\!\}\, (\xi_{\{x/C\}} \psi) \gg \left(\sum_{i \in I-K} (\xi_{\{x/C\}} \phi_i) \mapsto a_i; \xi_{\{x/C\}} p_i \right)
\end{aligned}$$

where $K \overset{\text{def}}{=} \{i \in I \mid \models \xi_{\{x/C\}}(\psi \wedge \phi_i) \Leftrightarrow \mathbf{ff}\}$.

Case $\psi \gg p$. By induction hypothesis assume

$$p = \{\!|x|\!\}\, \psi' \gg \left(\sum_{i \in I} \phi_i \mapsto a_i; p_i \right)$$

where each p_i is already a basic term. Moreover, by α-conversion, assume that $x \notin var(\psi)$. Then

$$\begin{aligned}
\psi \gg p \quad &\overset{\text{ind.}}{=} \quad \psi \gg \{\!|x|\!\}\, \psi' \gg \left(\sum_{i \in I} \phi_i \mapsto a_i; p_i \right) \\
&\overset{\text{I3,I2}}{=} \quad \{\!|x|\!\}\, (\psi \wedge \psi') \gg \left(\sum_{i \in I} \phi_i \mapsto a_i; p_i \right) \\
&\overset{\text{Prop. 6.2.5,G5,G2}}{=} \{\!|x|\!\}\, (\psi \wedge \psi') \gg \left(\sum_{i \in I} (\psi \wedge \psi' \wedge \phi_i) \mapsto a_i; p_i \right) \\
&\overset{\text{L1,Stp,Prop. 6.2.4}}{=} \{\!|x|\!\}\, (\psi \wedge \psi') \gg \left(\sum_{i \in I-K} (\psi \wedge \psi' \wedge \phi_i) \mapsto a_i; p_i \right)
\end{aligned}$$

where $K \overset{\text{def}}{=} \{i \in I \mid \models (\psi \wedge \psi' \wedge \phi_i) \Leftrightarrow \mathbf{ff}\}$. *Q.E.D.*

Appendix D

Some Concepts of Probability Theory

In this appendix we recall some notions of probabilities which are necessary for the understanding of the third part of the thesis. The reader is referred to (Shiryaev, 1996; Lang, 1993; Rudin, 1974) for further reading.

D.1 Probability Spaces and Measurable Functions

Definition D.1. Let Ω be a set which we call *sample space*. A collection $\mathcal{F}$ of subsets of Ω is a *σ-algebra* if

1. $\Omega \in \mathcal{F}$;
2. $A \in \mathcal{F} \implies A^c \in \mathcal{F}$; and
3. $\forall i \in \mathbb{N}. A_i \in \mathcal{F} \implies \bigcup_{i\in\mathbb{N}} A_i \in \mathcal{F}$

The elements of a σ-algebra are called *measurable sets* and the pair $(\Omega, \mathcal{F})$ is a *measurable space*. A *probability measure* on $(\Omega, \mathcal{F})$ is a function $P : \mathcal{F} \to \mathbb{R}_{\geq 0}$ such that the following properties hold:

1. $P(\varnothing) = 0$, $P(\Omega) = 1$.
2. Let $N \subseteq \mathbb{N}$. Let $(A_i)_{i\in N}$ be a pairwise disjoint family of measurable sets in the σ-algebra $\mathcal{F}$. Then $P\left(\bigcup_{i\in N} A_i\right) = \sum_{i\in N} P(A_i)$.

(With $\mathbb{N}$ we denote the set of non-negative integers.) The structure $(\Omega, \mathcal{F}, P)$ is called *probability space*. In particular, if there is a countable set $A \in \Omega$ such that $\sum_{a\in A} P(\{a\}) = 1$, we say that P is a *discrete probability measure* and $(\Omega, \mathcal{F}, P)$ is a *discrete probability space*.

The support set of a probability measure is the smallest subset of the sample space whose measure is 1. That is, the *support set* of P is the set

$$\mathrm{supp}(P) \stackrel{\text{def}}{=} \Omega - \bigcup \{A \in \mathcal{F} \mid A \text{ is open}^1 \wedge P(A) = 0\}$$

[1]Formally speaking, $(\Omega, \mathcal{F})$ should be a locally compact Hausdorff space (see e.g. Rudin, 1974). In this thesis, we only use that kind of measurable spaces.

□

Definition D.2. Let $(\Omega, \mathcal{F})$ and $(\Omega', \mathcal{F}')$ be two measurable spaces. A function $f : \Omega \to \Omega'$ is said to be a *measurable function* if $f^{-1}(A) \stackrel{\text{def}}{=} \{a \mid f(a) \in A\} \in \mathcal{F}$, for all $A \in \mathcal{F}'$. □

It can be straightforwardly checked that, if P is a probability measure on $(\Omega, \mathcal{F})$, $P \circ f^{-1}$ is a probability measure on $(\Omega', \mathcal{F}')$. Thus, we can state the following proposition:

Proposition D.3. *If f is a measurable function as before and $(\Omega, \mathcal{F}, P)$ is a probability space, then $(\Omega', \mathcal{F}', P \circ f^{-1})$ is also a probability space.*

Let $\mathcal{P} = (\Omega, \mathcal{F}, P)$ be a probability space. Let $\mathcal{D} : \Omega \to \Omega'$ be an injective function. We lift $\mathcal{D}$ to subsets of Ω as usual: $\mathcal{D}(A) \stackrel{\text{def}}{=} \{\mathcal{D}(a) \mid a \in A\}$. Observe that $\mathcal{D}(\mathcal{F}) \stackrel{\text{def}}{=} \{\mathcal{D}(A) \mid A \in \mathcal{F}\}$ is a σ-algebra on the sample space $\mathcal{D}(\Omega)$. As a consequence, $\mathcal{D}$ is a measurable function on $(\mathcal{D}(\Omega), \mathcal{D}(\mathcal{F}))$ and, by the previous proposition, $\mathcal{D}(\mathcal{P}) \stackrel{\text{def}}{=} (\mathcal{D}(\Omega), \mathcal{D}(\mathcal{F}), P \circ \mathcal{D}^{-1})$ is a probability space.

Since $\mathcal{D}(\mathcal{P})$ is basically the same probability space as $\mathcal{P}$, we say that $\mathcal{D}$ is a *decoration* and we refer to $\mathcal{D}(\mathcal{P})$ as the *decoration of $\mathcal{P}$ according to $\mathcal{D}$*. Decoration is a simple notion that originates in this thesis. It is a key concept in the semantics of stochastic automata.

D.2 Borel Spaces and Probability Measures

Borel algebras are an important class of σ-algebras. In particular we are interested in the Borel algebras defined on a Cartesian product of the real numbers.

Definition D.4. For every $k \in \{1, \ldots, n\}$, let $I_k = (a_k, b_k]$ with $a_k, b_k \in \mathbb{R} \cup \{-\infty, \infty\}$. In particular, the interval $(a, \infty]$ is taken to be (a, ∞). The set $I = I_1 \times \ldots \times I_n \subseteq \mathbb{R}^n$ is called a *rectangle* and each I_k is a *side*. Let $\mathcal{I}$ be the *set of all rectangles*. The *Borel algebra* of subsets of $\mathbb{R}^n$, denoted by $\mathcal{B}(\mathbb{R}^n)$, is the smallest σ-algebra containing $\mathcal{I}$. Thus $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ is a measurable space which we call *Borel space*. □

In this work we consider only probability spaces that are isomorphic to some Borel space defined on a real hyperspace (as in Definition D.4) whose coordinates are determined by independent random variables. So the class of probability measures we use is defined by the following theorem (Shiryaev, 1996, II–§3).

Theorem D.5. *For every $i = \{1, \ldots, n\}$, let F_i be a distribution function. There is a unique probability measure P on $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ such that*

$$P((a_1, b_1], \ldots, (a_n, b_n]) = \prod_{i=1}^{n} (F_i(b_i) - F_i(a_i))$$

with $-\infty \leq a_i < b_i < \infty$, for $i \in \{1, \ldots, n\}$ and where $\prod_{i=1}^{n} d_i$ is a shorthand notation for $d_1 \cdot d_2 \cdot \ldots \cdot d_n$.

As a consequence of this theorem we can uniquely identify by $\mathcal{R}(F_1, \ldots, F_n)$ the probability space $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n), P_n)$ where $\mathcal{B}(\mathbb{R}^n)$ is the Borel algebra on $\mathbb{R}^n$ and P_n is the unique probability measure obtained as in Theorem D.5 from a given family of distribution functions $F_1, \ldots, F_n$. In particular, if $n = 0$, $\mathcal{R}()$ is the trivial probability space $(\{\varnothing\}, \{\varnothing, \{\varnothing\}\}, P_0)$ with P_0 in the obvious way.

D.3 Random Variables

Definition D.6. Let $(\Omega, \mathcal{F})$ be a measurable space and $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ the Borel space on the real line. A measurable function $\aleph : \Omega \to \mathbb{R}$ is called a *random variable*.

If P is a probability measure on $(\Omega, \mathcal{F})$, the probability measure $P_\aleph \stackrel{\text{def}}{=} P \circ \aleph^{-1}$ on $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ is the *probability distribution of* $\aleph$. The function $F_\aleph$, defined by

$$F_\aleph(d) \stackrel{\text{def}}{=} P_\aleph((-\infty, d]) = P(\{a \mid \aleph(a) \leq d\})$$

with $d \in \mathbb{R}$, is the *distribution function of* $\aleph$. The *support set* of $F_\aleph$ is defined by $\text{supp}(F_\aleph) \stackrel{\text{def}}{=} \text{supp}(P_\aleph)$. □

Definition D.7. Let $\mathcal{P} = (\Omega, \mathcal{F}, P)$ be a probability space. n random variables $\aleph_1, \ldots, \aleph_n$ on $\mathcal{P}$ are *independent* if for all $B_i \in \mathcal{B}(\mathbb{R})$, $i = 1, \ldots, n$,

$$P\left(\bigcap_{i=1}^{n} \aleph_i^{-1}(B_i)\right) = \prod_{i=1}^{n} P_{\aleph_i}(B_i).$$

□

D.4 Some distributions

In the following we describe some distribution functions that are used in this dissertation. The reader is referred to e.g. (Devore, 1982; Jain, 1991) for more information. In the following we write D.F. for "distribuation function" and p.d.f. for "probability density function" . We recall that if a random variable $\aleph$ has distribution F then its probability density function f is such that $F(t) = \int_{-\infty}^{t} f(x)dx$. In general $f(t) = 0$ whenever t is not on the support set of F. Thus we only define f on the support set.

Uniform Distribution

Parameters	a (lower limit) b (upper limit), $a < b$	
Support set	$[a, b]$	
p.d.f.	$f(t) = \dfrac{1}{b-a}$	*(Cont.)*

D.F.	$F(t) = \begin{cases} 0 & \text{if } t < a \\ \dfrac{t-a}{b-a} & \text{if } a \le t < b \\ 1 & \text{if } b \le t \end{cases}$
Mean	$\dfrac{a+b}{2}$
Variance	$\dfrac{(b-a)^2}{12}$

(Negative) Exponential Distribution

Parameters	λ (rate), $\lambda > 0$
Support set	$\mathbb{R}_{\ge 0}$
p.d.f.	$f(t) = \dfrac{e^{-(t/\lambda)}}{\lambda}$
D.F.	$F(t) = \begin{cases} 0 & \text{if } t < 0 \\ 1 - e^{-(t/\lambda)} & \text{if } t \ge 0 \end{cases}$
Mean	λ
Variance	λ^2

Erlang Distribution

Parameters	a (scale parameter), $a > 0$ k (number of phases), k is a positive integer
Support set	$\mathbb{R}_{\ge 0}$
p.d.f.	$f(t) = \dfrac{t^{k-1}e^{-(t/a)}}{(k-1)!\,a^k}$
D.F.	$F(t) = \begin{cases} 0 & \text{if } t < 0 \\ 1 - e^{-t/a}\left(\displaystyle\sum_{j=0}^{k-1} \frac{(t/a)^j}{j!}\right) & \text{if } t \ge 0 \end{cases}$
Mean	ak
Variance	a^2k

Gamma Distribution

Parameters	a (scale parameter), $a > 0$ b (shape parameter), $b > 0$
Support set	$\mathbb{R}_{\geq 0}$
p.d.f.	$f(t) = \dfrac{(t/a)^{b-1}e^{-(t/a)}}{a\Gamma(b)}$
Mean	ab
Variance	a^2b

where Γ is the *gamma function* defined by

$$\Gamma(b) = \int_0^\infty e^{-x}x^{b-1}dx.$$

Beta Distribution

The beta distribution is defined in the interval $[0, 1]$. Nonetheless, it can be defined on any other support set by appropriately scaling t.

Parameters	a, b (shape parameters), $a, b > 0$
Support set	$[0, 1]$
p.d.f.	$f(t) = \dfrac{t^{a-1}(1-t)^{b-1}}{\beta(a,b)}$
Mean	$\dfrac{a}{a+b}$
Variance	$\dfrac{ab}{(a+b)^2\,(a+b+1)}$

where β is the *beta function* defined by

$$\beta(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Weibull Distribution

Parameters	a (scale parameter), $a > 0$ b (shape parameter), $b > 0$
Support set	$\mathbb{R}_{\geq 0}$
p.d.f.	$f(t) = \dfrac{bt^{b-1}}{a^b} e^{-(t/a)^b}$
D.F.	$F(t) = \begin{cases} 0 & \text{if } t < 0 \\ 1 - e^{-(t/a)^b} & \text{if } t \geq 0 \end{cases}$
Mean	$\dfrac{a}{b}\Gamma(1/b)$
Variance	$\left(\dfrac{a}{b}\right)^2 \left(2b\Gamma(2/b) - \Gamma(1/b)^2\right)$

Appendix E

Proofs from Chapter 9

E.1 $\sim_{\&}$ is Transitive

Let π and π' be two partitions of the same set. *π is a refinement of π'* (or *π' refines to π*) if for every $S \in \pi$ there is an $S' \in \pi'$ such that $S \subseteq S'$. Thus, π' contains union of sets from π. The sets of all partitions of a given set together with the refinement relation form a lattice.

Let R be a symbolic bisimulation and let $\langle s_1, s_2, SR\rangle \in R$. Then, by definition, $\pi_i(SR) \stackrel{\text{def}}{=} \{C_i \mid \langle C_1, C_2\rangle \in SR\}$ is a partition of some $\mathcal{C}' \subseteq \mathcal{C}$, for i being either 1 or 2. We can extend $\pi_i(SR)$ to a partition of $\mathcal{C}$ by including the atom sets $\{x\}$ whenever $x \notin \mathcal{C}'$. Indeed, this is the finest partition containing $\pi_i(SR)$. Let us denote it by $\pi_i^\star(SR)$. For SR_1 and SR_2 coming from some tuples in some symbolic bisimulations R_1 and R_2, we define $\mathit{fpi}(SR_1, SR_2)$ to be the finest partition which refines to both $\pi_2^\star(SR_1)$ and $\pi_1^\star(SR_2)$. We call $\mathit{fpi}(SR_1, SR_2)$ the *finest partition induced by SR_1 and SR_2*.

The fact that $\sim_{\&}$ is transitive follows immediately from the following lemma.

Lemma E.1. *Let R_1 and R_2 be two symbolic bisimulations, then the relation*

$$
\begin{aligned}
R \;=\; \Big\{ \langle s_1, s_3, SR\rangle \mid & \langle s_1, s_2, SR_1\rangle \in R_1 \wedge \langle s_2, s_3, SR_2\rangle \in R_2 \\
& \wedge \big(\langle C_1, C_3\rangle \in SR \iff \\
& \quad \big(\big(\exists C_2 \in \mathit{fpi}(SR_1, SR_2). \\
& \qquad C_1 = \bigcup\{C_1^\star \mid \langle C_1^\star, C_2^\star\rangle \in SR_1 \wedge \varnothing \neq C_2^\star \subseteq C_2\} \\
& \qquad \wedge C_3 = \bigcup\{C_3^\star \mid \langle C_2^\star, C_3^\star\rangle \in SR_2 \wedge \varnothing \neq C_2^\star \subseteq C_2\} \big) \\
& \quad \vee \big(\langle C_1, \varnothing\rangle \in SR_1 \wedge C_3 = \varnothing \big) \\
& \quad \vee \big(\langle \varnothing, C_3\rangle \in SR_2 \wedge C_1 = \varnothing \big) \big) \big) \Big\}
\end{aligned}
$$

is a symbolic bisimulation.

Proof. Let $\langle s_1, s_3, SR\rangle \in R$. Then there are $\langle s_1, s_2, SR_1\rangle \in R_1$ and $\langle s_2, s_3, SR_2\rangle \in R_2$ satisfying the above conditions. We prove that $\langle s_1, s_3, SR\rangle$ satisfies all the conditions in Definition 9.16.

1. Take $\langle C_1, C_3\rangle, \langle C_1', C_3'\rangle \in SR$. Assume $(C_1 \cap C_1') \cup (C_3 \cap C_3') \neq \varnothing$. We prove the case in which $(C_1 \cap C_1') \neq \varnothing$. The other case follows similarly.

   Suppose that $C_3 \neq \varnothing \neq C_3'$. Then, by definition, there are $C_2, C_2' \in \mathit{fpi}(SR_1, SR_2)$ such that,

   $$\begin{aligned}
   C_1 &= \bigcup\{C_1^\star \mid \langle C_1^\star, C_2^\star\rangle \in SR_1 \wedge \varnothing \neq C_2^\star \subseteq C_2\}\,;\\
   C_3 &= \bigcup\{C_3^\star \mid \langle C_2^\star, C_3^\star\rangle \in SR_2 \wedge \varnothing \neq C_2^\star \subseteq C_2\}\,;\\
   C_1' &= \bigcup\{C_1^\bullet \mid \langle C_1^\bullet, C_2^\bullet\rangle \in SR_1 \wedge \varnothing \neq C_2^\bullet \subseteq C_2'\}\,; \text{ and}\\
   C_3' &= \bigcup\{C_3^\bullet \mid \langle C_2^\bullet, C_3^\bullet\rangle \in SR_2 \wedge \varnothing \neq C_2^\bullet \subseteq C_2'\}\,.
   \end{aligned}$$

   Therefore, there are $\langle C_1^\star, C_2^\star\rangle, \langle C_1^\bullet, C_2^\bullet\rangle \in SR_1$ such that $C_1^\star \cap C_1^\bullet \neq \varnothing$, $\varnothing \neq C_2^\star \subseteq C_2$, and $\varnothing \neq C_2^\bullet \subseteq C_2'$. Since R_1 is a symbolic bisimulation and $(C_1^\star \cap C_1^\bullet) \cup (C_2^\star \cap C_2^\bullet) \neq \varnothing$, then $\langle C_1^\star, C_2^\star\rangle = \langle C_1^\bullet, C_2^\bullet\rangle$. But $\mathit{fpi}(SR_1, SR_2)$ is a partition and $C_2^\star = C_2^\bullet \subseteq C_2 \cap C_2'$, therefore $C_2 = C_2'$. As a consequence $C_1 = C_1'$, and $C_3 = C_3'$.

   If instead $C_3 \neq \varnothing$ and $C_3' = \varnothing$, either the previous case applies —which contradicts the fact that $C_3 \neq \varnothing$ and $C_3' = \varnothing$— or C_1 and C_3 are as before and $\langle C_1', \varnothing\rangle \in SR_1$. So, there is a $\langle C_1^\star, C_2^\star\rangle \in SR_1$ such that $C_1^\star \cap C_1' \neq \varnothing$. Since R_1 is a symbolic bisimulation $\langle C_1^\star, C_2^\star\rangle = \langle C_1', \varnothing\rangle$ which is simply impossible since $C_2^\star \neq \varnothing$.

   Symmetrically, we can prove the case in which $C_3 = \varnothing$ and $C_3' \neq \varnothing$.

   If $C_3 = C_3' = \varnothing$, any of the previous deductions also applies or both $\langle C_1, \varnothing\rangle, \langle C_1', \varnothing\rangle \in SR_1$, in which case $C_1 = C_1'$, since R_1 is a bisimulation.

2. We proceed by proving the double inclusion only for the case $i = 1$ (i.e. the projection on the left component). The other case follows similarly.

   Case ($\subseteq$). Assume $x \in \bigcup\{C_1 \mid \langle C_1, C_3\rangle \in SR\}$. Then, there exists $\langle C_1^\star, C_2^\star\rangle \in SR_1$ such that $x \in C_1^\star$. Since R_1 is a symbolic bisimulation, then $x \in \mathit{rel}(s_1)$.

   Case ($\supseteq$). Assume $x \in \mathit{rel}(s_1)$. Because R_1 is a symbolic bisimulation, there exists $\langle C_1^\bullet, C_2^\bullet\rangle \in SR_1$ such that $x \in C_1^\bullet$.

   If $C_2^\bullet = \varnothing$ then $\langle C_1^\bullet, \varnothing\rangle \in SR$ and $x \in \bigcup\{C_1 \mid \langle C_1, C_3\rangle \in SR\}$.

   Otherwise, there is a $C_2 \in \mathit{fpi}(SR_1, SR_2)$ such that $C_2^\bullet \subseteq C_2$. By definition,

   $$\begin{aligned}
   C_1 &= \bigcup\{C_1^\star \mid \langle C_1^\star, C_2^\star\rangle \in SR_1 \wedge \varnothing \neq C_2^\star \subseteq C_2\}\,; \text{ and}\\
   C_3 &= \bigcup\{C_3^\star \mid \langle C_2^\star, C_3^\star\rangle \in SR_2 \wedge \varnothing \neq C_2^\star \subseteq C_2\}
   \end{aligned}$$

   and $\langle C_1, C_3\rangle \in SR$. Clearly, $C_1^\bullet \subseteq C_1$. As a consequence $x \in \bigcup\{C_1 \mid \langle C_1, C_3\rangle \in SR\}$.

3. Suppose $\langle C_1, C_3\rangle \in SR$ and $C_i \cap \kappa(s_i) \neq \varnothing$ for some $i = 1, 3$. Take $i = 1$. The proof follows similarly if $i = 3$.

(a) If $C_3 = \varnothing$, it could be the case that $\langle C_1, \varnothing\rangle \in SR_1$, from which immediately $C_1 \subseteq \kappa(s_1)$ (and obviously, $C_3 = \varnothing \subseteq \kappa(s_3)$). Otherwise, by definition, there is a $C_2 \in \mathit{fpi}(SR_1, SR_2)$, such that

$$\begin{array}{l} C_1 = \bigcup\{C_1^\star \mid \langle C_1^\star, C_2^\star\rangle \in SR_1 \wedge \varnothing \neq C_2^\star \subseteq C_2\}\,; \text{ and} \\ C_3 = \bigcup\{C_3^\star \mid \langle C_2^\star, C_3^\star\rangle \in SR_2 \wedge \varnothing \neq C_2^\star \subseteq C_2\}. \end{array} \tag{E.1}$$

Hence, there is a $\langle C_1^\star, C_2^\star\rangle \in SR_1$ such that $C_1^\star \cap \kappa(s_i) \neq \varnothing$. Because R_1 is a symbolic bisimulation, $C_1^\star \subseteq \kappa(s_1)$ and $C_2^\star \subseteq \kappa(s_2)$. Since R_1 and R_2 are symbolic bisimulations, it suffices to prove that $C_2 \subseteq \kappa(s_2)$.

We know that a partition π induces an equivalence relation $\rho \stackrel{\text{def}}{=} \{\langle a, b\rangle \mid a, b \in S \in \pi\}$. Call ρ_1 and ρ_2 the equivalences induced respectively by $\pi_2^\star(SR_1)$ and $\pi_1^\star(SR_2)$. $\mathit{fpi}(SR_1, SR_2)$ also induces an equivalence relation on $\mathcal{C}$. Moreover, it is the smallest relation containing ρ_1 and ρ_2. So it must be $(\rho_1 \cup \rho_2)^*$.

Take any $y \in C_2$. Then $\langle x, y\rangle \in (\rho_1 \cup \rho_2)^*$ for some $x \in C_2^\star \subseteq \kappa(s_2)$. Thus, there is an $n \geq 1$ and $z_1, z_2, \ldots, z_n$ such that $x = z_1\ \rho_{i_1}\ z_2 \cdots z_{n-1}\ \rho_{i_{n-1}}\ z_n = y$ with each ρ_{i_j} being either ρ_1 or ρ_2. By induction on n, we show that $y \in \kappa(s_2)$. If $n = 1$ then it is immediate. Suppose that $x = z_1\ \rho_{i_1}\ z_2 \cdots z_k\ \rho_{i_k}\ z_{k+1} = y$. By induction, $z_k \in \kappa(s_2)$. Then, there is either $\langle C_1^\bullet, C_2^\bullet\rangle \in SR_1$ or $\langle C_2^\bullet, C_3^\bullet\rangle \in SR_2$ such that $z_k, y \in C_2^\bullet$ depending whether ρ_{i_k} is ρ_1 or ρ_2. But since $C_2^\bullet \cap \kappa(s_2) \neq \varnothing$ and R_1 and R_2 are symbolic bisimulations, $y \in C_2^\bullet \subseteq \kappa(s_2)$.

(b) By the previous case, we already know that $C_1 \subseteq \kappa(s_1)$ and $C_3 \subseteq \kappa(s_3)$.

First we prove that $C_3 \neq \varnothing$. By contradiction, assume $C_3 = \varnothing$. So, it could be that $\langle C_1, \varnothing\rangle \in SR_1$. Then $\prod_{x\in C_1} F_x(t) = \prod_{y\in\varnothing} F_y(t) = 1$ for all $t \in \mathbb{R}$, which is not possible since $\lim_{t\to-\infty} \prod_{x\in C_1} F_x(t) = 0$ as $\prod_{x\in C_1} F_x(t)$ is a distribution function. So, suppose there is a $C_2 \in \mathit{fpi}(SR_1, SR_2)$. Then, there must exist a $\langle C_2^\star, C_3^\star\rangle \in SR_2$ such that $\varnothing \neq C_2^\star \subseteq C_2 \subseteq \kappa(s_2)$. Proceeding as before, we conclude that $\varnothing \neq C_3^\star \subseteq C_3$.

Therefore, there is a $C_2 \in \mathit{fpi}(SR_1, SR_2)$, such that C_1 and C_3 are as in (E.1).

We also need to show that

$$\begin{array}{rcl} C_2 & = & \bigcup\{C_2^\star \mid \langle C_1^\star, C_2^\star\rangle \in SR_1 \wedge \varnothing \neq C_2^\star \subseteq C_2\} \\ & = & \bigcup\{C_2^\bullet \mid \langle C_2^\bullet, C_3^\bullet\rangle \in SR_2 \wedge \varnothing \neq C_2^\bullet \subseteq C_2\}. \end{array} \tag{E.2}$$

Clearly C_2 includes both sets, and in fact, it is the union of both of them. Suppose $x \in \bigcup\{C_2^\star \mid \langle C_1^\star, C_2^\star\rangle \in SR_1 \wedge C_2^\star \subseteq C_2\} \subseteq C_2$. Then $x \in \mathit{rel}(s_2)$. So, there must exist $\langle C_2^\bullet, C_3^\bullet\rangle \in SR_2$ such that $x \in C_2^\bullet$. By definition of C_2, $C_2^\bullet \subseteq C_2$, so $x \in \bigcup\{C_2^\bullet \mid \langle C_2^\bullet, C_3^\bullet\rangle \in SR_2 \wedge C_2^\bullet \subseteq C_2\}$. By symmetric reasoning, (E.2) follows.

Now, we can calculate

$$\begin{aligned}
\prod_{x\in C_1} F_x(t) &= \prod_{\substack{\langle C_1^\star, C_2^\star\rangle \in SR_1,\\ \varnothing \neq C_2^\star \subseteq C_2,\, x\in C_1^\star}} F_x(t) = \prod_{\substack{\langle C_1^\star, C_2^\star\rangle \in SR_1,\\ \varnothing \neq C_2^\star \subseteq C_2}} \left(\prod_{x\in C_1^\star} F_x(t) \right) = \\
&= \prod_{\substack{\langle C_1^\star, C_2^\star\rangle \in SR_1,\\ \varnothing \neq C_2^\star \subseteq C_2}} \left(\prod_{y\in C_2^\star} F_y(t) \right) = \prod_{\substack{\langle C_1^\star, C_2^\star\rangle \in SR_1,\\ \varnothing \neq C_2^\star \subseteq C_2,\, y\in C_2^\star}} F_y(t) = \\
&= \prod_{y\in C_2} F_y(t) = \\
&= \prod_{\substack{\langle C_2^\star, C_3^\star\rangle \in SR_2,\\ \varnothing \neq C_2^\star \subseteq C_2,\, y\subseteq C_2^\star}} F_y(t) = \prod_{\substack{\langle C_2^\star, C_3^\star\rangle \in SR_2,\\ \varnothing \neq C_2^\star \subseteq C_2}} \left(\prod_{y\in C_2^\star} F_y(t) \right) = \\
&= \prod_{\substack{\langle C_2^\star, C_3^\star\rangle \in SR_2,\\ \varnothing \neq C_2^\star \subseteq C_2}} \left(\prod_{z\in C_3^\star} F_z(t) \right) = \prod_{\substack{\langle C_2^\star, C_3^\star\rangle \in SR_2\\ \varnothing \neq C_2^\star \subseteq C_2,\, z\in C_3^\star}} F_z(t) = \\
&= \prod_{z\in C_3} F_z(t)
\end{aligned}$$

4. Suppose $s_1 \xrightarrow{a, C_1^\bullet} s_1'$. Then

   (a_1) $s_2 \xrightarrow{a, C_2^\bullet} s_2'$;

   (b_1) $\langle C_1^\star, C_2^\star\rangle \in SR_1$ and $(C_1^\star \cap C_1^\bullet) \cup (C_2^\star \cap C_2^\bullet) \neq \varnothing$ implies $C_1^\star \subseteq C_1^\bullet$ and $C_2^\star \subseteq C_2^\bullet$; and

   (c_1) there exists SR_1' such that $\langle s_1', s_2', SR_1'\rangle \in R_1$ and

$$\begin{aligned}
&\{\langle C_1^\star, C_2^\star\rangle \in SR_1 \mid (C_1^\star \cap FV(s_1')) \cup (C_2^\star \cap FV(s_2')) \neq \varnothing\} - \big(\wp(C_1^\bullet) \times \wp(C_2^\bullet)\big) = \\
&\{\langle C_1^\star, C_2^\star\rangle \in SR_1' \mid (C_1^\star \cap FV(s_1')) \cup (C_2^\star \cap FV(s_2')) \neq \varnothing\} - \big(\wp(C_1^\bullet) \times \wp(C_2^\bullet)\big).
\end{aligned}$$

   As a consequence of (a_1), we have the following statements.

   (a_2) $s_3 \xrightarrow{a, C_3^\bullet} s_3'$;

   (b_2) $\langle C_2^\star, C_3^\star\rangle \in SR_2$ and $(C_2^\star \cap C_2^\bullet) \cup (C_3^\star \cap C_3^\bullet) \neq \varnothing$ implies $C_2^\star \subseteq C_2^\bullet$ and $C_3^\star \subseteq C_3^\bullet$; and

(c_2) there exists SR'_2 such that $\langle s'_2, s'_3, SR'_2\rangle \in R_2$ and

$$\{\langle C^\star_2, C^\star_3\rangle \in SR_2 \mid (C^\star_2 \cap FV(s'_2)) \cup (C^\star_3 \cap FV(s'_3)) \neq \varnothing\} - \big(\wp(C^\bullet_2) \times \wp(C^\bullet_3)\big) =$$
$$\{\langle C^\star_2, C^\star_3\rangle \in SR'_2 \mid (C^\star_2 \cap FV(s'_2)) \cup (C^\star_3 \cap FV(s'_3)) \neq \varnothing\} - \big(\wp(C^\bullet_2) \times \wp(C^\bullet_3)\big).$$

We proceed by proving items (a), (b), and (c).

(a) $s_3 \xrightarrow{a, C^\bullet_3} s'_3$ is immediate by (a_2).

(b) Suppose $\langle C_1, C_3\rangle \in SR$ and $(C_1 \cap C^\bullet_1) \cup (C_3 \cap C^\bullet_3) \neq \varnothing$. If $C_3 = \varnothing$, it could be the case that $\langle C_1, \varnothing\rangle \in SR_2$. Because of ($b_1$), $C_1 \subseteq C^\bullet_1$, and trivially $C_3 = \varnothing \subseteq C^\bullet_3$. It proceeds analogously if $C_1 = \varnothing$.

Otherwise, there is a $C_2 \in fpi(SR_1, SR_2)$, such that C_1 and C_3 are as in (E.1). Moreover, C_2 is as in (E.2). Because of (b_1) and (b_2), it is enough to prove that $C_2 \subseteq C^\bullet_2$.

We proceed inductively as we did in item 3a. So consider ρ_1 and ρ_2 as before and some $C^\star_2 \subseteq C^\bullet_2$, $C^\star_2 \neq \varnothing$, as in ($b_1$) or ($b_2$), which certainly exists since $(C_1 \cap C^\bullet_1) \cup (C_3 \cap C^\bullet_3) \neq \varnothing$. Take any $y \in C_2$. Then $\langle x, y\rangle \in (\rho_1 \cup \rho_2)^*$ for some $x \in C^\star_2 \subseteq C^\bullet_2$. Thus, there is an $n \geq 1$ and $z_1, z_2, \ldots, z_n$ such that $x = z_1\ \rho_{i_1}\ z_2 \cdots z_{n-1}\ \rho_{i_{n-1}}\ z_n = y$ with each ρ_{i_j} being either ρ_1 or ρ_2. If $n = 1$, then it is immediate. Suppose that $x = z_1\ \rho_{i_1}\ z_2 \cdots z_k\ \rho_{i_k}\ z_{k+1} = y$. By induction, $z_k \in C^\bullet_2$. Then, there is either $\langle C'^\star_1, C'^\star_2\rangle \in SR_1$ or $\langle C'^\star_2, C'^\star_3\rangle \in SR_2$ such that $z_k, y \in C'^\star_2$ depending whether ρ_{i_k} is ρ_1 or ρ_2. But since $C'^\star_2 \cap C^\bullet_2 \neq \varnothing$, either because ($b_1$) or ($b_2$), $y \in C'^\star_2 \subseteq C^\bullet_2$.

(c) Define SR' such that

$$\begin{aligned}
\langle C_1, C_3\rangle \in SR' \iff{}& \\
&\big(\ (\ \exists C_2 \in fpi(SR'_1, SR'_2).\\
&\quad C_1 = \bigcup\{C^\star_1 \mid \langle C^\star_1, C^\star_2\rangle \in SR'_1 \wedge \varnothing \neq C^\star_2 \subseteq C_2\}\\
&\quad \wedge C_3 = \bigcup\{C^\star_3 \mid \langle C^\star_2, C^\star_3\rangle \in SR'_2 \wedge \varnothing \neq C^\star_2 \subseteq C_2\}\ )\\
&\vee (\ \langle C_1, \varnothing\rangle \in SR'_1 \wedge C_3 = \varnothing\ )\\
&\vee (\ \langle \varnothing, C_3\rangle \in SR'_2 \wedge C_1 = \varnothing\ )\ \big).
\end{aligned} \tag{E.3}$$

Clearly, $\langle s'_1, s'_3, SR'\rangle \in R$.

It remains to prove that SR' is forward compatible with SR, i.e.,

$$\{\langle C_1, C_3\rangle \in SR \mid (C_1 \cap FV(s'_1)) \cup (C_3 \cap FV(s'_3)) \neq \varnothing\} - \big(\wp(C^\bullet_1) \times \wp(C^\bullet_3)\big) =$$
$$\{\langle C_1, C_3\rangle \in SR' \mid (C_1 \cap FV(s'_1)) \cup (C_3 \cap FV(s'_3)) \neq \varnothing\} - \big(\wp(C^\bullet_1) \times \wp(C^\bullet_3)\big).$$

We proceed by showing the double inclusion.

Case ($\subseteq$). In this case we have that $\langle C_1, C_3\rangle \in SR$, $(C_1 \cap FV(s_1')) \cup (C_3 \cap FV(s_3')) \neq \varnothing$, and $C_1 - C_1^\bullet \neq \varnothing$ or $C_3 - C_3^\bullet \neq \varnothing$.
If $C_3 = \varnothing$, it could be the case that $\langle C_1, \varnothing\rangle \in SR_1$. Then $\langle C_1, \varnothing\rangle \in SR_1'$, by ($c_1$). Similarly, if $C_1 = \varnothing$, it could be that $\langle \varnothing, C_3\rangle \in SR_2$, and then $\langle \varnothing, C_3\rangle \in SR_2'$ because of (c_2). In either case, we can conclude that

$$\langle C_1, C_3\rangle \in \{\langle C_1, C_3\rangle \in SR' \mid (C_1 \cap FV(s_1')) \cup (C_3 \cap FV(s_3')) \neq \varnothing\} - \big(\wp(C_1^\bullet) \times \wp(C_3^\bullet)\big).$$

Now, suppose that there is a $C_2 \in \mathit{fpi}(SR_1, SR_2)$, such that C_1 and C_3 are as in (E.1). Moreover, C_2 is as in (E.2).
We show that $C_2 \subseteq FV(s_2') - C_2^\bullet$. In fact, since $C_1 - C_1^\bullet \neq \varnothing$ or $C_3 - C_3^\bullet \neq \varnothing$, by 4b, $C_2 \cap C_2^\bullet = C_1 \cap C_1^\bullet = C_3 \cap C_3^\bullet = \varnothing$. So we only have to prove that $C_2 \subseteq FV(s_2')$. W.l.o.g. suppose $C_1 \cap FV(s_1') \neq \varnothing$. Then, there should exist some $\langle C_1^\star, C_2^\star\rangle \in SR_1$ such that $C_1^\star \cap FV(s_1') \neq \varnothing$ and $\varnothing \neq C_2^\star \subseteq C_2$. Because of ($c_1$), $\langle C_1^\star, C_2^\star\rangle \in SR_1'$ and since R_1 is a bisimulation $C_1^\star \subseteq FV(s_1')$ and $C_2^\star \subseteq FV(s_2')$ (this follows from conditions 2 and 3a).
Once again we proceed by induction on the construction of C_2 using ρ_1 and ρ_2. Let $y \in C_2$. Then $\langle x, y\rangle \in (\rho_1 \cup \rho_2)^*$ for some $x \in C_2^\star \subseteq FV(s_2')$. Thus, there is an $n \geq 1$ and $z_1, z_2, \ldots, z_n$ such that $x = z_1\ \rho_{i_1}\ z_2 \cdots z_{n-1}\ \rho_{i_{n-1}}\ z_n = y$ with each ρ_{i_j} being either ρ_1 or ρ_2. If $n = 1$, then it is immediate. Suppose that $x = z_1\ \rho_{i_1}\ z_2 \cdots z_k\ \rho_{i_k}\ z_{k+1} = y$. By induction, $z_k \in FV(s_2')$. Then, there is either $\langle C'^\star_1, C'^\star_2\rangle \in SR_1$ or $\langle C'^\star_2, C'^\star_3\rangle \in SR_2$ such that $z_k, y \in C'^\star_2$ depending whether ρ_{i_k} is ρ_1 or ρ_2. But since $z_k \in C'^\star_2 \cap FV(s_2') \neq \varnothing$, because of ($c_1$) or ($c_2$), $\langle C'^\star_1, C'^\star_2\rangle \in SR_1'$ or $\langle C'^\star_2, C'^\star_3\rangle \in SR_2'$, respectively. As a consequence, since R_1 and R_2 are bisimulations, $y \in C'^\star_2 \subseteq FV(s_2')$. Thus $C_2 \subseteq FV(s_2')$.
Take $C_2^\star \subseteq C_2 \subseteq FV(s_2') - C_2^\bullet$ such that $\langle C_1^\star, C_2^\star\rangle \in SR_1$. Then $\langle C_1^\star, C_2^\star\rangle \in SR_1'$ by (c_1). Similarly, if $\langle C_2^\star, C_3^\star\rangle \in SR_2$, then $\langle C_2^\star, C_3^\star\rangle \in SR_2'$ by (c_2). It should be clear that $C_2 \in \mathit{fpi}(SR_1', SR_2')$, which implies $\langle C_1, C_3\rangle \in SR'$. As a consequence

$$\langle C_1, C_3\rangle \in \{\langle C_1, C_3\rangle \in SR' \mid (C_1 \cap FV(s_1')) \cup (C_3 \cap FV(s_3')) \neq \varnothing\} - \big(\wp(C_1^\bullet) \times \wp(C_3^\bullet)\big).$$

Case ($\supseteq$). We first notice that, to prove 1, 2, and 3, we only needed the definition of SR and the fact that $\langle s_1, s_2, SR_1\rangle \in R_1$ and $\langle s_2, s_3, SR_2\rangle \in R_2$. Because of (E.3), $\langle s_1', s_2', SR_1'\rangle \in R_1$ and $\langle s_2', s_3', SR_2'\rangle \in R_2$, it follows that conditions 1, 2, and 3 also apply to SR'.
Now, take

$$\langle C_1, C_3\rangle \in \{\langle C_1, C_3\rangle \in SR' \mid (C_1 \cap FV(s_1')) \cup (C_3 \cap FV(s_3')) \neq \varnothing\} - \big(\wp(C_1^\bullet) \times \wp(C_3^\bullet)\big).$$

If $C_3 = \varnothing$, it could be the case that $\langle C_1, \varnothing\rangle \in SR'_1$. Then $\langle C_1, \varnothing\rangle \in SR_1$, by ($c_1$). Similarly, if $C_1 = \varnothing$ it could be that $\langle \varnothing, C_3\rangle \in SR'_2$, and then $\langle \varnothing, C_3\rangle \in SR_2$ because of (c_2). In either case, we can conclude that

$$\langle C_1, C_3\rangle \in \{\langle C_1, C_3\rangle \in SR \mid (C_1 \cap FV(s'_1)) \cup (C_3 \cap FV(s'_3)) \neq \varnothing\} - \big(\wp(C_1^\bullet) \times \wp(C_3^\bullet)\big).$$

Otherwise, there is a $C_2 \in fpi(SR'_1, SR'_2)$, such that

$$C_1 = \bigcup\{C_1^\star \mid \langle C_1^\star, C_2^\star\rangle \in SR'_1 \wedge \varnothing \neq C_2^\star \subseteq C_2\}\,; \text{ and}$$

$$C_3 = \bigcup\{C_3^\star \mid \langle C_2^\star, C_3^\star\rangle \in SR'_2 \wedge \varnothing \neq C_2^\star \subseteq C_2\}.$$

Moreover, C_2 can be proven to satisfy similar conditions to (E.2), namely,

$$\begin{aligned} C_2 &= \bigcup\{C_2^\star \mid \langle C_1^\star, C_2^\star\rangle \in SR'_1 \wedge \varnothing \neq C_2^\star \subseteq C_2\} \\ &= \bigcup\{C_2^\star \mid \langle C_2^\star, C_3^\star\rangle \in SR'_2 \wedge \varnothing \neq C_2^\star \subseteq C_2\}. \end{aligned}$$

Since $(C_1 \cap FV(s'_1)) \cup (C_3 \cap FV(s'_3)) \neq \varnothing$, using conditions 2 and 3a, $C_1 \subseteq FV(s'_1)$ and $C_3 \subseteq FV(s'_3)$, as well as $C_2 \subseteq FV(s'_2)$, in this last case considering each of the subsets that form C_2.

We show that moreover $C_2 \subseteq FV(s'_2) - C_2^\bullet$. Notice that there should be an $x \in (C_1 \cap FV(s'_1)) \cup (C_3 \cap FV(s'_3))$ such that $x \notin C_1^\bullet \cup C_3^\bullet$. W.l.o.g. suppose $x \in C_1 \cap FV(s'_1)$. Then, there is a $\langle C_1^\star, C_2^\star\rangle \in SR'_1$ such that $x \in C_1^\star \cap FV(s'_1)$ and $C_2^\star \neq \varnothing$. By ($c_1$), $\langle C_1^\star, C_2^\star\rangle \in SR_1$, and because of ($b_1$), $C_1^\star \cap C_1^\bullet = C_2^\star \cap C_2^\bullet = \varnothing$. Again we proceed by induction on the construction of C_2 using ρ_1 and ρ_2 and prove that for all $y \in C_2$, $y \notin C_2^\bullet$. Let $y \in C_2$. Then $\langle x, y\rangle \in (\rho_1 \cup \rho_2)^*$ for some $x \in C_2^\star$, $x \notin C_2^\bullet$. Thus, there is an $n \geq 1$ and $z_1, z_2, \ldots, z_n$ such that $x = z_1\ \rho_{i_1}\ z_2 \cdots z_{n-1}\ \rho_{i_{n-1}}\ z_n = y$ with each ρ_{i_j} being either ρ_1 or ρ_2. If $n = 1$, then it is immediate. Suppose that $x = z_1\ \rho_{i_1}\ z_2 \cdots z_k\ \rho_{i_k}\ z_{k+1} = y$. By induction, $z_k \notin C_2^\bullet$. Then, there is either $\langle C'^\star_1, C'^\star_2\rangle \in SR'_1$ or $\langle C'^\star_2, C'^\star_3\rangle \in SR'_2$ such that $z_k, y \in C'^\star_2$ depending whether ρ_{i_k} is ρ_1 or ρ_2. But since $z_k \in C'^\star_2 \subseteq FV(s'_2)$ and $z_k \notin C_2^\bullet$, because of (c_1) or (c_2), $\langle C'^\star_1, C'^\star_2\rangle \in SR_1$ or $\langle C'^\star_2, C'^\star_3\rangle \in SR_2$, respectively. Moreover, because of (b_1) or (b_2), $C'^\star_2 \cap C_2^\bullet = \varnothing$. As a consequence, $y \notin C_2^\bullet$, which proves that $C_2 \subseteq FV(s'_2) - C_2^\bullet$.

Take $C_2^\star \subseteq C_2 \subseteq FV(s'_2) - C_2^\bullet$ such that $\langle C_1^\star, C_2^\star\rangle \in SR'_1$. Then $\langle C_1^\star, C_2^\star\rangle \in SR_1$, because of ($c_1$). Similarly, if $\langle C_2^\star, C_3^\star\rangle \in SR'_2$, then $\langle C_2^\star, C_3^\star\rangle \in SR_2$, by ($c_2$). It should be clear that $C_2 \in fpi(SR_1, SR_2)$, which implies $\langle C_1, C_3\rangle \in SR$. As a consequence

$$\langle C_1, C_3\rangle \in \{\langle C_1, C_3\rangle \in SR \mid (C_1 \cap FV(s'_1)) \cup (C_3 \cap FV(s'_3)) \neq \varnothing\} - \big(\wp(C_1^\bullet) \times \wp(C_3^\bullet)\big).$$

5. This case is symmetric to the previous one. *Q.E.D.*

E.2 Proof of Theorem 9.20

Lemma E.2. *Let $R_{\&}$ be a symbolic bisimulation on a given stochastic automaton SA. The relation*

$$
\begin{aligned}
R_o \stackrel{\text{def}}{=} \Big(& \big\{ \langle (s_1, v_1), (s_2, v_2) \rangle \mid \langle s_1, s_2, SR \rangle \in R_{\&} \\
& \wedge \forall \langle C_1, C_2 \rangle \in SR - (\wp\mathcal{O}(\kappa(s_1)) \times \wp\mathcal{O}(\kappa(s_2))). \\
& \quad \max\{v_1(x) \mid x \in C_1\} = \max\{v_2(y) \mid y \in C_2\} \\
& \quad \vee \big(\max\{v_1(x) \mid x \in C_1\} \leq 0 \wedge \max\{v_2(y) \mid y \in C_2\} \leq 0 \big) \big\} \qquad \text{(E.4)} \\
\cup \big\{ & \langle [s_1, v_1], [s_2, v_2] \rangle \mid \langle s_1, s_2, SR \rangle \in R_{\&} \\
& \wedge \forall \langle C_1, C_2 \rangle \in SR. \ \max\{v_1(x) \mid x \in C_1\} = \max\{v_2(y) \mid y \in C_2\} \\
& \quad \vee \big(C_1 \cap \kappa(s_1) = \varnothing = C_2 \cap \kappa(s_2) \\
& \qquad \wedge \max\{v_1(x) \mid x \in C_1\} \leq 0 \wedge \max\{v_2(y) \mid y \in C_2\} \leq 0 \big) \big\} \Big)^{\mathsf{s}} \qquad \text{(E.5)}
\end{aligned}
$$

is a probabilistic bisimulation up to $\sim_p$ in $PTS_o(SA)$. (In general, we take $\max \varnothing = 0$*.)*

Proof. By definition, R_o is symmetric. Thus, we directly proceed to prove the transfer properties in Definition 8.10. We only prove the straight cases. The symmetric ones follow in the similar way.

1. Let $\langle (s_1, v_1), (s_2, v_2) \rangle \in R_o$. Then $\langle s_1, s_2, SR \rangle \in R_{\&}$ satisfies the requirements in (E.4).
   Let $\overrightarrow{\kappa(s_1)} = (x_1, \ldots, x_n)$ and $\overrightarrow{\kappa(s_2)} = (y_1, \ldots, y_m)$. By Definition 9.7 (rule **Prob**),

   $$
   \begin{aligned}
   T(s_1, v_1) &= \mathcal{D}^{s_1}_{v_1}(\mathcal{R}(F_{x_1}, \ldots, F_{x_n})) = (\Omega_1, \mathcal{F}_1, P_1) \qquad \text{and} \\
   T(s_2, v_2) &= \mathcal{D}^{s_2}_{v_2}(\mathcal{R}(F_{y_1}, \ldots, F_{y_m})) = (\Omega_2, \mathcal{F}_2, P_2).
   \end{aligned}
   $$

   Let N be the number of pairs $\langle C_1, C_2 \rangle \in SR$ such that $C_1 \in \kappa(s_1)$ (or equivalently $C_2 \in \kappa(s_2)$). Let us take $\langle C_1^1, C_2^1 \rangle$, $\ldots$, $\langle C_1^N, C_2^N \rangle$ to be those tuples. For $i = 1, 2$, define the function $\zeta_i : \Omega_i \to \mathbb{R}^N$ by

   $$
   \zeta_i([s_i, v_i']) \stackrel{\text{def}}{=} (\max\{v_i'(x) \mid x \in C_i^1\}, \ldots, \max\{v_i'(x) \mid x \in C_i^N\}).
   $$

   Notice that, for $i = 1$ or $i = 2$, if $[s_i, v_i'] \in \Omega_i$, then, all elements in $(\zeta_1)^{-1}(\zeta_i([s_i, v_i']))$ and $(\zeta_2)^{-1}(\zeta_i([s_i, v_i']))$ are related by $(\sim_p \cup R_o)^*$. In other words, there exists a set

$S_{eq} \in [\mathcal{S} \times \mathbf{V}]/(\sim_p \cup R_o)^*$ such that

$$(\zeta_1)^{-1}\left(\zeta_i([s_i, v_i'])\right) \cup (\zeta_2)^{-1}\left(\zeta_i([s_i, v_i'])\right) \subseteq S_{eq} \in [\mathcal{S} \times \mathbf{V}]/(\sim_p \cup R_o)^* \tag{E.6}$$

For $k = 1, \ldots, N$, $i = 1, 2$, define ζ_i^k to be the k-th projection of ζ_i, i.e., $\zeta_i^k \stackrel{\text{def}}{=} \pi_k \circ \zeta_i$. Notice that ζ_i^k is a random variable. More precisely it is the random variable $\max C_i^k$. As a consequence the distribution function of ζ_i^k is the function $F_{\zeta_i^k}$ defined by

$$F_{\zeta_i^k}(t) \stackrel{\text{def}}{=} \prod_{x \in C_i^k} F_x(t)$$

for all $t \in \mathbb{R}$.

Since $R_{\&}$ is a symbolic bisimulation, it follows that

$$F_{\zeta_1^k}(t) = \prod_{x \in C_1^k} F_x(t) = \prod_{y \in C_2^k} F_y(t) = F_{\zeta_2^k}(t) \tag{E.7}$$

by item 3b in Definition 9.16.

From observations (E.6) and (E.7) it is immediate that, if $S \subseteq [\mathcal{S} \times \mathbf{V}]/(\sim_p \cup R_o)^*$ and $\mu((s_1, v_1), \bigcup S) > 0$, then

$$\begin{aligned}\mu((s_1, v_1), \bigcup S) &= P_1(\Omega_1 \cap \bigcup S) = P_1'(\zeta_1(\Omega_1 \cap \bigcup S)) \\ &= P_2'(\zeta_2(\Omega_2 \cap \bigcup S)) = P_2(\Omega_2 \cap \bigcup S) = \mu((s_2, v_2), \bigcup S)\end{aligned}$$

where each P_i', $i = 1, 2$, is the probability function in the respective probability space $\mathcal{R}(F_{\zeta_i^1}, \ldots, F_{\zeta_i^N})$, which actually is the same probability space because of (E.7).

2. Let $\langle [s_1, v_1], [s_2, v_2] \rangle \in R_o$ and suppose that $[s_1, v_1] \xrightarrow{a(d)} (s_1', v_1 - d)$. Then, by rule **Open** in Definition 9.7,

$$s_1 \xrightarrow{a, C_1^\star} s_1', \quad d \in \mathbb{R}_{\geq 0}, \quad \text{and} \quad \forall x \in C_1^\star.\ (v_1 - d)(x) \leq 0.$$

Since $\langle s_1, s_2, SR \rangle \in R_{\&}$,

$$s_2 \xrightarrow{a, C_2^\star} s_2' \tag{E.8}$$

by item 4a in Definition 9.16. Moreover, because of item 4b, there is a set of pairs $\langle C_1^1, C_2^1 \rangle, \ldots, \langle C_1^N, C_2^N \rangle \in SR$, for some $N \geq 0$, such that $C_i^\star = \bigcup_k C_i^k$, with $i = 1, 2$. Because of conditions in (E.5), we can ensure that

$$\forall y \in C_2^\star.\ (v_2 - d)(y) \leq 0. \tag{E.9}$$

From (E.8), (E.9), and rule **Open**, it follows that $[s_2, v_2] \xrightarrow{a(d)} (s_2', v_2 - d)$.

It remains to prove that $\langle (s_1', v_1 - d), (s_2', v_2 - d) \rangle \in R_o$ which reduces to prove conditions in (E.4). Because of item 4c in Definition 9.16, we have that $\langle s_1', s_2', SR' \rangle \in R_{\&}$ for some SR' satisfying forward compatibility:

$$\{\langle C_1, C_2\rangle \in SR \mid (C_1 \cap FV(s_1')) \cup (C_2 \cap FV(s_2')) \neq \varnothing\} - \big(\wp(C_1^\star) \times \wp(C_2^\star)\big) = \{\langle C_1, C_2\rangle \in SR' \mid (C_1 \cap FV(s_1')) \cup (C_2 \cap FV(s_2')) \neq \varnothing\} - \big(\wp(C_1^\star) \times \wp(C_2^\star)\big).$$

Notice that this last equality consider exactly all pairs in $SR' - ((\wp(C_1^\star) \times \wp(C_2^\star)) \cup (\wp(\kappa(s_1')) \times \wp(\kappa(s_2'))))$. Thus, if $\langle C_1, C_2\rangle \in SR' - ((\wp(C_1^\star) \times \wp(C_2^\star)) \cup (\wp(\kappa(s_1')) \times \wp(\kappa(s_2'))))$, $\langle C_1, C_2\rangle \in SR$. Hence

$$\max\{v_1(x) \mid x \in C_1\} = \max\{v_2(y) \mid y \in C_2\}$$

$$\vee\ (\ \max\{v_1(x) \mid x \in C_1\} \leq 0 \wedge \max\{v_2(y) \mid y \in C_2\} \leq 0\ )$$

because of (E.5). Now, it is immediate that

$$\max\{(v_1 - d)(x) \mid x \in C_1\} = \max\{(v_2 - d)(y) \mid y \in C_2\}$$

$$\vee\ (\ \max\{(v_1 - d)(x) \mid x \in C_1\} \leq 0 \wedge \max\{(v_2(y) - d) \mid y \in C_2\} \leq 0\ ).$$

If instead $\langle C_1, C_2\rangle \in SR' \cap (\wp(C_1^\star) \times \wp(C_2^\star))$, then

$$\max\{(v_1 - d)(x) \mid x \in C_1\} \leq 0 \wedge \max\{(v_2(y) - d) \mid y \in C_2\} \leq 0.$$

This concludes the proof. *Q.E.D.*

Theorem 9.20. *Let s_1 and s_2 be two locations in a stochastic automaton SA. If $s_1 \sim_{\&} s_2$ then $s_1 \sim_o s_2$.*

Proof. The proof follows from Lemma E.2. It only remains to notice that if SR satisfies conditions (a)–(c) in Definition 9.16, then any $v_1 = v_2$ satisfies conditions in (E.4). *Q.E.D.*

E.3 Proof of Theorem 9.21

Lemma E.3. *Let SA be a stochastic automaton and let $PTS_c(SA)$ and $PTS_o(SA)$ be its closed and open behaviours, respectively. The two following statements are equivalent*

1. $[s, v] \xrightarrow{a(d)} (s', v')$
2. $[s, v] \xrightarrow{a(d)}' (s', v')$ *and for all* $d' \in [0, d)$, $b \in \mathcal{A}$, $[s, v] \xrightarrow{b(d')}\!\!\!\!\!\!/\ \ '$

where $\longrightarrow$ and $\longrightarrow'$ are respectively the transition relations of $PTS_c(SA)$ and $PTS_o(SA)$.

Proof. (*1. $\Rightarrow$ 2.*) By definition of **Closed** and **Open** it is immediate that $[s,v] \xrightarrow{a(d)} (s',v')$ implies $[s,v] \xrightarrow{a(d)}{}' (s',v')$. Moreover, because of **Closed**,

$$\forall d' \in [0,d).\ \forall s \xrightarrow{b,C'}.\ \exists y \in C'.\ (v-d')(y) > 0$$

which, by **Open**, implies that there is no $d' \in [0,d)$ and no $b \in \mathcal{A}$ such that $[s,v] \xrightarrow{b(d')}{}'$. (*2. $\Rightarrow$ 1.*) Assume $[s,v] \xrightarrow{a(d)}{}' (s',v')$. By **Open**,

$$s \xrightarrow{a,C} s' \quad d \in \mathbb{R}_{\geq 0} \quad \forall x \in C.\ (v-d)(x) \leq 0 \tag{E.10}$$

By contradiction, suppose $[s,v] \xrightarrow{a(d)}\!\!\!\!\!/\ $. By (E.10) and **Close**, it must be the case that

$$\neg \Big(\forall d' \in [0,d).\ \forall s \xrightarrow{b,C'}.\ \exists y \in C'.\ (v-d')(y) > 0 \Big)$$

which is equivalent to say

$$\exists d' \in [0,d).\ \exists s \xrightarrow{b,C'}.\ \forall y \in C'.\ (v-d')(y) \leq 0$$

or equivalent to say

$$\exists d' \in [0,d), b \in \mathcal{A}, C' \subseteq \mathcal{C}.\quad s \xrightarrow{b,C'} \quad \text{and} \quad \forall y \in C'.\ (v-d')(y) \leq 0$$

By **Open**, this implies that

$$\exists d' \in [0,d), b \in \mathcal{A}.\ \ [s,v] \xrightarrow{b(d')}{}'$$

which contradicts the hypothesis. *Q.E.D.*

Theorem 9.21. *Let s_1 and s_2 be two locations in a stochastic automaton SA. If $s_1 \sim_o s_2$ then $s_1 \sim_c s_2$.*

Proof. Let R be a probabilistic bisimulation relation on the open behaviour of SA. Using Lemma E.3, it is not difficult to check that the same relation R is also a probabilistic bisimulation on the closed behaviour of SA. *Q.E.D.*

Appendix F

Proofs from Chapter 10

F.1 Proofs of Theorems 10.17 and 10.18

The proof of these theorems is built from several definitions and lemmas. Remember that both Theorems 10.17 and 10.18 state that, under some conditions, two stochastic automata are symbolically bisimilar. We instead will prove something stronger, we will prove that they are *renaming bisimilar*. A renaming bisimulation is like a structural bisimulation that allows clocks to change name while preserving the distribution.

Definition F.1. Let $(\mathcal{S}, \mathcal{A}, \mathcal{C}, \longrightarrow, \kappa)$ be a stochastic automaton. A *renaming bisimulation* is a relation $R \subseteq \mathcal{S} \times \mathcal{S} \times \wp(\mathcal{C} \times \mathcal{C})$ that, whenever $\langle s_1, s_2, SR \rangle \in R$, the following properties hold.

1. SR is a stochastically appropriate bijection.
2. For $i = 1, 2$, $\{x_i \mid \langle x_1, x_2 \rangle \in SR\} \supseteq rel(s_i)$, and moreover, if $\langle x_1, x_2 \rangle \in SR$, then $x_1 \in rel(s_1) \iff x_2 \in rel(s_2)$.
3. If $\langle x_1, x_2 \rangle \in SR$, then $x_1 \in \kappa(s_1) \iff x_2 \in \kappa(s_2)$.
4. $s_1 \xrightarrow{a,C_1} s_1'$, then there are s_2' and C_2 such that
   (a) $s_2 \xrightarrow{a,C_2} s_2'$;
   (b) If $\langle x_1, x_2 \rangle \in SR$, then $x_1 \in C_1 \iff x_2 \in C_2$; and
   (c) there exists SR' such that $\langle s_1', s_2', SR' \rangle \in R$ and
   $$\{\langle x_1, x_2 \rangle \in SR \mid x_1 \in FV(s_1') \vee x_2 \in FV(s_2')\} - (C_1 \times C_2) = \{\langle x_1, x_2 \rangle \in SR' \mid x_1 \in FV(s_1') \vee x_2 \in FV(s_2')\} - (C_1 \times C_2).$$
5. $s_2 \xrightarrow{a,C_2} s_2'$, then there are s_1' and C_1 such that
   (a) $s_1 \xrightarrow{a,C_1} s_1'$;

(b) If $\langle x_1, x_2\rangle \in SR$, then $x_1 \in C_1 \iff x_2 \in C_2$; and

(c) there exists SR' such that $\langle s_1', s_2', SR'\rangle \in R$ and

$$\{\langle x_1, x_2\rangle \in SR \mid x_1 \in FV(s_1') \vee x_2 \in FV(s_2')\} - (C_1 \times C_2) = \{\langle x_1, x_2\rangle \in SR' \mid x_1 \in FV(s_1') \vee x_2 \in FV(s_2')\} - (C_1 \times C_2).$$

□

The proof of the following lemma is routine.

Lemma F.2. *Let R_r be a renaming bisimulation. The relation*

$$R_{\&} \stackrel{\text{def}}{=} \big\{\langle s_1, s_2, SR'\rangle \mid \langle s_1, s_2, SR\rangle \in R_r \wedge SR' = \{\langle\{x_1\},\{x_2\}\rangle \mid \langle x_1, x_2\rangle \in SR \wedge x_1 \in rel(s_1)\}\big\}$$

is a symbolic bisimulation. Moreover, if $\langle x, x\rangle \in SR$ for all $x \in FV(s_1) \cup FV(s_2)$, then $s_1 \sim_{\&} s_2$.

In a similar manner to the timed case (see Definition 4.15), we define substitutions ξ^i_{SR}.

Definition F.3. Let $SR \subseteq \mathcal{C} \times \mathcal{C}$ be a stochastically appropriate bijection. We say that ξ^1_{SR} and ξ^2_{SR} are the *left and right substitutions induced by SR* if they are stochastically appropriate and satisfy

$$\xi^1_{SR}(x_1) = \xi^2_{SR}(x_2) \quad \stackrel{\text{def}}{\iff} \quad \langle x_1, x_2\rangle \in SR$$

According to our convenience we are going to consider either that ξ^i_{SR} is a bijection with domain set $\{x_i \mid \langle x_1, x_2\rangle \in SR\}$ and image containing only fresh variables, or that it is completed with the identity, i.e. $\xi^i_{SR}(y) = y$ whenever $y \notin \{x_i \mid \langle x_1, x_2\rangle \in SR\}$. □

The following lemma states that for all 🏠 term without local conflict of variables, its set of free variables and the set of clock settings it defines do not share any clock in common.

Lemma F.4. *Let $p \in$ 🏠 such that $\neg\mathsf{cv}(p)$. Then $fv(p) \cap \kappa(p) = \varnothing$.*

Proof. We prove that

(*i*) $x \in fv(p) \implies x \notin \kappa(p)$, and

(*ii*) $x \in \kappa(p) \implies x \notin fv(p)$.

Case (*i*). We use induction on the proof tree of $x \in fv(p)$[1] by analysing each syntactic case separately. We only report some paradigmatic cases.

[1] We have not explicitly defined a proof system neither for fv nor for κ. However, the reader could well figure out how it is by looking the set of rules proposed for ♡ in Appendix B.1.

Subcase $C \mapsto p$.

$$\neg\mathsf{cv}(C \mapsto p) \quad \wedge \quad x \in fv(C \mapsto p)$$
$$\Longrightarrow \qquad \text{(by Def. 10.4 (cv) and Def. 10.3 } (fv))$$
$$\neg\mathsf{cv}(p) \quad \wedge \quad (C \cap \kappa(p) = \varnothing) \quad \wedge \quad (x \in C \ \vee \ x \in fv(p))$$
$$\Longrightarrow \qquad \text{(by calculations)}$$
$$(x \in C \ \wedge \ x \notin \kappa(p)) \quad \vee \quad (\neg\mathsf{cv}(p) \ \wedge \ x \in fv(p))$$
$$\Longrightarrow \qquad \text{(by induction)}$$
$$(x \in C \ \wedge \ x \notin \kappa(p)) \quad \vee \quad x \notin \kappa(p) \qquad \Longrightarrow \qquad x \notin \kappa(p)$$
$$\Longrightarrow \qquad \text{(by def. of } \kappa)$$
$$x \notin \kappa(C \mapsto p)$$

Subcase $\{\!|C|\!\}\, p$.

$$\neg\mathsf{cv}(\{\!|C|\!\}\, p) \quad \wedge \quad x \in fv(\{\!|C|\!\}\, p)$$
$$\Longrightarrow \qquad \text{(by Def. 10.4 (cv) and Def. 10.3 } (fv))$$
$$\neg\mathsf{cv}(p) \quad \wedge \quad x \in fv(p) - C \qquad \Longrightarrow \qquad \neg\mathsf{cv}(p) \quad \wedge \quad x \in fv(p) \quad \wedge \quad x \notin C$$
$$\Longrightarrow \qquad \text{(by induction)}$$
$$x \notin C \quad \wedge \quad x \notin \kappa(p) \qquad \Longrightarrow \qquad x \notin C \cup \kappa(p)$$
$$\Longrightarrow \qquad \text{(by def. of } \kappa)$$
$$x \notin \kappa(\{\!|C|\!\}\, p)$$

Subcase $p \,||_A\, q$.

$$\neg\mathsf{cv}(p \,||_A\, q) \quad \wedge \quad x \in fv(p \,||_A\, q)$$
$$\Longrightarrow \qquad \text{(by Def. 10.4 (cv) and Def. 10.3 } (fv))$$
$$\neg\mathsf{cv}(p) \quad \wedge \quad \neg\mathsf{cv}(q) \quad \wedge \quad (\kappa(p) \cap fv(q) = \varnothing) \quad \wedge \quad (\kappa(q) \cap fv(p) = \varnothing)$$
$$\wedge \quad (x \in fv(p) \ \vee \ x \in fv(q))$$
$$\Longrightarrow \qquad \text{(by calculations)}$$
$$(\neg\mathsf{cv}(p) \ \wedge \ x \notin \kappa(q) \ \wedge \ x \in fv(p)) \quad \vee \quad (\neg\mathsf{cv}(q) \ \wedge \ x \notin \kappa(p) \ \wedge \ x \in fv(q))$$
$$\Longrightarrow \qquad \text{(by induction)}$$
$$(x \notin \kappa(q) \ \wedge \ x \notin \kappa(p)) \quad \vee \quad (x \notin \kappa(p) \ \wedge \ x \notin \kappa(q))$$
$$\Longrightarrow \qquad \text{(by calculations)}$$
$$x \notin \kappa(p) \cup \kappa(q)$$
$$\Longrightarrow \qquad \text{(by def. of } \kappa)$$
$$x \notin \kappa(p \,||_A\, q)$$

Case (ii). This case follows in a similar manner by doing induction on the proof of $\kappa(p)$. We also give here a couple of paradigmatic cases.

Subcase $C \mapsto p$.

$$\begin{array}{ll}
\neg\mathsf{cv}(C \mapsto p) \quad \wedge \quad x \in \kappa(C \mapsto p) & \\
\Longrightarrow & \text{(by Def. 10.4 (}\mathsf{cv}\text{) and def. of } \kappa) \\
\quad \neg\mathsf{cv}(p) \quad \wedge \quad (C \cap \kappa(p) = \varnothing) \quad \wedge \quad x \in \kappa(p) & \\
\Longrightarrow & \text{(by calculations)} \\
\quad \neg\mathsf{cv}(p) \quad \wedge \quad x \notin C \quad \wedge \quad x \in \kappa(p) & \\
\Longrightarrow & \text{(by induction)} \\
\quad x \notin C \quad \wedge \quad x \notin \mathit{fv}(p) \qquad \Longrightarrow \qquad x \notin C \cup \mathit{fv}(p) & \\
\Longrightarrow & \text{(by Def. 10.3 (}\mathit{fv}\text{))} \\
\quad x \notin \mathit{fv}(C \mapsto p) &
\end{array}$$

Subcase $\{\!|C|\!\}\, p$.

$$\begin{array}{ll}
\neg\mathsf{cv}(\{\!|C|\!\}\, p) \quad \wedge \quad x \in \kappa(\{\!|C|\!\}\, p) & \\
\Longrightarrow & \text{(by Def. 10.4 (}\mathsf{cv}\text{) and def. of } \kappa) \\
\quad \neg\mathsf{cv}(p) \quad \wedge \quad (x \in C \ \vee \ x \in \kappa(p)) & \\
\Longrightarrow & \text{(by calculations)} \\
\quad x \in C \quad \vee \quad (\neg\mathsf{cv}(p) \ \wedge \ x \in \kappa(p)) & \\
\Longrightarrow & \text{(by induction)} \\
\quad x \in C \quad \vee \quad x \notin \mathit{fv}(p) \qquad \Longrightarrow \qquad x \notin \mathit{fv}(p) - C & \\
\Longrightarrow & \text{(by Def. 10.3 (}\mathit{fv}\text{))} \\
\quad x \notin \mathit{fv}(\{\!|C|\!\}\, p) &
\end{array}$$

Subcase $p \,||_A\, q$.

$$\begin{array}{ll}
\neg\mathsf{cv}(p \,||_A\, q) \quad \wedge \quad x \in \kappa(p \,||_A\, q) & \\
\Longrightarrow & \text{(by Def. 10.4 (}\mathsf{cv}\text{) and def. of } \kappa) \\
\quad \neg\mathsf{cv}(p) \quad \wedge \quad \neg\mathsf{cv}(q) \quad \wedge \quad (\kappa(p) \cap \mathit{fv}(q) = \varnothing) \quad \wedge \quad (\kappa(q) \cap \mathit{fv}(p) = \varnothing) & \\
\quad \wedge \quad (x \in \kappa(p) \ \vee \ x \in \kappa(q)) & \\
\Longrightarrow & \text{(by calculations)} \\
\quad (\neg\mathsf{cv}(p) \ \wedge \ x \notin \mathit{fv}(q) \ \wedge \ x \in \kappa(p)) \quad \vee \quad (\neg\mathsf{cv}(q) \ \wedge \ x \notin \mathit{fv}(p) \ \wedge \ x \in \kappa(q)) & \\
\Longrightarrow & \text{(by induction)} \\
\quad (x \notin \mathit{fv}(q) \ \wedge \ x \notin \mathit{fv}(p)) \quad \vee \quad (x \notin \mathit{fv}(p) \ \wedge \ x \notin \mathit{fv}(q)) & \\
\Longrightarrow & \text{(by calculations)} \\
\quad x \notin \mathit{fv}(p) \cup \mathit{fv}(q) & \\
\Longrightarrow & \text{(by Def. 10.3 (}\mathit{fv}\text{))} \\
\quad x \notin \mathit{fv}(p \,||_A\, q) &
\end{array}$$

Q.E.D.

The following lemma explains how the set of free variables of a given ♤ term is preserved in a derivative.

Lemma F.5. *Let $p \xrightarrow{a,C} p'$ where $\rightarrow$ is obtained using rules in Table 10.3 and rule* **alpha**. *Then $fv(p') \subseteq fv(p) \cup \kappa(p)$.*

Proof. The proof proceed by induction on the proof tree of $\rightarrow$ doing case analysis and taking into account Proposition 10.12(b). We report some paradigmatic cases.

Case $\{\!|C|\!\}\, p$.

$$
\begin{array}{ll}
\{\!|C|\!\}\, p \xrightarrow{a,C'} p' & \\
\Longrightarrow & \text{(by def. of } \rightarrow\text{)} \\
\quad p \xrightarrow{a,C'} p' & \\
\Longrightarrow & \text{(by induction)} \\
\quad fv(p') \subseteq fv(p) \cup \kappa(p) & \\
\Longrightarrow & \text{(by calculations)} \\
\quad fv(p') \subseteq (fv(p) - C) \cup (\kappa(p) \cup C) & \\
\Longrightarrow & \text{(by Def. 10.3 } (fv) \text{ and def. of } \kappa) \\
\quad fv(p') \subseteq fv(\{\!|C|\!\}\, p) \cup \kappa(\{\!|C|\!\}\, p) &
\end{array}
$$

Case $p \,||_A\, q$. We assume $a \notin A$ and p moves. The other cases follow in a similar manner.

$$
\begin{array}{ll}
p \,||_A\, q \xrightarrow{a,C'} p' \,||_A\, \overline{\mathsf{ck}}(q) & \\
\Longrightarrow & \text{(by def. of } \rightarrow \text{ and Def. 10.3 } (fv)) \\
\quad p \xrightarrow{a,C'} p' \quad \wedge \quad fv(\overline{\mathsf{ck}}(q)) \subseteq fv(q) \cup \kappa(q) & \\
\Longrightarrow & \text{(by induction)} \\
\quad fv(p') \subseteq fv(p) \cup \kappa(p) \quad \wedge \quad fv(\overline{\mathsf{ck}}(q)) \subseteq fv(q) \cup \kappa(q) & \\
\Longrightarrow & \text{(by calculations)} \\
\quad fv(p') \cup fv(\overline{\mathsf{ck}}(q)) \subseteq fv(p) \cup fv(q) \cup \kappa(p) \cup \kappa(q) & \\
\Longrightarrow & \text{(by Def. 10.3 } (fv) \text{ and def. of } \kappa) \\
\quad fv(p' \,||_A\, \overline{\mathsf{ck}}(q)) \subseteq fv(p \,||_A\, q) \cup \kappa(p \,||_A\, q) &
\end{array}
$$

Case **alpha**.

$$
\begin{array}{ll}
p \xrightarrow{a,C'} p' & \\
\Longrightarrow & \text{(by } \textbf{alpha}\text{)} \\
\quad p \xrightarrow{a,C'} q \quad \wedge \quad p' \simeq_\alpha q & \\
\Longrightarrow & \text{(by induction)}
\end{array}
$$

$$
\begin{aligned}
& fv(q) \subseteq fv(p) \cup \kappa(p) \quad \wedge \quad p' \simeq_\alpha q \\
\Longrightarrow & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by Prop. 10.12}(b)\text{)} \\
& fv(p') \subseteq fv(p) \cup \kappa(p)
\end{aligned}
$$

Q.E.D.

The following condition on substitutions will be used quite often in the following. Therefore, we dedicate a special remark on it.

Condition F.6. Given $p_1, p_2 \in \spadesuit$, and two substitutions ξ_1 and ξ_2, we require that the following requirements hold.

1. $\neg\mathsf{cv}(p_1)$ and $\neg\mathsf{cv}(p_2)$.
2. ξ_1 and ξ_2 are stochastically appropriate.
3. $\xi_1(p_1) \simeq_\alpha \xi_2(p_2)$.
4. For $i = 1, 2$, $\xi_i(fv(p_i)) \cap (var(p_1) \cup var(p_2)) = \varnothing$, where $var(p)$ is the set of any clock variable appearing in p and in the definition of every process variable reachable from p. In other words, $\xi_i(fv(p_i))$ should be a set of "fresh" clock variables.

In addition, notice that $\xi_1(fv(p_1)) = \xi_2(fv(p_2))$ can be deduced from 3 using Proposition 10.12(b). Hence, the item 4 reduces to ask that $\xi_i(fv(p_i))$ is fresh for at least one of $i = 1, 2$. □

In the following we define a function that, given two α-congruent terms, return the pairs of clocks that have to be renamed in order to calculate that they are indeed α-congruent. However, this is not in general, but only for some relevant clocks in the set of clocks determined by κ. This function is the key on the definition of the renaming bisimulations which prove the theorems.

Definition F.7. We define the function $\mathsf{sr} : (\spadesuit \times \spadesuit) \to (\mathcal{C} \times \mathcal{C})$ as follows. $\mathsf{sr}(p_1, p_2)$ is only defined when there exist ξ_1, and ξ_2 such that p_1, p_2, ξ_1, and ξ_2 are as required by Condition F.6. In this case, sr is defined as the least set satisfying the equations in Table F.1. □

The following proposition serves as an alternative definition for sr.

Proposition F.8. *From the previous definition, the following properties can be derived.*

$$
\begin{aligned}
& \mathsf{sr}(\mathbf{0}, \mathbf{0}) = \mathsf{sr}(a; p_1, a; p_2) = \varnothing \\
& \mathsf{sr}(C_1 \mapsto p_1, C_2 \mapsto p_2) = \mathsf{sr}(p_1[f], p_2[f]) = \mathsf{sr}(p_1, p_2) \\
& \mathsf{sr}(X_1, X_2) = \mathsf{sr}(p_1, p_2) && \text{provided } X_i = p_i,\ i = 1, 2 \\
& \mathsf{sr}(p_1 \oplus q_1, p_2 \oplus q_2) = \mathsf{sr}(p_1, p_2) \cup \mathsf{sr}(q_1, q_2) && \text{with } \oplus \in \{+, ||_A, \lfloor\!\lfloor_A, |_A\} \\
& \mathsf{sr}(\{\!|C_1|\!\}\, p_1, \{\!|C_2|\!\}\, p_2) = \\
& \qquad \big(\{\langle \xi^{-1}_{[C^\star/C_1]}(x), \xi^{-1}_{[C^\star/C_2]}(x)\rangle \mid x \in C^\star\} \cap (fv(p_1) \times fv(p_2))\big) \\
& \qquad \cup\, \mathsf{sr}(p_1, p_2) && \text{with } C^\star \text{ as in Table F.1}
\end{aligned}
$$

$$\frac{\langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2)}{\langle x_1, x_2\rangle \in \mathsf{sr}(C_1 \mapsto p_1, C_2 \mapsto p_2)} \qquad \frac{\langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2)}{\langle x_1, x_2\rangle \in \mathsf{sr}(\{\!|C_1|\!\}\, p_1, \{\!|C_2|\!\}\, p_2)}$$

$$\frac{\langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2)}{\langle x_1, x_2\rangle \in \mathsf{sr}(p_1[f], p_2[f])} \qquad \frac{\langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2) \quad X_i = p_i,\ i = 1, 2}{\langle x_1, x_2\rangle \in \mathsf{sr}(X_1, X_2)}$$

$$\frac{\langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2) \quad \oplus \in \{+, ||_A, \between_A, |_A\}}{\langle x_1, x_2\rangle \in \mathsf{sr}(p_1 \oplus q_1, p_2 \oplus q_2) \quad \langle x_1, x_2\rangle \in \mathsf{sr}(q_1 \oplus p_1, q_2 \oplus p_2)}$$

$$\frac{\langle x_1, x_2\rangle \in \{\langle \xi^{-1}_{[C^\star/C_1]}(x), \xi^{-1}_{[C^\star/C_2]}(x)\rangle \mid x \in C^\star\} \cap (fv(p_1) \times fv(p_2))}{\langle x_1, x_2\rangle \in \mathsf{sr}(\{\!|C_1|\!\}\, p_1, \{\!|C_2|\!\}\, p_2)}$$

where $C^\star$ is such that

- (i) $C^\star \cap (var(p_1) \cup var(p_2)) = \varnothing$ (i.e., $C^\star$ is fresh);
- (ii) $\xi_{[C^\star/C_1]}$ and $\xi_{[C^\star/C_2]}$ are stochastically appropriate; and
- (iii) $\xi_1\xi_{[C^\star/C_1]}(p_1) \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}(p_2)$.

Table F.1: Definition of sr

Proposition F.9. *For $i = 1, 2$, $\kappa(p_i) \cap rel(p_i) \subseteq \{x_i \mid \langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2)\} \subseteq \kappa(p_i)$.*

Proof sketch. By induction on the proof of $\langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2)$, it can be shown that

$$\langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2) \implies x_i \in \kappa(p_i).$$

The case of $\kappa(p_i) \cap rel(p_i) \subseteq \{x_i \mid \langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2)\}$ follows from the observation that once a pair $\langle x_1, x_2\rangle$ is "put" in $\mathsf{sr}(p_1, p_2)$, it is "maintained" along the proof. The only case in which a new pair is inserted is for the syntactic case $\{\!|C_i|\!\}\, p_i$. Notice that the only reason not to introduce a pair $\langle x_1, x_2\rangle$ with clocks $x_i \in C_i$ is if $x_i \notin fv(p_i) \supseteq FV(p_i)$. In this case, either $x_i \notin \kappa(p_i)$, from which clearly $x_i \notin rel(p_i) = rel(\{\!|C_i|\!\}\, p_i)$, or $x_i \in \kappa(p_i)$, in which case $x_i \in rel(\{\!|C_i|\!\}\, p_i)$ if and only if $x_i \in rel(p_i)$, and hence, it inductively depends on the processes p_i's. *Q.E.D.*

Lemma F.10. *For all p_1 and p_2 such that there are ξ_1 and ξ_2 satisfying Condition F.6, $\mathsf{sr}(p_1, p_2)$ is a stochastically appropriate function.*

Proof. We have to prove that

1. if $\langle x, y\rangle, \langle x, z\rangle \in \mathsf{sr}(p_1, p_2)$ then $y = z$, i.e., $\mathsf{sr}(p_1, p_2)$ is a function;
2. if $\langle x, z\rangle, \langle y, z\rangle \in \mathsf{sr}(p_1, p_2)$ then $x = y$, i.e., $\mathsf{sr}(p_1, p_2)$ is injective; and
3. if $\langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2)$ then $F_{x_1} = F_{x_2}$, i.e., $\mathsf{sr}(p_1, p_2)$ preserves distributions.

Case 1. We proceed by induction on the proof trees of $\langle x, y\rangle, \langle x, z\rangle \in \mathsf{sr}(p_1, p_2)$ doing syntactic case analysis. We only consider some paradigmatic cases.

Subcase $\{\!|C|\!\}\, p$. Suppose $\langle x, y\rangle, \langle x, z\rangle \in \mathsf{sr}(\{\!|C_1|\!\}\, p_1, \{\!|C_2|\!\}\, p_2)$. Then three different cases may arise.

$$\langle x, y\rangle, \langle x, z\rangle \in \mathsf{sr}(p_1, p_2); \tag{*}$$
$$\langle x, y\rangle, \langle x, z\rangle \in \{\langle \xi^{-1}_{[C^\star/C_1]}(x), \xi^{-1}_{[C^\star/C_2]}(x)\rangle \mid x \in C^\star\} \cap (fv(p_1) \times fv(p_2)); \text{ or} \tag{**}$$
$$\langle x, y\rangle \in \mathsf{sr}(p_1, p_2) \quad \wedge \quad \langle x, z\rangle \in \{\langle \xi^{-1}_{[C^\star/C_1]}(x), \xi^{-1}_{[C^\star/C_2]}(x)\rangle \mid x \in C^\star\} \cap (fv(p_1) \times fv(p_2)). \tag{***}$$

Case (*) follows immediately by induction. Case (**) is an immediate consequence from the fact that both $\xi_{[C^\star/C_i]}$ are bijective functions. The remaining case (***) is not possible since, by Proposition F.9 and simple calculations, we have that $x \in \kappa(p_1) \wedge x \in fv(p_1)$ which contradicts Lemma F.4.

Subcase $p \,||_A\, q$. Suppose $\langle x, y\rangle, \langle x, z\rangle \in \mathsf{sr}(p_1 \,||_A\, q_1, p_2 \,||_A\, q_2)$. Then three different cases may arise.

$$\langle x, y\rangle, \langle x, z\rangle \in \mathsf{sr}(p_1, p_2); \tag{*}$$
$$\langle x, y\rangle, \langle x, z\rangle \in \mathsf{sr}(q_1, q_2); \text{ or} \tag{**}$$
$$\langle x, y\rangle \in \mathsf{sr}(p_1, p_2) \quad \wedge \quad \langle x, z\rangle \in \mathsf{sr}(q_1, q_2). \tag{***}$$

Cases (*) and (**) follow immediately by induction. Case (***) is not possible since, by Proposition F.9, $x \in \kappa(p_1) \wedge x \in \kappa(q_1)$ which is not possible since $\neg\mathsf{cv}(p_1)$ and hence $\kappa(p_1) \cap \kappa(q_1) = \varnothing$.

Case 2. Notice that $\mathsf{sr}(p_1, p_2)^{-1} = \mathsf{sr}(p_2, p_1)$. Then, this case follows immediately from the previous one.

Case 3. We proceed by induction on the proof tree of $\langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2)$ doing syntactic case analysis. For all the cases, it is either trivial or follows immediately by induction, except if $\langle x_1, x_2\rangle \in \mathsf{sr}(\{\!|C_1|\!\}\, p_1, \{\!|C_2|\!\}\, p_2)$. In this last case, either $\langle x_1, x_2\rangle \in \mathsf{sr}(p_1, p_2)$, and by induction Case 3 follows, or

$$\langle x_1, x_2\rangle \in \{\langle \xi^{-1}_{[C^\star/C_1]}(x), \xi^{-1}_{[C^\star/C_2]}(x)\rangle \mid x \in C^\star\} \cap (fv(p_1) \times fv(p_2)).$$

But both $\xi_{[C^\star/C_i]}$ are stochastically appropriate bijections. As a consequence, if we take $x = \xi_{[C^\star/C_1]}(x_1) = \xi_{[C^\star/C_2]}(x_2)$, $F_{x_1} = F_x = F_{x_2}$. *Q.E.D.*

Lemma F.11. *Let $p_1, p_2 \in \mathbb{O}$, and let ξ_1 and ξ_2 be two substitutions such that they satisfy Condition F.6. Then, $\xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(\overline{\mathsf{ck}}(p_1)) \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(\overline{\mathsf{ck}}(p_2))$.*

Proof. Because of Condition F.6, $\xi_1(p_1) \simeq_\alpha \xi_2(p_2)$. Thus, we will proceed by induction on its proof tree, by separated case analysis on the syntactic form. We only report on the paradigmatic cases.

Case $\{\!|C|\!\}\, p$.

$$
\begin{array}{l}
\xi_1(\{\!|C_1|\!\}\, p_1) \simeq_\alpha \xi_2(\{\!|C_2|\!\}\, p_2) \\
\Longrightarrow \qquad \text{(by Def. 10.10 (substitution), Def. 10.11 } (\simeq_\alpha)\text{, and calculations)} \\
\quad \xi_1\xi_{[C^\star/C_1]}(p_1) \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}(p_2) \quad \wedge \quad C^\star \text{ is ``fresh''} \\
\Longrightarrow \qquad \text{(by induction)} \\
\quad \xi_1\xi_{[C^\star/C_1]}\xi^1_{\mathsf{sr}(p_1,p_2)}(\overline{\mathsf{ck}}(p_1)) \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}\xi^2_{\mathsf{sr}(p_1,p_2)}(\overline{\mathsf{ck}}(p_2)) \quad \wedge \quad C^\star \text{ is ``fresh''} \qquad (*)
\end{array}
$$

Let $SR^\star = \{\langle \xi^{-1}_{[C^\star/C_1]}(x), \xi^{-1}_{[C^\star/C_2]}(x)\rangle \mid x \in C^\star\}$ and let $SR = SR^\star \cap (fv(p_1) \times fv(p_2))$. Then we have:

$$
\begin{array}{l}
(*) \implies \\
\quad \xi_1\xi^1_{SR^\star}\xi^1_{\mathsf{sr}(p_1,p_2)}(\overline{\mathsf{ck}}(p_1)) \simeq_\alpha \xi_2\xi^2_{SR^\star}\xi^2_{\mathsf{sr}(p_1,p_2)}(\overline{\mathsf{ck}}(p_2)) \\
\Longrightarrow \qquad \text{(since } \neg\mathsf{cv}(p_i) \text{ using Lemma F.4)} \\
\quad \xi_1\xi^1_{SR}\xi^1_{\mathsf{sr}(p_1,p_2)}(\overline{\mathsf{ck}}(p_1)) \simeq_\alpha \xi_2\xi^2_{SR}\xi^2_{\mathsf{sr}(p_1,p_2)}(\overline{\mathsf{ck}}(p_2)) \\
\Longrightarrow \qquad \text{(since } \neg\mathsf{cv}(p_i) \text{ using Lemma F.4)} \\
\quad \xi_1\xi^1_{SR\cup\mathsf{sr}(p_1,p_2)}(\overline{\mathsf{ck}}(p_1)) \simeq_\alpha \xi_2\xi^2_{SR\cup\mathsf{sr}(p_1,p_2)}(\overline{\mathsf{ck}}(p_2)) \\
\Longrightarrow \qquad \text{(by Def. F.7 (}\mathsf{sr}\text{) and def. of } \overline{\mathsf{ck}}) \\
\quad \xi_1\xi^1_{\mathsf{sr}(\{\!|C_1|\!\}\, p_1,\{\!|C_2|\!\}\, p_2)}(\overline{\mathsf{ck}}(\{\!|C_1|\!\}\, p_1)) \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(\{\!|C_1|\!\}\, p_1,\{\!|C_2|\!\}\, p_2)}(\overline{\mathsf{ck}}(\{\!|C_2|\!\}\, p_2))
\end{array}
$$

Case $p \,||_A\, q$.

$$
\begin{array}{l}
\xi_1(p_1 \,||_A\, q_1) \simeq_\alpha \xi_2(p_2 \,||_A\, q_2) \\
\Longrightarrow \qquad \text{(by Def. 10.10 (substitution), and Def. 10.11 } (\simeq_\alpha)) \\
\quad \xi_1(p_1) \simeq_\alpha \xi_2(p_2) \quad \wedge \quad \xi_1(q_1) \simeq_\alpha \xi_2(q_2) \\
\Longrightarrow \qquad \text{(by induction)} \\
\quad \xi_1(\overline{\mathsf{ck}}(p_1)) \simeq_\alpha \xi_2(\overline{\mathsf{ck}}(p_2)) \quad \wedge \quad \xi_1(\overline{\mathsf{ck}}(q_1)) \simeq_\alpha \xi_2(\overline{\mathsf{ck}}(q_2)) \\
\Longrightarrow \qquad \text{(by Def. 10.11 } (\simeq_\alpha)\text{, Def. 10.10 (substitution), and def. of } \overline{\mathsf{ck}}) \\
\quad \xi_1(\overline{\mathsf{ck}}(p_1 \,||_A\, q_1)) \simeq_\alpha \xi_2(\overline{\mathsf{ck}}(p_2 \,||_A\, q_2))
\end{array}
$$

Q.E.D.

The proof in Theorems 10.17 and 10.18 of the transfer properties 4 and 5 in Definition F.1 stands on the following lemma.

Lemma F.12. *Let* $p_1, p_2 \in \text{⌂}$, *and let* ξ_1 *and* ξ_2 *be two substitutions such that they satisfy Condition F.6, in particular* $\xi_1(p_1) \simeq_\alpha \xi_2(p_2)$. *Consider* $\longrightarrow$ *to be derived using rules in Table 10.3 or rule* **alpha** *(page 144). Then, the following statements hold.*

(*a*) $p_1 \xrightarrow{a,C_1} p_1'$ *implies that there are* $p_2' \in \text{⌂}$ *and* $C_2 \subseteq \mathcal{C}$, *such that*

$$p_2 \xrightarrow{a,C_2} p_2',$$
$$\xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2), \quad \textit{and}$$
$$\xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2').$$

(*b*) $p_2 \xrightarrow{a,C_2} p_2'$ *implies that there are* $p_1' \in \text{⌂}$ *and* $C_1 \subseteq \mathcal{C}$, *such that*

$$p_1 \xrightarrow{a,C_1} p_1',$$
$$\xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2), \quad \textit{and}$$
$$\xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2').$$

Proof. We only prove property (*a*). Property (*b*) follows from (*a*) by symmetry of $\simeq_\alpha$ and Condition F.6. We proceed by induction on the depth of the proof tree of $\rightarrow$, doing analysis by syntactic case.

Case **0** is trivial and case $a;p$ is straightforward. We omit the proofs of cases $p[f]$ and X since they follow similar reasoning to the one of $C \mapsto p$. We also omit subcases $p \between_A q$ and $p \mid_A q$ since they are particular cases of $p \,||_A\, q$.

Case $C \mapsto p$.

$$\xi_1(C_1 \mapsto p_1) \simeq_\alpha \xi_2(C_2 \mapsto p_2) \quad \wedge \quad C_1 \mapsto p_1 \xrightarrow{a,C_1 \cup C_1'} p_1'$$
$$\Longrightarrow \qquad \text{(by Def. 10.10 (substitution), Def. 10.11 } (\simeq_\alpha)\text{, and def. of } \rightarrow)$$
$$\xi_1(C_1) = \xi_2(C_2) \quad \wedge \quad \xi_1(p_1) \simeq_\alpha \xi_2(p_2) \quad \wedge \quad p_1 \xrightarrow{a,C_1'} p_1'$$
$$\Longrightarrow \qquad \text{(by induction)}$$
$$\xi_1(C_1) = \xi_2(C_2) \quad \wedge \quad p_2 \xrightarrow{a,C_2'} p_2'$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1') = \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2')$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2')$$
$$\Longrightarrow \qquad \text{(by def. of } \rightarrow\text{, Def. F.7 } (\mathsf{sr})\text{, and Lemma F.4)}$$
$$C_2 \mapsto p_2 \xrightarrow{a,C_2 \cup C_2'} p_2'$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(C_1 \mapsto p_1, C_2 \mapsto p_2)}(C_1 \cup C_1') = \xi_2\xi^2_{\mathsf{sr}(C_1 \mapsto p_1, C_2 \mapsto p_2)}(C_2 \cup C_2')$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(C_1 \mapsto p_1, C_2 \mapsto p_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(C_1 \mapsto p_1, C_2 \mapsto p_2)}(p_2')$$

Case $\{\!|C|\!\}\, p$.

$$\xi_1(\{\!|C_1|\!\}\, p_1) \simeq_\alpha \xi_2(\{\!|C_2|\!\}\, p_2) \quad \wedge \quad \{\!|C_1|\!\}\, p_1 \xrightarrow{a,C_1'} p_1'$$
$$\Longrightarrow \qquad \text{(by Def. 10.10 (substitution), Def. 10.11 } (\simeq_\alpha)\text{, and def. of } \rightarrow)$$
$$\xi_1\xi_{[C^\star/C_1]}(p_1) \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}(p_2) \quad \wedge \quad p_1 \xrightarrow{a,C_1'} p_1'$$
$$\Longrightarrow \qquad \text{(by induction)}$$

$$
\begin{array}{ll}
& p_2 \xrightarrow{a,C_2'} p_2' \\
\wedge & \xi_1\xi_{[C^\star/C_1]}\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1') = \xi_2\xi_{[C^\star/C_2]}\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2') \\
\wedge & \xi_1\xi_{[C^\star/C_1]}\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1') \simeq_\alpha \xi_2\xi_{[C^\star/C_2]}\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2') \\
\Longrightarrow & \qquad\qquad \text{(proceeding as in the proof of Lemma F.11, page 269, break (*))} \\
& p_2 \xrightarrow{a,C_2'} p_2' \\
\wedge & \xi_1\xi^1_{\mathsf{sr}(\{\!|C_1|\!\}\, p_1,\{\!|C_2|\!\}\, p_2)}(C_1') = \xi_2\xi^2_{\mathsf{sr}(\{\!|C_1|\!\}\, p_1,\{\!|C_2|\!\}\, p_2)}(C_2') \\
\wedge & \xi_1\xi^1_{\mathsf{sr}(\{\!|C_1|\!\}\, p_1,\{\!|C_2|\!\}\, p_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(\{\!|C_1|\!\}\, p_1,\{\!|C_2|\!\}\, p_2)}(p_2')
\end{array}
$$

Case $p + q$.

$$
\begin{array}{ll}
& p_1 + q_1 \xrightarrow{a,C_1} p_1' \\
\Longrightarrow & \qquad\qquad \text{(by def. of } \rightarrow\text{)} \\
& p_1 \xrightarrow{a,C_1} p_1' \qquad (*) \\
\vee & q_1 \xrightarrow{a,C_1} p_1' \qquad (**)
\end{array}
$$

Assume (*) is the case. Case (**) proceeds in a similar way. Then, we have:

$$
\begin{array}{ll}
& \xi_1(p_1 + q_1) \simeq_\alpha \xi_2(p_2 + q_2) \quad \wedge \quad p_1 \xrightarrow{a,C_1} p_1' \\
\Longrightarrow & \qquad\qquad \text{(by Def. 10.10 (substitution) and Def. 10.11 } (\simeq_\alpha)) \\
& \xi_1(p_1) \simeq_\alpha \xi_2(p_2) \quad \wedge \quad p_1 \xrightarrow{a,C_1} p_1' \\
\Longrightarrow & \qquad\qquad \text{(by induction)} \\
& p_2 \xrightarrow{a,C_2} p_2' \quad \wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2) \\
& \qquad\qquad\quad \wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2') \\
\Longrightarrow & \qquad \text{(by def. of } \rightarrow \text{ and Def. F.7 } (\mathsf{sr}), \text{ since } \neg\mathsf{cv}(p_i + q_i) \text{ and Lemma F.5)} \\
& p_2 + q_2 \xrightarrow{a,C_2} p_2' \quad \wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1+q_1,p_2+q_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1+q_1,p_2+q_2)}(C_2) \\
& \qquad\qquad\quad \wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1+q_1,p_2+q_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1+q_1,p_2+q_2)}(p_2')
\end{array}
$$

Subcase $p \,||_A\, q$.

$$
\begin{array}{ll}
& p_1 \,||_A\, q_1 \xrightarrow{a,C_1} p_1' \,||_A\, q_1' \\
\Longrightarrow & \qquad\qquad \text{(by def. of } \rightarrow\text{)} \\
& (p_1 \xrightarrow{a,C_1} p_1' \quad \wedge \quad a \notin A \quad \wedge \quad q_1' = \overline{\mathsf{ck}}(q_1)) \qquad (*) \\
\vee & (q_1 \xrightarrow{a,C_1} q_1' \quad \wedge \quad a \notin A \quad \wedge \quad p_1' = \overline{\mathsf{ck}}(p_1)) \qquad (**) \\
\vee & (p_1 \xrightarrow{a,C_1'} p_1' \quad \wedge \quad q_1 \xrightarrow{a,C_1''} q_1' \quad \wedge \quad C_1 \equiv C_1' \cup C_1'' \quad \wedge \quad a \in A) \qquad (***)
\end{array}
$$

Assume (*) is the case. Case (**) is analogous to (*), and case (***) follows the same lines but it is simpler. Then, we proceed as follows.

$$\xi_1(p_1 \,||_A\, q_1) \simeq_\alpha \xi_2(p_2 \,||_A\, q_2) \quad \wedge \quad p_1 \xrightarrow{a,C_1} p_1'$$
$\Longrightarrow$ (by Def. 10.10 (substitution) and Def. 10.11 ($\simeq_\alpha$))
$$\xi_1(p_1) \simeq_\alpha \xi_2(p_2) \quad \wedge \quad \xi_1(q_1) \simeq_\alpha \xi_2(q_2) \quad \wedge \quad p_1 \xrightarrow{a,C_1} p_1'$$
$\Longrightarrow$ (by Lemma F.11 and induction)
$$\xi_1\xi^1_{\mathsf{sr}(q_1,q_2)}(\overline{\mathsf{ck}}(q_1)) \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(q_1,q_2)}(\overline{\mathsf{ck}}(q_2))$$
$$\wedge \quad p_2 \xrightarrow{a,C_2} p_2'$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2)$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2')$$
$\Longrightarrow$ (by Def. F.7 (sr), considering that $\neg\mathsf{cv}(p_i \,||_A\, q_i)$ and Lemma F.5)
$$\xi_1\xi^1_{\mathsf{sr}(p_1||_A q_1,p_2||_A q_2)}(\overline{\mathsf{ck}}(q_1)) \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1||_A q_1,p_2||_A q_2)}(\overline{\mathsf{ck}}(q_2))$$
$$\wedge \quad p_2 \xrightarrow{a,C_2} p_2'$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1||_A q_1,p_2||_A q_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1||_A q_1,p_2||_A q_2)}(C_2)$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1||_A q_1,p_2||_A q_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1||_A q_1,p_2||_A q_2)}(p_2')$$
$\Longrightarrow$ (by def. of $\rightarrow$, Def. 10.11 ($\simeq_\alpha$), and Def. 10.10 (substitution))
$$p_2 \,||_A\, q_2 \xrightarrow{a,C_2} p_2' \,||_A\, \overline{\mathsf{ck}}(q_2)$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1||_A q_1,p_2||_A q_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1||_A q_1,p_2||_A q_2)}(C_2)$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1||_A q_1,p_2||_A q_2)}(p_1' \,||_A\, \overline{\mathsf{ck}}(q_1)) \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1||_A q_1,p_2||_A q_2)}(p_2' \,||_A\, \overline{\mathsf{ck}}(q_2))$$

Subcase **alpha**. Let $p_1 \xrightarrow{a,C_1} p_1'$ and suppose it was obtained by applying rule **alpha**. Then, for some p_1'', $p_1 \xrightarrow{a,C_1} p_1''$, $p_1'' \simeq_\alpha p_1'$, and $\neg\mathsf{cv}(p_1')$. Hence, we proceed,

$$\xi_1(p_1) \simeq_\alpha \xi_2(p_2) \quad \wedge \quad p_1 \xrightarrow{a,C_1} p_1'' \quad \wedge \quad p_1'' \simeq_\alpha p_1'$$
$\Longrightarrow$ (by induction)
$$p_2 \xrightarrow{a,C_2} p_2'$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2)$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1'') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2')$$
$$\wedge \quad p_1'' \simeq_\alpha p_1'$$
$\Longrightarrow$ (substitution preserves $\simeq_\alpha$)
$$p_2 \xrightarrow{a,C_2} p_2'$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2)$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1'') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2')$$
$$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1'') \simeq_\alpha \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1')$$

$\Longrightarrow$ (by transitivity of $\simeq_\alpha$)

$p_2 \xrightarrow{a,C_2} p_2'$
$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2)$
$\wedge \quad \xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2')$

Q.E.D.

The proof of the following lemma follows as its similar for ♡, namely Lemma B.4.

Lemma F.13. *Let $p \in \spadesuit^\alpha$. Then $FV(p) \subseteq fv(p)$.*

Lemma F.14. *Let $p_1, p_2 \in \spadesuit^\alpha$, and let ξ_1 and ξ_2 be two substitutions such that they satisfy Condition F.6. Then, $\xi_1(FV(p_1)) = \xi_2(FV(p_2))$.*

Proof. We first give an inductive characterisation of FV. Let $(\mathcal{S}, \mathcal{C}, \mathcal{A}, \rightarrow, \kappa)$ be a stochastic automaton. For each $s \in \mathcal{S}$, $n \geq 0$, we define

$$\begin{array}{lcl} FV_0(s) & = & \varnothing \\ FV_{n+1}(s) & = & \big(\bigcup\{C \cup FV_n(s') \mid \exists a \in \mathcal{A}.\ s \xrightarrow{a,C} s'\}\big) - \kappa(s) \end{array}$$

Then $FV(s) = \bigcup_{n=0}^{\infty} FV_n(s)$.

Now, we prove that for all n, $\xi_1(FV_n(p_1)) \subseteq \xi_2(FV_n(p_2))$, or more accurately, that for all n, for all x, $\xi_1^{-1}(x) \in FV_n(p_1)$ implies $\xi_2^{-1}(x) \in FV_n(p_2)$ (remember that both ξ_i are injective) from which the lemma follows by symmetry and the previous observation. We apply induction on n. The case $n = 0$ is trivial.

Suppose $\xi_1^{-1}(x) \in FV_{n+1}(p_1)$. Then,

$$\xi_1^{-1}(x) \in \big(\bigcup\{C \cup FV_n(p_1') \mid \exists a \in \mathcal{A}.\ p_1 \xrightarrow{a,C} p_1'\}\big) - \kappa(p_1)$$

But $\xi_1^{-1}(x) \in FV_{n+1}(p_1) \subseteq FV(p_1) \subseteq fv(p_1)$ by Lemma F.13. Thus, by Lemma F.4 $\xi_1^{-1}(x) \notin \kappa(p_1)$.

Therefore, it reduces to consider that there is an edge $p_1 \xrightarrow{a,C_1}{}' p_1'$ such that

1. $\xi_1^{-1}(x) \in C_1$, or
2. $\xi_1^{-1}(x) \in FV_n(p_1')$.

By Lemma F.12, rule **alpha**, and transitivity of $\simeq_\alpha$, we have that there are C_2 and p_2' such that

$p_2 \xrightarrow{a,C_2}{}' p_2'$,
$\xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1) = \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2)$, and
$\xi_1\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1') \simeq_\alpha \xi_2\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2')$.

In addition, since $x \in \xi_1(fv(p_1)) = \xi_2(fv(p_2))$, then $\xi_2^{-1}(x) \notin \kappa(p_2)$, and as a consequence $\xi_2^{-1}(x) = (\xi_2 \xi^2_{\mathsf{sr}(p_1,p_2)})^{-1}(x)$.

Thus, for case 1 we have that $\xi_2^{-1}(x) \in C_2 \subseteq FV_{n+1}(p_2) \cup \kappa(p_2)$, and by the observation above, $\xi_2^{-1}(x) \in FV_{n+1}(p_2)$.

Otherwise, for case 2, by induction $(\xi_2 \xi^2_{\mathsf{sr}(p_1,p_2)})^{-1}(x) \in FV_n(p_2') \subseteq FV_{n+1}(p_2) \cup \kappa(p_2)$, and henceforth $\xi_2^{-1}(x) \in FV_{n+1}(p_2)$. *Q.E.D.*

The proof of the following lemma follows by induction on n using FV_n.

Lemma F.15. *Let $p \in \mathbb{O}^\alpha$, and let ξ be a stochastically appropriate substitutions. Then $FV(\xi(p)) = \xi(FV(p))$.*

Corollary F.15.1. *Let $p_1, p_2 \in \mathbb{O}$, and let ξ_1 and ξ_2 be two substitutions such that they satisfy Condition F.6. Then, $\xi_1(rel(p_1)) = rel(\xi_1(p_1)) = rel(\xi_2(p_2)) = \xi_2(rel(p_2))$.*

Theorem 10.17. *Let $p, q \in \mathbb{O}^\alpha$; so, their behaviour is not only locally definable but also α-definable. If $p \simeq_\alpha q$ then $p \sim_\& q$ where p and q should be interpreted here as states in $SA(\mathbb{O}^\alpha)$.*

Proof. We actually prove that there is a renaming bisimulation between p and q satisfying the additional conditions in Lemma F.2, from which the theorem follows.

Let R_r be the least relation satisfying the following rules

$$\frac{p_1 \simeq_\alpha p_2 \qquad \neg\mathsf{cv}(p_1) \qquad \neg\mathsf{cv}(p_2) \qquad SR = \mathsf{sr}(p_1,p_2) \cup \{\langle x,x\rangle \mid x \in fv(p_1) = fv(p_2)\}}{\langle p_1, p_2, SR\rangle \in R_r} \tag{F.1}$$

$$\frac{\langle p_1,p_2,SR\rangle \in R_r \qquad p_1 \xrightarrow{a,C_1}{}' p_1' \qquad p_2 \xrightarrow{a,C_2}{}' p_2' \qquad \xi^1_{SR}(C_1) = \xi^2_{SR}(C_2) \qquad \xi^1_{SR}(p_1') \simeq_\alpha \xi^2_{SR}(p_2') \qquad SR' = \mathsf{sr}(p_1',p_2') \cup \{\langle x,y\rangle \in SR \mid x \in fv(p_1') \vee y \in fv(p_2')\}}{\langle p_1', p_2', SR'\rangle \in R_r} \tag{F.2}$$

where ξ^i_{SR} are as in Definition F.3. We prove that R_r is a renaming bisimulation. We will need the following claim.

Claim F.16. *$\langle p_1, p_2, SR\rangle \in R_r$ implies $\xi^1_{SR}(p_1) \simeq_\alpha \xi^2_{SR}(p_2)$.*

Proof. If $\langle p_1, p_2, SR\rangle \in R_r$ is derived by rule (F.1), the claim follows immediately since, by Lemma F.4, $id \cap (fv(p_1) \times fv(p_2)) = \xi^i_{SR} \cap (fv(p_1) \times fv(p_2))$.

If instead $\langle p_1, p_2, SR\rangle \in R_r$ is derived by rule (F.2) then $\xi^1_{SR'}(p_1) \simeq_\alpha \xi^2_{SR'}(p_2)$ with $SR = \mathsf{sr}(p_1,p_2) \cup \{\langle x,y\rangle \in SR' \mid x \in fv(p_1) \vee y \in fv(p_2)\}$ for some SR'. Using Lemma F.4, we can then conclude that $\xi^i_{SR'} \cap (fv(p_1) \times fv(p_2)) = \xi^i_{SR} \cap (fv(p_1) \times fv(p_2))$, and as a consequence $\xi^1_{SR}(p_1) \simeq_\alpha \xi^2_{SR}(p_2)$. *Q.E.D. (Claim)*

We proceed to show that R_r satisfies the properties in Definition F.1. So, assume $\langle p_1, p_2, SR\rangle \in R_r$.

1. It follows straightforwardly by induction on the proof tree of $\langle p_1, p_2, SR\rangle \in R_r$ using Lemmas F.10 and F.4.

2. The fact that $\{x_i \mid \langle x_1, x_2\rangle \in SR\} \supseteq rel(s_i)$, for $i = 1, 2$, follows from Proposition F.9.

   Using first Claim F.16 and then Corollary F.15.1 we can conclude that if $\langle x_1, x_2\rangle \in SR$, then $x_1 \in rel(s_1) \iff x_2 \in rel(s_2)$.

3. It follows by Proposition F.9 and the fact that $\xi^1_{SR}(fv(p_1)) = \xi^2_{SR}(fv(p_2))$.

4. Suppose $p_1 \xrightarrow{a,C_1}{}' p_1'$. Then $p_1 \xrightarrow{a,C_1} p_1'$. By Claim F.16, $\xi^1_{SR}(p_1) \simeq_\alpha \xi^2_{SR}(p_2)$. Using Lemma F.12($a$), it follows that there are $p_2' \in \text{⏚}$ and $C_2 \subseteq \mathcal{C}$, such that

   $$p_2 \xrightarrow{a,C_2} p_2'', \tag{F.3}$$
   $$\xi^1_{SR}\xi^1_{\mathsf{sr}(p_1,p_2)}(C_1) = \xi^2_{SR}\xi^2_{\mathsf{sr}(p_1,p_2)}(C_2), \text{ and} \tag{F.4}$$
   $$\xi^1_{SR}\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1') \simeq_\alpha \xi^2_{SR}\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2''). \tag{F.5}$$

   From here, we have:

   (a) Since the behaviour of p_2 is α-definable, then there is a p_2' such that $\neg\mathsf{cv}(p_2')$, $p_2'' \simeq_\alpha p_2'$. By (F.3) and rule **alpha**, $p_2 \xrightarrow{a,C_2}{}' p_2'$.

   (b) Since SR is a bijective function, by (F.4), if $\langle x_1, x_2\rangle \in SR$, then $x_1 \in C_1 \iff x_2 \in C_2$.

   (c) Since $p_2'' \simeq_\alpha p_2'$, then $\xi^2_{SR}\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2'') \simeq_\alpha \xi^2_{SR}\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2')$. By (F.5), it follows that $\xi^1_{SR}\xi^1_{\mathsf{sr}(p_1,p_2)}(p_1') \simeq_\alpha \xi^2_{SR}\xi^2_{\mathsf{sr}(p_1,p_2)}(p_2')$. Take SR' as in rule (F.2). Then $\langle p_1', p_2', SR'\rangle \in R_r$. It remains to prove that

   $$\{\langle x_1, x_2\rangle \in SR \mid x_1 \in FV(p_1') \vee x_2 \in FV(p_2')\} - (C_1 \times C_2) = \{\langle x_1, x_2\rangle \in SR' \mid x_1 \in FV(p_1') \vee x_2 \in FV(p_2')\} - (C_1 \times C_2).$$

   which follows by definition of SR' since $FV(p_i') \subseteq fv(p_i')$ (see Lemma F.13).

5. This case is analogous to the previous one. *Q.E.D.*

Theorem 10.18. *Let $p \in \text{⏚}^{\mathsf{cf}}$; so, the behaviour of p is definable. Then, the rooted stochastic automata $(SA(\text{⏚}^{\mathsf{cf}}), p)$ and $(SA(\text{⏚}^{\alpha}), p)$ are symbolically bisimilar.*

Proof. Similar to the proof of Theorem 10.17 above. *Q.E.D.*

F.2 Proof of Theorem 10.24

In this section, we prove that $\sim_o$ is a congruence in $\spadesuit$. The theorem has being stated as follows in Chapter 10.

Theorem 10.24. *Let p and q be two $\spadesuit$ terms whose behaviours are α-definable. Assume $p \sim_o q$. Let $\mathbf{C}[\ ]$ be a $\spadesuit$ context such that $\mathbf{C}[p]$ and $\mathbf{C}[q]$ are α-definable. Then, it holds that $\mathbf{C}[p] \sim_o \mathbf{C}[q]$.*

We dedicate a different subsection to prove that every operation of $\spadesuit$ preserves open p-bisimilarity. The theorem follows from Lemmas F.18, F.19, F.22, F.23, F.25, F.26, and F.27 given on the remaining of this appendix.

Preliminary considerations

Since we want to prove that $\sim_o$ is a congruence, all probabilistic and non deterministic states, as well as the transition relations, come from the interpretation as an opens system of the stochastic automaton up to α-conversion derived from $\spadesuit$, (Definitions 9.7 and 10.15, respectively).

We recall that, according to Definition 9.9, $p \sim_o q$ if and only if for every valuation v $(p, v) \sim_p (q, v)$. Moreover, for each of the operations, we will check that a given relation is a probabilistic bisimulation up to $\sim_p$. Therefore, we have to check that the transfer properties given in Definition 8.10 hold. In the proofs, we number the transfer property that is being proved. We also recall that, by Theorem 10.17, if $p \simeq_\alpha q$ then $p \sim_\& q$ and as a consequence $p \sim_o q$. We will usually use these facts without explicitly referring to it.

We will use some particular notation. In general we denote by Σ' the set of non-deterministic states $\left[\spadesuit^{\mathsf{cf}} \times \mathbf{V}\right]$. Moreover, we let Ω^p_v, $\mathcal{F}^p_v$, and P^p_v be the sample space, the σ-algebra, and the probability measure of the probability space $T\,(p, v)$, respectively, where (p, v) is some probabilistic state in the open behaviour and T is the probabilistic transition. We will in general let i range over $\{1, 2\}$.

Finally, we mention that in the proof of Lemmas F.22, F.23, and F.25 we use concepts integration in measurable spaces. The reader is referred to (Lang, 1993; Rudin, 1974) for an insight on this concepts.

> We remark that, in general along this section, we assume that $\spadesuit$ terms belong to $\spadesuit^\alpha$ (i.e., terms are both α-definable and locally definable) except if specially noticed.

In several occasions we will also make use of the following properties.

Lemma F.17. *Let SA be a stochastic automaton and let s be a location of SA. Let v and v' be two valuations such that for all $x \in FV(s)$, $v(x) = v'(x)$. Then $(s, v) \sim_p (s, v')$ in both the open and close behaviour. If in addition $x \in \kappa(s)$, $v(x) = v'(x)$, then $[s, v] \sim_p [s, v']$.*

Proof. The proof that the relation

$$\begin{aligned} R \stackrel{\text{def}}{=} \quad & \{\langle (s,v),(s,v')\rangle \mid \forall x \in FV(s).\ v(x) = v'(x)\} \\ & \cup \{\langle [s,v],[s,v']\rangle \mid \forall x \in FV(s) \cup \kappa(s).\ v(x) = v'(x)\} \end{aligned}$$

is a probabilistic bisimulation is straightforward. *Q.E.D.*

The following corollary is immediate by transitivity of $\sim_p$.

Corollary F.17.1. *Let s_1 and s_2 be locations of SA. Let v_1, v_1', v_2, and v_2' be valuations such that $x \in FV(s_i)$, $v_i(x) = v_i'(x)$, $i = 1,2$. Then, $(s_1,v_1) \sim_p (s_2,v_2)$ implies that $(s_1,v_1') \sim_p (s_2,v_2')$. If in addition $x \in \kappa(s_i)$, $v_i(x) = v_i'(x)$, $i = 1,2$, then, $[s_1,v_1] \sim_p [s_2,v_2]$ implies that $[s_1,v_1'] \sim_p [s_2,v_2']$.*

Prefixing

Lemma F.18. *The following relation is a probabilistic bisimulation up to $\sim_p$.*

$$\begin{aligned} R^{\mathrm{p}} \stackrel{\text{def}}{=} & \{\langle (a;p_1,v),(a;p_2,v)\rangle \mid v \in \mathbf{V} \wedge p_1 \sim_o p_2\} \\ & \cup\ \{\langle [a;p_1,v],[a;p_2,v]\rangle \mid v \in \mathbf{V} \wedge p_1 \sim_o p_2\} \end{aligned}$$

Proof. Since $\sim_o$ is an equivalence, then R^{p} is symmetric.

1. Let $\langle (a;p_1,v),(a;p_2,v)\rangle \in R^{\mathrm{p}}$ and let $S \subseteq \Sigma'/(\sim_p \cup R^{\mathrm{p}})^*$. Notice that:

   (a) $T\,(a;p_i,v) = \mathcal{D}_v^{a;p_i}(\mathcal{R}())$, thus $\Omega_v^{a;p_i} = \{[a;p_i,v]\}$, for $i = 1,2$; and

   (b) $\langle [a;p_1,v],[a;p_2,v]\rangle \in R^{\mathrm{p}}$, which implies $[a;p_1,v] \in \bigcup S \iff [a;p_2,v] \in \bigcup S$.

   As a consequence, it is immediate that $\mu((a;p_1,v),\bigcup S) = \mu((a;p_2,v),\bigcup S)$.

2. Let $\langle [a;p_1,v],[a;p_2,v]\rangle \in R^{\mathrm{p}}$. Then, for $i = 1,2$,

$$\begin{array}{ll} a;p_i \xrightarrow{a,\varnothing} p_i & \text{(by Table 10.3)} \\ \Longrightarrow & \text{(by rule } \textbf{alpha}\text{)} \\ \quad a;p_i \xrightarrow{a,\varnothing}{}' p_i' \quad \wedge \quad p_i' \simeq_\alpha p_i \quad \wedge \quad \neg \mathsf{cv}(p_i') & \\ \Longrightarrow & \text{(by Def. 9.7, } d \in \mathbb{R}_{\geq 0}\text{)} \\ \quad [a;p_i,v] \xrightarrow{a(d)} (p_i,(v-d)) \quad \wedge \quad p_i' \simeq_\alpha p_i \quad \wedge \quad \neg \mathsf{cv}(p_i') & \end{array}$$

   But $p_1' \simeq_\alpha p_1 \sim_o p_2 \simeq_\alpha p_2'$, thus $p_1' \sim_o p_2'$. In particular, for valuation $v - d$ it holds that $\langle (p_1,(v-d)),(p_2,(v-d))\rangle \in \sim_p \subseteq (\sim_p \cup R^{\mathrm{p}})^*$. *Q.E.D.*

Triggering condition

Lemma F.19. *The following relation is a probabilistic bisimulation up to* $\sim_p$.

$$R^{\mathrm{t}} \stackrel{\text{def}}{=} \{\langle (C \mapsto p_1, v), (C \mapsto p_2, v)\rangle \mid (p_1, v) \sim_p (p_2, v)\} \cup \{\langle [C \mapsto p_1, v_1], [C \mapsto p_2, v_2]\rangle \mid [p_1, v_1] \sim_p [p_2, v_2] \ \wedge\ \forall x \in C.\ v_1(x) = v_2(x)\}$$

Proof. Since $\sim_p$ is an equivalence, then R^{t} is symmetric.

1. Let $\langle (C \mapsto p_1, v), (C \mapsto p_2, v)\rangle \in R^{\mathrm{t}}$ and let $S \subseteq \Sigma'/(\sim_p \cup R^{\mathrm{t}})^*$.
   Let $\overrightarrow{\kappa(C \mapsto p_i)} = \overrightarrow{\kappa(p_i)} = (x^i_1, \ldots, x^i_{n_i})$. Then

   $$T(C \mapsto p_i, v) = \mathcal{D}_v^{C \mapsto p_i}(\mathcal{R}(F_{x^i_1}, \ldots, F_{x^i_{n_i}})), \quad \text{and}$$
   $$T(p_i, v) = \mathcal{D}_v^{p_i}(\mathcal{R}(F_{x^i_1}, \ldots, F_{x^i_{n_i}}))$$

   Thus, $T(C \mapsto p_i, v)$ and $T(p_i, v)$ are isomorphic.
   Moreover, notice that, whenever $v_1(x) = v_2(x)$ for all $x \in C$,

   $$\langle [C \mapsto p_1, v_1], [C \mapsto p_2, v_2]\rangle \in R^{\mathrm{t}} \iff [p_1, v_1] \sim_p [p_2, v_2]$$

   As a consequence there is an $S' \subseteq \Sigma'/\sim_p$ such that

   $$(\mathcal{D}_v^{C \mapsto p_i})^{-1}((\bigcup S) \cap \Omega_v^{C \mapsto p_i}) \ = \ (\mathcal{D}_v^{p_i})^{-1}((\bigcup S') \cap \Omega_v^{p_i}),$$

   from which

   $$\begin{aligned} \mu((C \mapsto p_1, v), \bigcup S) &= \mu((p_1, v), \bigcup S') \\ &= \mu((p_2, v), \bigcup S') \\ &= \mu((C \mapsto p_2, v), \bigcup S) \end{aligned}$$

2. Let $\langle [C \mapsto p_1, v_1], [C \mapsto p_2, v_2]\rangle \in R^{\mathrm{t}}$. Then

   $$\begin{array}{ll} & [C \mapsto p_1, v_1] \xrightarrow{a(d)} (p'_1, (v_1 - d)) \\ \Longrightarrow & \text{(by Def. 9.7, rule \textbf{Open})} \\ & C \mapsto p_1 \xrightarrow{a, C \cup C_1}{}' p'_1 \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C \cup C_1.\ (v_1 - d)(x) \leq 0 \\ \Longrightarrow & \text{(by rule \textbf{alpha})} \\ & C \mapsto p_1 \xrightarrow{a, C \cup C_1} p''_1 \quad \wedge \quad p''_1 \simeq_\alpha p'_1 \quad \wedge \quad \neg \mathsf{cv}(p'_1) \\ & \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C \cup C_1.\ (v_1 - d)(x) \leq 0 \\ \Longrightarrow & \text{(by Table 10.3)} \\ & p_1 \xrightarrow{a, C_1} p''_1 \quad \wedge \quad p''_1 \simeq_\alpha p'_1 \quad \wedge \quad \neg \mathsf{cv}(p'_1) \\ & \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C \cup C_1.\ (v_1 - d)(x) \leq 0 \\ \Longrightarrow & \text{(by rule \textbf{alpha} and calculations)} \end{array}$$

$$
\begin{aligned}
& p_1 \xrightarrow{a,C_1}{}' p_1' \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 \\
& \wedge \quad \forall x \in C.\ (v_1 - d)(x) \leq 0 \\
\Longrightarrow & \hfill \text{(by Def. 9.7, rule } \textbf{Open}\text{)} \\
& [p_1, v_1] \xrightarrow{a(d)} (p_1', (v_1 - d)) \quad \wedge \quad \forall x \in C.\ (v_1 - d)(x) \leq 0 \\
\Longrightarrow & \hfill \text{(since } [p_1, v_1] \sim_p [p_2, v_2] \text{ and } \forall x \in C.\ v_1(x) = v_2(x) \text{, by def. of } R^{\mathrm{t}}\text{)} \\
& [p_2, v_2] \xrightarrow{a(d)} (p_2', (v_2 - d)) \quad \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d)) \\
& \wedge \quad \forall x \in C.\ (v_2 - d)(x) \leq 0 \\
\Longrightarrow & \hfill \text{(by Def. 9.7, rule } \textbf{Open}\text{)} \\
& p_2 \xrightarrow{a,C_2}{}' p_2' \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge \quad \forall x \in C.\ (v_2 - d)(x) \leq 0 \quad \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d)) \\
\Longrightarrow & \hfill \text{(by rule } \textbf{alpha} \text{ and calculations)} \\
& p_2 \xrightarrow{a,C_2} p_2'' \quad \wedge \quad p_2'' \simeq_\alpha p_2' \quad \wedge \quad \neg\mathsf{cv}(p_2') \\
& \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C \cup C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d)) \\
\Longrightarrow & \hfill \text{(by Table 10.3)} \\
& C \mapsto p_2 \xrightarrow{a,C \cup C_2} p_2'' \quad \wedge \quad p_2'' \simeq_\alpha p_2' \quad \wedge \quad \neg\mathsf{cv}(p_2') \\
& \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C \cup C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d)) \\
\Longrightarrow & \hfill \text{(by rule } \textbf{alpha}\text{)} \\
& C \mapsto p_2 \xrightarrow{a,C \cup C_2}{}' p_2' \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C \cup C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d)) \\
\Longrightarrow & \hfill \text{(by Def. 9.7, rule } \textbf{Open}\text{, and } \sim_p \subseteq (\sim_p \cup R^{\mathrm{t}})^*\text{)} \\
& [C \mapsto p_2, v_2] \xrightarrow{a(d)} (p_2', (v_2 - d)) \\
& \wedge \quad \langle (p_1', (v_1 - d)), (p_2', (v_2 - d)) \rangle \in (\sim_p \cup R^{\mathrm{t}})^*
\end{aligned}
$$

Q.E.D.

Clock setting

The proof of the following lemma is straightforward. It follows the same strategy as the second transfer property in the proof of Lemma F.19, but it is simpler.

Lemma F.20. *For every term* $p \in \spadesuit^{\mathsf{cf}}$ *and valuation* $v \in \mathbf{V}$, $[\{\!|C|\!\}\, p, v] \sim_p [p, v]$.

Lemma F.21. *The following relation is a probabilistic bisimulation up to* $\sim_p$.

$$R^{\mathrm{cs}} \stackrel{\mathrm{def}}{=} \{\langle (\{\!|x|\!\}\, p_1, v), (\{\!|x|\!\}\, p_2, v) \rangle \mid x \notin \kappa(p_1) \cup \kappa(p_2) \ \wedge \ p_1 \sim_o p_2\}$$

(Recall that $p_1 \sim_o p_2$ iff for all $v \in \mathbf{V}$, $(p_1, v) \sim_p (p_2, v)$.)

Proof. Since $\sim_o$ is an equivalence, then R^{cs} is symmetric.

1. Let $\langle (\{\!|x|\!\}\, p_1, v), (\{\!|x|\!\}\, p_2, v)\rangle \in R^{\text{cs}}$ and let $S \subseteq \Sigma'/(\sim_p \cup R^{\text{cs}})^*$.
   Let $\overrightarrow{\kappa(\{\!|x|\!\}\, p_i)} = \overrightarrow{\{x\} \cup \kappa(p_i)} = (x, x^i_1, \ldots, x^i_{n_i})$. Then

   $$T(\{\!|x|\!\}\, p_i, v) = \mathcal{D}_v^{\{\!|x|\!\}\, p_i}(\mathcal{R}(F_x, F_{x^i_1}, \ldots, F_{x^i_{n_i}})).$$

   We assume P_{n_i+1} is the probability measure in $\mathcal{R}(F_x, F_{x^i_1}, \ldots, F_{x^i_{n_i}})$, and P_{n_i} is the probability measure in $\mathcal{R}(F_{x^i_1}, \ldots, F_{x^i_{n_i}})$
   We have to prove that

   $$\mu((\{\!|x|\!\}\, p_1, v), \bigcup S) \;=\; \mu((\{\!|x|\!\}\, p_2, v), \bigcup S)$$

   In the following, we write $B(\vec{d})$ to refer to the indicator function $B(\vec{d}) = \textbf{if } \vec{d} \in B \textbf{ then } 1 \textbf{ else } 0$. We calculate,

   $$\begin{aligned}
   &\mu((\{\!|x|\!\}\, p_i, v), \bigcup S)\\
   &\quad= P_v^{\{\!|x|\!\}\, p_i}\big((\bigcup S) \cap \Omega_v^{\{\!|x|\!\}\, p_i}\big)\\
   &\quad= P_{n_i+1}\Big(\big(\mathcal{D}_v^{\{\!|x|\!\}\, p_i}\big)^{-1}\big((\bigcup S) \cap \Omega_v^{\{\!|x|\!\}\, p_i}\big)\Big)\\
   &\quad= P_{n_i+1}(B_i) \qquad\qquad (\text{with } B_i = \big(\mathcal{D}_v^{\{\!|x|\!\}\, p_i}\big)^{-1}\big((\bigcup S) \cap \Omega_v^{\{\!|x|\!\}\, p_i}\big))\\
   &\quad= \int_{\mathcal{B}(\mathbb{R}^{n_i+1})} B_i \,\mathbf{d}P_{n_i+1}\\
   &\quad= \int_{\mathcal{B}(\mathbb{R}^{n_i+1})} B_i(d, \vec{d_i}) \,\mathbf{d}P_{n_i+1}(d, \vec{d_i})\\
   &\quad= \int_{\mathcal{B}(\mathbb{R})} \left(\int_{\mathcal{B}(\mathbb{R}^{n_i})} B_i(d, \vec{d_i}) \,\mathbf{d}P_{n_i}(\vec{d_i})\right) \mathbf{d}P(d) \qquad \text{(F.6)}
   \end{aligned}$$

   Let $B_x = \{d \mid \exists i \in \{1, 2\}.\ \ (d, d^i_1, \ldots, d^i_{n_i}) \in B_i\}$. From (F.6), it suffices to prove that for all $d \in B_x$,

   $$\int_{\mathcal{B}(\mathbb{R}^{n_1})} B_1(d, \vec{d_1}) \,\mathbf{d}P_{n_i}(\vec{d_1}) \;=\; \int_{\mathcal{B}(\mathbb{R}^{n_2})} B_2(d, \vec{d_2}) \,\mathbf{d}P_{n_i}(\vec{d_2}) \qquad \text{(F.7)}$$

   Notice that some of this integrals may be 0.

   Recall that $p_1 \sim_o p_2$, i.e., for all $v^\star$, $(p_1, v^\star) \sim_p (p_2, v^\star)$. As a consequence, the proof of (F.7) reduces to prove that for all $d \in B_x$, there is an $S_d \subseteq \Sigma'/\sim_p$, such that for all $i = 1, 2$,

   $$\mu((p_i, v_d), \bigcup S_d) \;=\; \int_{\mathcal{B}(\mathbb{R}^{n_i})} B_i(d, \vec{d_i}) \,\mathbf{d}P_{n_i}(\vec{d_i}) \qquad \text{(F.8)}$$

where $v_d(x) = d$, and $v_d(y) = v(y)$ otherwise. We can do so since $(p_1, v_d) \sim_p (p_2, v_d)$ implies $\mu((p_1, v_d), \bigcup S_d) = \mu((p_2, v_d), \bigcup S_d)$.

But

$$\begin{aligned}
\mu((p_i, v_d), \bigcup S_d) &= P_{v_d}^{p_i}\big((\bigcup S_d) \cap \Omega_{v_d}^{p_1}\big) && \text{(by def. of } \mu) \\
&= P_{n_i}\Big(\big(\mathcal{D}_{v_d}^{p_i}\big)^{-1}\big((\bigcup S_d) \cap \Omega_{v_d}^{p_i}\big)\Big) && \text{(by def. of } P_{v_d}^{p_i}) \\
&= \int_{\mathcal{B}(\mathbb{R}^{n_i})} \big(\mathcal{D}_{v_d}^{p_i}\big)^{-1}\big((\bigcup S_d) \cap \Omega_{v_d}^{p_i}\big)\ \mathbf{d}P_{n_i}
\end{aligned}$$

Therefore, we have to prove that

$$\int_{\mathcal{B}(\mathbb{R}^{n_i})} \big(\mathcal{D}_{v_d}^{p_i}\big)^{-1}\big((\bigcup S_d) \cap \Omega_{v_d}^{p_i}\big)\ \mathbf{d}P_{n_i} \;=\; \int_{\mathcal{B}(\mathbb{R}^{n_i})} B_i(d, \vec{d_i})\ \mathbf{d}P_{n_i}(\vec{d_i})$$

For all $d \in B_x$, define

$$B_i^d = \{(d_1^i, \ldots, d_{n_i}^i) \mid (d, d_1^i, \ldots, d_{n_i}^i) \in B_i\}.$$

Then, it suffices to prove that for all $i = 1, 2$

$$B_i^d \;=\; \big(\mathcal{D}_{v_d}^{p_i}\big)^{-1}\big((\bigcup S_d) \cap \Omega_{v_d}^{p_i}\big) \tag{F.9}$$

So, choose S_d to be the minimal set satisfying

$$\left.\begin{array}{l} (v_i(x_1^i), \ldots, v_i(x_{n_i}^i)) \in B_i^d \\ \wedge \quad \forall y \in \mathcal{C} - \kappa(p_i).\ v_i(y) = v_d(y) \end{array}\right\} \implies [p_i, v_i] \in (\bigcup S_d) \cap \Omega_{v_d}^{p_i}$$

for both $i = 1, 2$. Clearly

$$B_i^d \;\subseteq\; \big(\mathcal{D}_{v_d}^{p_i}\big)^{-1}\big((\bigcup S_d) \cap \Omega_{v_d}^{p_i}\big)$$

To prove the other inclusion, we proceed as follows. We only prove case $i = 1$, the other follows in a similar manner. Assume

$$[p_1, u] \in (\bigcup S_d) \cap \Omega_{v_d}^{p_1} \quad \text{such that} \quad (u(x_1^1), \ldots, u(x_{n_1}^1)) \notin B_1^d$$

Since S_d is minimal, one of the following statements must hold:

(a) there is a $[p_1, v_1] \in (\bigcup S_d) \cap \Omega_{v_d}^{p_1}$ such that

$$(v_1(x_1^1), \ldots, v_1(x_{n_1}^1)) \in B_1^d \quad \text{and} \quad [p_1, u] \sim_p [p_1, v_1]\,; \text{ or}$$

(b) there is a $[p_2, v_2] \in (\bigcup S_d) \cap \Omega_{v_d}^{p_2}$ such that

$$(v_2(x_1^2), \ldots, v_2(x_{n_2}^2)) \in B_2^d \quad \text{and} \quad [p_1, u] \sim_p [p_2, v_2]\,.$$

Suppose that option (a) holds. By Lemma F.20

$$[\{\!|x|\!\}\, p_1, u] \sim_p [p_1, u] \sim_p [p_1, v_1] \sim_p [\{\!|x|\!\}\, p_1, v_1]$$

which implies

$$\{[\{\!|x|\!\}\, p_1, u], [\{\!|x|\!\}\, p_1, v_1]\} \subseteq (\bigcup S) \cap \Omega_v^{\{\!|x|\!\}\, p_1}$$

As a consequence

$$(u(x), u(x_1^1), \ldots, u(x_{n_1}^1)) \in B_1$$

that is to say,

$$(u(x_1^1), \ldots, u(x_{n_1}^1)) \in B_1^d$$

which contradicts our assumption. So it should be the case that option (b) holds. We proceed in a similar way. By Lemma F.20

$$[\{\!|x|\!\}\, p_1, u] \sim_p [p_1, u] \sim_p [p_2, v_2] \sim_p [\{\!|x|\!\}\, p_2, v_2]$$

which implies

$$[\{\!|x|\!\}\, p_1, u] \in (\bigcup S) \cap \Omega_v^{\{\!|x|\!\}\, p_1} \quad \text{and} \quad [\{\!|x|\!\}\, p_2, v_2] \in (\bigcup S) \cap \Omega_v^{\{\!|x|\!\}\, p_2}.$$

As a consequence

$$(u(x), u(x_1^1), \ldots, u(x_{n_1}^1)) \in B_1$$

that is to say,

$$(u(x_1^1), \ldots, u(x_{n_1}^1)) \in B_1^d$$

which again contradicts our assumption. The equality in (F.9) is finally proved, and therefore this transfer property.

2. Notice that R^{cs} does not contain any pair of non-deterministic states (i.e., pairs with the shape $\langle [q_1, v_1], [q_2, v_2] \rangle$). That makes trivial the proof of the second transference property.

   The reader might be surprised about this. To take him/her out of this "proved-by-magic" feeling, we remark that a natural way to enlarge R^{cs} to contain also pairs of non-deterministic states would be to choose instead $R^{\text{ext}} \stackrel{\text{def}}{=} R^{\text{cs}} \cup R^{\text{add}}$, where

   $$R^{\text{add}} \stackrel{\text{def}}{=} \{\langle [\{\!|x|\!\}\, p_1, v_1], [\{\!|x|\!\}\, p_2, v_2] \rangle \mid [p_1, v_1] \sim_p [p_2, v_2]\}.$$

   However, by Lemma F.20, $[\{\!|x|\!\}\, p_1, v_1] \sim_p [p_1, v_1] \sim_p [p_2, v_2] \sim_p [\{\!|x|\!\}\, p_2, v_2]$, and as a consequence, $R^{\text{add}} \subseteq \sim_p$ which would prove anyway this transfer property. *Q.E.D.*

Lemma F.22. *Let p_1 and p_2 be two $\mathbb{O}$ terms whose behaviours are α-definable. Assume $p_1 \sim_o p_1$. Moreover, suppose that the behaviours of $\{\!|C|\!\}\, p_1$ and $\{\!|C|\!\}\, p_2$ are α-definable as well. Then $\{\!|C|\!\}\, p_1 \sim_o \{\!|C|\!\}\, p_2$.*

Proof. Notice that,

1. for each $i = 1, 2$, there exists p_i' such that its behaviour is α-definable, $\kappa(p_i') \cap C = \varnothing$, and $p_i \simeq_\alpha p_i'$; and
2. for all $p \in \mathbb{O}^\alpha$, $\{\!|x_1, x_2, \ldots, x_n|\!\}\, p = \{\!|x_1|\!\}\, \{\!|x_2|\!\} \cdots \{\!|x_n|\!\}\, p_i$ (see axiom **CS2** in Table 11.1).

Since $\simeq_\alpha$ is a congruence, by 1 it suffices to prove that $\{\!|C|\!\}\, p_1' \sim_o \{\!|C|\!\}\, p_2'$. Because of 2, this can be achieve by applying induction on the number of clocks in C, using Lemma F.21 in the inductive case. *Q.E.D.*

Summation

Lemma F.23. *The following relations are probabilistic bisimulations up to $\sim_p$.*

$$\begin{aligned} R_l^{\mathrm{s}} \stackrel{\text{def}}{=} & \{\langle (p_1 + q, v), (p_2 + q, v)\rangle \mid p_1 \sim_o p_2\} \\ & \cup \{\langle [p_1 + q, v_1], [p_2 + q, v_2]\rangle \mid p_1 \sim_o p_2 \\ & \qquad \wedge\ \forall x \in \mathcal{C} - (\kappa(p_1) \cup \kappa(p_2)).\ v_1(x) = v_2(x) \\ & \qquad \wedge\ \exists v, v_1', v_2'.\ \forall i = 1, 2.\ (\ [p_i, v_i'] \in \Omega_v^{p_i}\ \wedge\ [p_1, v_1'] \sim_p [p_2, v_2'] \\ & \qquad\qquad \wedge\ \forall x \in \kappa(p_i) \cup fv(p_i).\ v_i(x) = v_i'(x)\ )\ \} \end{aligned}$$

$$\begin{aligned} R_r^{\mathrm{s}} \stackrel{\text{def}}{=} & \{\langle (q + p_1, v), (q + p_2, v)\rangle \mid p_1 \sim_o p_2\} \\ & \cup \{\langle [q + p_1, v_1], [q + p_2, v_2]\rangle \mid p_1 \sim_o p_2 \\ & \qquad \wedge\ \forall x \in \mathcal{C} - (\kappa(p_1) \cup \kappa(p_2)).\ v_1(x) = v_2(x) \\ & \qquad \wedge\ \exists v, v_1', v_2'.\ \forall i = 1, 2.\ (\ [p_i, v_i'] \in \Omega_v^{p_i}\ \wedge\ [p_1, v_1'] \sim_p [p_2, v_2'] \\ & \qquad\qquad \wedge\ \forall x \in \kappa(p_i) \cup fv(p_i).\ v_i(x) = v_i'(x)\ )\ \} \end{aligned}$$

Proof. We only prove that R_l^{s} is a probabilistic bisimulation up to $\sim_p$. The proof for R_r^{s} follows exactly in the same way. We remark that R_l^{s} is symmetric because $\sim_o$, $\sim_p$ and $=$ are equivalence relations and hence, also symmetric.

1. Let $\langle (p_1 + q, v), (p_2 + q, v)\rangle \in R_l^{\mathrm{s}}$ and let $S \subseteq \Sigma'/(\sim_p \cup R_l^{\mathrm{s}})^*$.
   Let $\overrightarrow{\kappa(p_i + q)} = \overrightarrow{\kappa(p_i) \cup \kappa(q)} = (x_1^i, \ldots, x_{n_i}^i, y_1, \ldots, y_m)$[2]. Then

$$T\,(p_i + q, v) = \mathcal{D}_v^{p_i + q}(\mathcal{R}(F_{x_1^i}, \ldots, F_{x_{n_i}^i}, F_{y_1}, \ldots, F_{y_m})).$$

[2]Recall that $\kappa(p_i) \cap \kappa(q) = \kappa(p_i) \cap fv(q) = fv(p_i) \cap \kappa(q) = \varnothing$ since we assume $p_i + q$ is locally definable and hence locally free of conflict.

We prove that

$$\mu((p_1+q,v),\bigcup S) \;=\; \mu((p_2+q,v),\bigcup S) \tag{F.10}$$

For $i=1,2$, we calculate,

$$\begin{aligned}
&\mu((p_i+q,v),\bigcup S)\\
&\quad= P_v^{p_i+q}\big((\bigcup S)\cap\Omega_v^{p_i+q}\big) && \text{(by def. of } \mu)\\
&\quad= P_{n_i+m}\Big(\big(\mathcal{D}_v^{p_i+q}\big)^{-1}\big((\bigcup S)\cap\Omega_v^{p_i+q}\big)\Big) && \text{(by def. of } P_v^{p_i+q})\\
&\quad= P_{n_i+m}(B_i) && \text{(with } B_i=\big(\mathcal{D}_v^{p_i+q}\big)^{-1}\big((\bigcup S)\cap\Omega_v^{p_i+q}\big))\\
&\quad= \int_{\mathcal{B}(\mathbb{R}^{n_i+m})} B_i\,\mathbf{d}P_{n_i+m}\\
&\quad= \int_{\mathcal{B}(\mathbb{R}^{n_i+m})} B_i(\vec{d_i},\vec{d})\,\mathbf{d}P_{n_i+1}(\vec{d_i},\vec{d})\\
&\quad= \int_{\mathcal{B}(\mathbb{R}^{m})}\left(\int_{\mathcal{B}(\mathbb{R}^{n_i})} B_i(\vec{d_i},\vec{d})\,\mathbf{d}P_{n_i}(\vec{d_i})\right)\,\mathbf{d}P_m(\vec{d})
\end{aligned}\tag{F.11}$$

Let $B_{\vec{y}} = \{(d_1,\ldots,d_m)\mid \exists i\in\{1,2\}.\ (d_1^i,\ldots,d_{n_i}^i,d_1,\ldots,d_m)\in B_i\}$. From (F.11), it suffices to prove that for all $\vec{d}\in B_{\vec{y}}$,

$$\int_{\mathcal{B}(\mathbb{R}^{n_1})} B_1(\vec{d_1},\vec{d})\,\mathbf{d}P_{n_1}(\vec{d_1}) \;=\; \int_{\mathcal{B}(\mathbb{R}^{n_2})} B_2(\vec{d_2},\vec{d})\,\mathbf{d}P_{n_2}(\vec{d_2}) \tag{F.12}$$

Notice that some of this integrals may be 0.

Recall that $p_1\sim_o p_2$, i.e., for all $v^\star$, $(p_1,v^\star)\sim_p(p_2,v^\star)$. As a consequence, the proof of (F.12) reduces to prove that for all $\vec{d}\in B_{\vec{y}}$, there is an $S_{\vec{d}}\subseteq\Sigma'/\sim_p$, such that for all $i=1,2$,

$$\mu(\big(p_i,v_{\vec{d}}\big),\bigcup S_{\vec{d}}) \;=\; \int_{\mathcal{B}(\mathbb{R}^{n_i})} B_i(\vec{d_i},\vec{d})\,\mathbf{d}P_{n_i}(\vec{d_i}) \tag{F.13}$$

where $v_{\vec{d}}(y_j)=d_j$ with $j\in\{1,\ldots,m\}$, and $v_{\vec{d}}(x)=v(x)$ otherwise. We can do so since $\big(p_1,v_{\vec{d}}\big)\sim_p\big(p_2,v_{\vec{d}}\big)$ implies $\mu(\big(p_1,v_{\vec{d}}\big),\bigcup S_{\vec{d}})=\mu(\big(p_2,v_{\vec{d}}\big),\bigcup S_{\vec{d}})$.

But

$$\begin{aligned}
\mu(\big(p_i,v_{\vec{d}}\big),\bigcup S_{\vec{d}}) \;&=\; P_{v_{\vec{d}}}^{p_i}\big((\bigcup S_{\vec{d}})\cap\Omega_{v_{\vec{d}}}^{p_1}\big) && \text{(by def. of } \mu)\\
&=\; P_{n_i}\Big(\big(\mathcal{D}_{v_{\vec{d}}}^{p_i}\big)^{-1}\big((\bigcup S_{\vec{d}})\cap\Omega_{v_{\vec{d}}}^{p_i}\big)\Big) && \text{(by def. of } P_{v_{\vec{d}}}^{p_i})
\end{aligned}$$

For all $\vec{d}\in B_{\vec{y}}$, define

$$B_i^{\vec{d}} = \{(d_1^i,\ldots,d_{n_i}^i)\mid \vec{d}=(d_1,\ldots,d_m)\wedge(d_1^i,\ldots,d_{n_i}^i,d_1,\ldots,d_m)\in B_i\}.$$

Then, we only have to prove that for all $i = 1, 2$

$$B_i^{\vec{d}} \;=\; \left(\mathcal{D}_{v_{\vec{d}}}^{p_i}\right)^{-1}\left(\left(\bigcup S_{\vec{d}}\right) \cap \Omega_{v_{\vec{d}}}^{p_i}\right) \tag{F.14}$$

So, choose $S_{\vec{d}}$ to be the minimal set satisfying

$$\left.\begin{array}{l} (v_i(x_1^i), \ldots, v_i(x_{n_i}^i)) \in B_i^{\vec{d}} \\ \wedge \quad \forall x \in \mathcal{C} - \kappa(p_i).\ v_i(x) = v_{\vec{d}}(x) \end{array}\right\} \implies [p_i, v_i] \in \left(\bigcup S_{\vec{d}}\right) \cap \Omega_{v_{\vec{d}}}^{p_i}$$

for both $i = 1, 2$. Clearly

$$B_i^{\vec{d}} \;\subseteq\; \left(\mathcal{D}_{v_{\vec{d}}}^{p_i}\right)^{-1}\left(\left(\bigcup S_{\vec{d}}\right) \cap \Omega_{v_{\vec{d}}}^{p_i}\right)$$

To prove the other inclusion, we proceed as follows. We only prove case $i = 1$, the other follows in a similar manner. Assume

$$[p_1, u] \in \left(\bigcup S_{\vec{d}}\right) \cap \Omega_{v_{\vec{d}}}^{p_1} \quad \text{such that} \quad (u(x_1^1), \ldots, u(x_{n_1}^1)) \notin B_1^{\vec{d}}$$

Since $S_{\vec{d}}$ is minimal, one of the following statements must hold:

(a) there is a $[p_1, v_1] \in \left(\bigcup S_{\vec{d}}\right) \cap \Omega_{v_{\vec{d}}}^{p_1}$ such that

$$(v_1(x_1^1), \ldots, v_1(x_{n_1}^1)) \in B_1^{\vec{d}} \quad \text{and} \quad [p_1, u] \sim_p [p_1, v_1]\,;\ \text{or}$$

(b) there is a $[p_2, v_2] \in \left(\bigcup S_{\vec{d}}\right) \cap \Omega_{v_{\vec{d}}}^{p_2}$ such that

$$(v_2(x_1^2), \ldots, v_2(x_{n_2}^2)) \in B_2^{\vec{d}} \quad \text{and} \quad [p_1, u] \sim_p [p_2, v_2]\,.$$

Suppose that option (a) holds. Notice that for all $x \in \mathcal{C} - \kappa(p_1)$, $v_1(x) = u(x)$. Then, notice that

$$\langle [p_1 + q, u], [p_1 + q, v_1] \rangle \in R_l^{\mathrm{s}}$$

from which it is immediate that

$$\{[p_1 + q, u], [p_1 + q, v_1]\} \subseteq \left(\bigcup S\right) \cap \Omega_v^{p_1+q}.$$

As a consequence

$$(u(x_1^1), \ldots, u(x_{n_1}^1), u(y_1), \ldots, u(y_m)) \in B_1$$

that is to say,

$$(u(x_1^1), \ldots, u(x_{n_1}^1)) \in B_1^{\vec{d}}$$

which contradicts our assumption. So it should be the case that option (b) holds. We proceed in a similar way. Notice that for all $x \in \mathcal{C} - (\kappa(p_1) \cup \kappa(p_2))$, $v_2(x) = u(x)$. Then, notice that

$$\langle [p_1 + q, u], [p_2 + q, v_2] \rangle \in R_l^{\mathrm{s}}$$

from which it is immediate that

$$[p_1 + q, u] \in (\bigcup S) \cap \Omega_v^{p_1+q} \quad \text{and} \quad [p_2 + q, v_2] \in (\bigcup S) \cap \Omega_v^{p_2+q}.$$

As a consequence

$$(u(x_1^1), \ldots, u(x_{n_1}^1), u(y_1), \ldots, u(y_m)) \in B_1$$

that is to say,

$$(u(x_1^1), \ldots, u(x_{n_1}^1)) \in B_1^{\vec{d}}$$

which again contradicts our assumption. The equality in (F.14) is finally proved, and therefore this transfer property.

2. Let $\langle [p_1 + q, v_1], [p_2 + q, v_2] \rangle \in R_l^{\mathrm{s}}$ and let v_1' and v_2' as in the definition of R_l^{s}. Then

$$\begin{array}{l}
[p_1 + q, v_1] \xrightarrow{a(d)} (p_1', (v_1 - d)) \\
\Longrightarrow \qquad \text{(by Def. 9.7, rule } \textbf{Open}\text{)} \\
\quad p_1 + q \xrightarrow{a,C_1}{}' p_1' \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 \\
\Longrightarrow \qquad \text{(by rule } \textbf{alpha}\text{)} \\
\quad p_1 + q \xrightarrow{a,C_1} p_1'' \quad \wedge \quad p_1'' \simeq_\alpha p_1' \quad \wedge \quad \neg \mathsf{cv}(p_1') \\
\quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 \\
\Longrightarrow \qquad \text{(by Table 10.3)} \\
\quad (p_1 \xrightarrow{a,C_1} p_1'' \ \vee \ q \xrightarrow{a,C_1} p_1'') \quad \wedge \quad p_1'' \simeq_\alpha p_1' \quad \wedge \quad \neg \mathsf{cv}(p_1') \\
\quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 \\
\Longrightarrow \qquad \text{(by rule } \textbf{alpha}\text{)} \\
\quad (p_1 \xrightarrow{a,C_1}{}' p_1' \ \vee \ q \xrightarrow{a,C_1}{}' p_1') \\
\quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 \\
\Longrightarrow \qquad \text{(by Def. 9.7, rule } \textbf{Open}\text{)} \\
\quad [p_1, v_1] \xrightarrow{a(d)} (p_1', (v_1 - d)) \quad \vee \quad [q, v_1] \xrightarrow{a(d)} (p_1', (v_1 - d)) \\
\Longrightarrow \qquad \text{(by Lemma F.17 and Cor. F.17.1)} \\
\quad ([p_2, v_2] \xrightarrow{a(d)} (p_2', (v_2 - d)) \quad \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d))) \\
\quad \vee \quad ([q, v_2] \xrightarrow{a(d)} (p_2', (v_2 - d)) \quad \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d))) \\
\Longrightarrow
\end{array}$$

$$
\begin{array}{ll}
& ([p_2, v_2] \xrightarrow{a(d)} (p_2', (v_2 - d)) \ \vee \ [q, v_2] \xrightarrow{a(d)} (p_2', (v_2 - d))) \\
& \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d)) \\
\Longrightarrow & \qquad \text{(by Def. 9.7, rule } \textbf{Open}\text{)} \\
& (p_2 \xrightarrow{a,C_2}{}' p_2' \ \vee \ q \xrightarrow{a,C_2}{}' p_2') \\
& \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d)) \\
\Longrightarrow & \qquad \text{(by rule } \textbf{alpha}\text{)} \\
& (p_2 \xrightarrow{a,C_2} p_2'' \ \vee \ q \xrightarrow{a,C_2} p_2'') \quad \wedge \quad p_2'' \simeq_\alpha p_2' \quad \wedge \quad \neg \mathsf{cv}(p_2') \\
& \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d)) \\
\Longrightarrow & \qquad \text{(by Table 10.3)} \\
& p_2 + q \xrightarrow{a,C_2} p_2'' \quad \wedge \quad p_2'' \simeq_\alpha p_2' \quad \wedge \quad \neg \mathsf{cv}(p_2') \\
& \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d)) \\
\Longrightarrow & \qquad \text{(by rule } \textbf{alpha}\text{)} \\
& p_2 + q \xrightarrow{a,C_2}{}' p_2' \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge \quad (p_1', (v_1 - d)) \sim_p (p_2', (v_2 - d)) \\
\Longrightarrow & \qquad \text{(by Def. 9.7, rule } \textbf{Open}\text{, and } \sim_p \subseteq (\sim_p \cup R^{\mathrm{t}})^*\text{)} \\
& [p_2 + q, v_2] \xrightarrow{a(d)} (p_2', (v_2 - d)) \\
& \wedge \quad \langle (p_1', (v_1 - d)), (p_2', (v_2 - d)) \rangle \in (\sim_p \cup R^{\mathrm{t}})^*
\end{array}
$$

Q.E.D.

Parallel compositions

The proof of the following lemma is straightforward.

Lemma F.24. *Let $[p_1, v_1] \sim_p [p_2, v_2]$. For all $d \in \mathbb{R}_{\geq 0}$ it holds that*

$$\left[\overline{\mathsf{ck}}(p_1), (v_1 - d)\right] \sim_p \left[\overline{\mathsf{ck}}(p_2), (v_2 - d)\right] \quad \textit{and} \quad \left(\overline{\mathsf{ck}}(p_1), (v_1 - d)\right) \sim_p \left(\overline{\mathsf{ck}}(p_2), (v_2 - d)\right)$$

Lemma F.25. *The following relations are probabilistic bisimulations up to $\sim_p$*

$$
\begin{array}{rl}
R_l^{\mathrm{m}} \stackrel{\mathrm{def}}{=} \big\{ \langle (p_1 \,||_A\, q, v_1), (p_2 \,||_A\, q, v_2) \rangle \mid \exists v_1', v_2' \in \mathbf{V}.\ & (p_1, v_1') \sim_p (p_2, v_2') \\
& \wedge\ \forall x \in fv(q).\ v_1(x) = v_2(x) \\
& \wedge\ \forall i = 1, 2.\ \forall x \in fv(p_i).\ v_i(x) = v_i'(x) \ \big\}
\end{array}
$$

$$
\begin{aligned}
&\cup \left\{ \left\langle [p_1 \mid\mid_A q, v_1], [p_2 \mid\mid_A q, v_2] \right\rangle \mid \exists v_1', v_2' \in \mathbf{V}.\ \ [p_1, v_1'] \sim_p [p_2, v_2'] \right. \\
&\qquad\qquad \wedge\ \forall x \in \kappa fv(q).\ v_1(x) = v_2(x) \\
&\qquad\qquad \left. \wedge\ \forall i = 1,2.\ \forall x \in \kappa fv(p_i).\ v_i(x) = v_i'(x) \right\} \\
&\cup \left\{ \left\langle \left(\overline{\mathsf{ck}}(p_1) \mid\mid_A q, v_1\right), \left(\overline{\mathsf{ck}}(p_2) \mid\mid_A q, v_2\right) \right\rangle \mid \right. \\
&\qquad\qquad \exists v_1', v_2' \in \mathbf{V}.\ \ \left(\overline{\mathsf{ck}}(p_1), v_1'\right) \sim_p \left(\overline{\mathsf{ck}}(p_2), v_2'\right) \\
&\qquad\qquad \wedge\ \forall x \in fv(q).\ v_1(x) = v_2(x) \\
&\qquad\qquad \left. \wedge\ \exists d \in \mathbb{R}_{\geq 0}.\ \forall i = 1,2.\ \forall x \in \kappa fv(p_i).\ v_i(x) = (v_i' - d)(x) \right\} \\
&\cup \left\{ \left\langle \left[\overline{\mathsf{ck}}(p_1) \mid\mid_A q, v_1\right], \left[\overline{\mathsf{ck}}(p_2) \mid\mid_A q, v_2\right] \right\rangle \mid \right. \\
&\qquad\qquad \exists v_1', v_2' \in \mathbf{V}.\ \ \left[\overline{\mathsf{ck}}(p_1), v_1'\right] \sim_p \left[\overline{\mathsf{ck}}(p_2), v_2'\right] \\
&\qquad\qquad \wedge\ \forall x \in \kappa fv(q).\ v_1(x) = v_2(x) \\
&\qquad\qquad \left. \wedge\ \exists d \in \mathbb{R}_{\geq 0}.\ \forall i = 1,2.\ \forall x \in \kappa fv(p_i).\ v_i(x) = (v_i' - d)(x) \right\}
\end{aligned}
$$

$$
\begin{aligned}
R_r^{\mathrm{m}} \stackrel{\mathrm{def}}{=} &\left\{ \left\langle (q \mid\mid_A p_1, v_1), (q \mid\mid_A p_2, v_2) \right\rangle \mid \exists v_1', v_2' \in \mathbf{V}.\ \ (p_1, v_1') \sim_p (p_2, v_2') \right. \\
&\qquad\qquad \wedge\ \forall x \in fv(q).\ v_1(x) = v_2(x) \\
&\qquad\qquad \left. \wedge\ \forall i = 1,2.\ \forall x \in fv(p_i).\ v_i(x) = v_i'(x) \right\} \\
&\cup \left\{ \left\langle [q \mid\mid_A p_1, v_1], [q \mid\mid_A p_2, v_2] \right\rangle \mid \exists v_1', v_2' \in \mathbf{V}.\ \ [p_1, v_1'] \sim_p [p_2, v_2'] \right. \\
&\qquad\qquad \wedge\ \forall x \in \kappa fv(q).\ v_1(x) = v_2(x) \\
&\qquad\qquad \left. \wedge\ \forall i = 1,2.\ \forall x \in \kappa fv(p_i).\ v_i(x) = v_i'(x) \right\} \\
&\cup \left\{ \left\langle \left(q \mid\mid_A \overline{\mathsf{ck}}(p_1), v_1\right), \left(q \mid\mid_A \overline{\mathsf{ck}}(p_2), v_2\right) \right\rangle \mid \right. \\
&\qquad\qquad \exists v_1', v_2' \in \mathbf{V}.\ \ \left(\overline{\mathsf{ck}}(p_1), v_1'\right) \sim_p \left(\overline{\mathsf{ck}}(p_2), v_2'\right) \\
&\qquad\qquad \wedge\ \forall x \in fv(q).\ v_1(x) = v_2(x) \\
&\qquad\qquad \left. \wedge\ \exists d \in \mathbb{R}_{\geq 0}.\ \forall i = 1,2.\ \forall x \in \kappa fv(p_i).\ v_i(x) = (v_i' - d)(x) \right\} \\
&\cup \left\{ \left\langle \left[q \mid\mid_A \overline{\mathsf{ck}}(p_1), v_1\right], \left[q \mid\mid_A \overline{\mathsf{ck}}(p_2), v_2\right] \right\rangle \mid \right. \\
&\qquad\qquad \exists v_1', v_2' \in \mathbf{V}.\ \ \left[\overline{\mathsf{ck}}(p_1), v_1'\right] \sim_p \left[\overline{\mathsf{ck}}(p_2), v_2'\right] \\
&\qquad\qquad \wedge\ \forall x \in \kappa fv(q).\ v_1(x) = v_2(x) \\
&\qquad\qquad \left. \wedge\ \exists d \in \mathbb{R}_{\geq 0}.\ \forall i = 1,2.\ \forall x \in \kappa fv(p_i).\ v_i(x) = (v_i' - d)(x) \right\}
\end{aligned}
$$

where κfv *is defined as* $\kappa fv(p) \stackrel{\mathrm{def}}{=} \kappa(p) \cup fv(p)$.

Proof. We only prove that R_l^{m} is a probabilistic bisimulation up to $\sim_p$. The proof for R_r^{m} follows exactly in the same way. We remark that R_l^{m} is symmetric since $\sim_p$ and $=$ are symmetric as well. The way in which we proceed to prove the first transfer property is quite similar to the case of summation.

1. We consider two different cases according to the definition of R_l^{m}.

Case $\langle (p_1 \,||_A\, q, v_1), (p_2 \,||_A\, q, v_2)\rangle \in R_l^{\mathrm{m}}$. Let $S \subseteq \Sigma'/(\sim_p \cup R_l^{\mathrm{m}})^*$. Let $\overrightarrow{\kappa(p_i \,||_A\, q)} = \overrightarrow{\kappa(p_i) \cup \kappa(q)} = (x_1^i, \ldots, x_{n_i}^i, y_1, \ldots, y_m)$. Then

$$T\,(p_i \,||_A\, q, v_i) = \mathcal{D}_{v_i}^{p_i||_A q}(\mathcal{R}(F_{x_1^i}, \ldots, F_{x_{n_i}^i}, F_{y_1}, \ldots, F_{y_m})).$$

We have to show that

$$\mu((p_1 \,||_A\, q, v_1), \bigcup S) \;=\; \mu((p_2 \,||_A\, q, v_2), \bigcup S) \tag{F.15}$$

For $i = 1, 2$, we calculate,

$$\begin{aligned}
&\mu((p_i \,||_A\, q, v_i), \bigcup S)\\
&\qquad = \; P_{v_i}^{p_i||_A q}\big((\bigcup S) \cap \Omega_{v_i}^{p_i||_A q}\big) && \text{(by def. of } \mu)\\
&\qquad = \; P_{n_i+m}\left(\big(\mathcal{D}_{v_i}^{p_i||_A q}\big)^{-1}\big((\bigcup S) \cap \Omega_{v_i}^{p_i||_A q}\big)\right) && \text{(by def. of } P_{v_i}^{p_i||_A q})\\
&\qquad = \; P_{n_i+m}(B_i) && \text{(with } B_i = \big(\mathcal{D}_{v_i}^{p_i||_A q}\big)^{-1}\big((\bigcup S) \cap \Omega_{v_i}^{p_i||_A q}\big))\\
&\qquad = \; \int_{\mathcal{B}(\mathbb{R}^{n_i+m})} B_i \,\mathbf{d}P_{n_i+m}\\
&\qquad = \; \int_{\mathcal{B}(\mathbb{R}^{n_i+m})} B_i(\vec{d_i}, \vec{d}) \,\mathbf{d}P_{n_i+1}(\vec{d_i}, \vec{d})\\
&\qquad = \; \int_{\mathcal{B}(\mathbb{R}^m)} \left(\int_{\mathcal{B}(\mathbb{R}^{n_i})} B_i(\vec{d_i}, \vec{d}) \,\mathbf{d}P_{n_i}(\vec{d_i})\right) \mathbf{d}P_m(\vec{d})
\end{aligned} \tag{F.16}$$

Let $B_{\vec{y}} = \{(d_1, \ldots, d_m) \mid \exists i \in \{1, 2\}.\ (d_1^i, \ldots, d_{n_i}^i, d_1, \ldots, d_m) \in B_i\}$. From (F.16), it suffices to prove that for all $\vec{d} \in B_{\vec{y}}$,

$$\int_{\mathcal{B}(\mathbb{R}^{n_1})} B_1(\vec{d_1}, \vec{d}) \,\mathbf{d}P_{n_1}(\vec{d_1}) \;=\; \int_{\mathcal{B}(\mathbb{R}^{n_2})} B_2(\vec{d_2}, \vec{d}) \,\mathbf{d}P_{n_2}(\vec{d_2}) \tag{F.17}$$

Notice that some of this integrals may be 0.

Since $\langle (p_1 \,||_A\, q, v_1), (p_2 \,||_A\, q, v_2)\rangle \in R_l^m$, there are valuations v_1' and v_2' such that $(p_1, v_1') \sim_p (p_2, v_2')$, for all $x \in fv(q)$ $v_1(x) = v_2(x)$, and for both $i = 1, 2$ and all $x \in fv(p_i)$, $v_i(x) = v_i'(x)$. Thus, the proof of (F.17) reduces to prove that for all $\vec{d} \in B_{\vec{y}}$, there is an $S_{\vec{d}} \subseteq \Sigma'/\sim_p$, such that for all $i = 1, 2$,

$$\mu((p_i, v_i'), \bigcup S_{\vec{d}}) \;=\; \int_{\mathcal{B}(\mathbb{R}^{n_i})} B_i(\vec{d_i}, \vec{d}) \,\mathbf{d}P_{n_i}(\vec{d_i}) \tag{F.18}$$

We can do so since $(p_1, v_1') \sim_p (p_2, v_2')$ implies $\mu((p_1, v_1'), \bigcup S_{\vec{d}}) = \mu((p_2, v_2'), \bigcup S_{\vec{d}})$.

But

$$\begin{aligned}\mu((p_i, v_i'), \bigcup S_{\vec{d}}) &= P^{p_i}_{v_i'}\big((\bigcup S_{\vec{d}}) \cap \Omega^{p_1}_{v_i'}\big) && \text{(by def. of } \mu\text{)}\\ &= P_{n_i}\Big(\big(\mathcal{D}^{p_i}_{v_i'}\big)^{-1}\big((\bigcup S_{\vec{d}}) \cap \Omega^{p_i}_{v_i'}\big)\Big) && \text{(by def. of } P^{p_i}_{v_i'}\text{)}\end{aligned}$$

For all $\vec{d} \in B_{\vec{y}}$, define

$$B_i^{\vec{d}} = \{(d_1^i, \ldots, d_{n_i}^i) \mid \vec{d} = (d_1, \ldots, d_m) \wedge (d_1^i, \ldots, d_{n_i}^i, d_1, \ldots, d_m) \in B_i\}.$$

Then, we only have to prove that for all $i = 1, 2$

$$B_i^{\vec{d}} = \big(\mathcal{D}^{p_i}_{v_i'}\big)^{-1}\big((\bigcup S_{\vec{d}}) \cap \Omega^{p_i}_{v_i'}\big) \tag{F.19}$$

So, choose $S_{\vec{d}}$ to be the minimal set satisfying

$$\left.\begin{array}{l}(\hat{v}_i'(x_1^i), \ldots, \hat{v}_i'(x_{n_i}^i)) \in B_i^{\vec{d}}\\ \wedge \quad \forall x \in \mathcal{C} - \kappa(p_i).\ \hat{v}_i'(x) = v_i'(x)\end{array}\right\} \implies [p_i, \hat{v}_i'] \in (\bigcup S_{\vec{d}}) \cap \Omega^{p_i}_{v_i'}$$

for both $i = 1, 2$. Clearly

$$B_i^{\vec{d}} \subseteq \big(\mathcal{D}^{p_i}_{v_i'}\big)^{-1}\big((\bigcup S_{\vec{d}}) \cap \Omega^{p_i}_{v_i'}\big)$$

To prove the other inclusion, we proceed as follows. We only prove case $i = 1$, the other follows in a similar manner. Assume

$$[p_1, u'] \in (\bigcup S_{\vec{d}}) \cap \Omega^{p_1}_{v_1'} \quad \text{such that} \quad (u'(x_1^1), \ldots, u'(x_{n_1}^1)) \notin B_1^{\vec{d}}$$

Since $S_{\vec{d}}$ is minimal, one of the following statements must hold:

(a) there is a $[p_1, \hat{v}_1'] \in (\bigcup S_{\vec{d}}) \cap \Omega^{p_1}_{v_1'}$ such that

$$(\hat{v}_1'(x_1^1), \ldots, \hat{v}_1'(x_{n_1}^1)) \in B_1^{\vec{d}} \quad \text{and} \quad [p_1, u'] \sim_p [p_1, \hat{v}_1']\,;\ \text{or}$$

(b) there is a $[p_2, \hat{v}_2'] \in (\bigcup S_{\vec{d}}) \cap \Omega^{p_2}_{v_2}$ such that

$$(\hat{v}_2'(x_1^2), \ldots, \hat{v}_2'(x_{n_2}^2)) \in B_2^{\vec{d}} \quad \text{and} \quad [p_1, u'] \sim_p [p_2, \hat{v}_2']\,.$$

Suppose that option (a) holds. Define $\hat{v}_1$ such that

$$\begin{array}{l}\hat{v}_1(y_j) = d_j \quad \text{for all } j \in \{1, \ldots, m\},\\ \hat{v}_1(x) = \hat{v}_1'(x) \quad \text{for all } x \in \kappa fv(p_1), \quad \text{and}\\ \hat{v}_1(x) = v_1(x) \quad \text{for all } x \in \mathcal{C} - (\kappa fv(p_1) \cup \kappa(q));\end{array}$$

and u such that for all $x \in \mathcal{C}$

$$u(x) = \begin{cases} u'(x) & \text{if } x \in \kappa fv(p_1) \\ \hat{v}_1(x) & \text{otherwise} \end{cases}$$

From the way we choose u and $\hat{v}_1$, we have that

$$\langle [p_1 \,||_A\, q, u], [p_1 \,||_A\, q, \hat{v}_1] \rangle \in R_l^{\mathrm{m}}$$

from which it is immediate that

$$\{[p_1 \,||_A\, q, u], [p_1 \,||_A\, q, \hat{v}_1]\} \in (\bigcup S) \cap \Omega_{v_1}^{p_1||_A q}$$

As a consequence

$$(u(x_1^1), \ldots, u(x_{n_1}^1), u(y_1), \ldots, u(y_m)) \in B_1$$

so,

$$(u'(x_1^1), \ldots, u'(x_{n_1}^1)) = (u(x_1^1), \ldots, u(x_{n_1}^1)) \in B_1^{\vec{d}}$$

which contradicts the assumption. So it should be the case that option (b) holds. We proceed in a similar way. Define $\hat{v}_2'$ such that

$$\begin{array}{l} \hat{v}_2(y_j) = d_j \quad \text{for all } j \in \{1, \ldots, m\}, \\ \hat{v}_2(x) = \hat{v}_2'(x) \quad \text{for all } x \in \kappa fv(p_2), \quad \text{and} \\ \hat{v}_2(x) = v_2(x) \quad \text{for all } x \in \mathcal{C} - (\kappa fv(p_2) \cup \kappa(q)); \end{array}$$

and u such that for all $x \in \mathcal{C}$

$$u(x) = \begin{cases} u'(x) & \text{if } x \in \kappa fv(p_1) \\ \hat{v}_2(x) & \text{if } x \in \kappa(q) \\ v_1(x) & \text{otherwise} \end{cases}$$

Because of the way we chose $\hat{v}_2$ and u, we have that

$$\langle [p_1 \,||_A\, q, u], [p_2 \,||_A\, q, \hat{v}_2] \rangle \in R_l^{\mathrm{m}}$$

from which it is immediate that

$$[p_2 \,||_A\, q, \hat{v}_2] \in (\bigcup S) \cap \Omega_{v_2}^{p_2||_A q} \quad \text{and} \quad [p_1 \,||_A\, q, u] \in (\bigcup S) \cap \Omega_{v_1}^{p_1||_A q}$$

As a consequence

$$(u(x_1^1), \ldots, u(x_{n_1}^1), u(y_1), \ldots, u(y_m)) \in B_1$$

so,

$$(u'(x_1^1), \ldots, u'(x_{n_1}^1)) = (u(x_1^1), \ldots, u(x_{n_1}^1)) \in B_1^{\vec{d}}$$

which again contradicts the assumption. The equality in (F.19) is finally proved, and therefore this transfer property.

Case $\left\langle \left(\overline{\mathsf{ck}}(p_1) \mid\mid_A q, v_1\right), \left(\overline{\mathsf{ck}}(p_2) \mid\mid_A q, v_2\right)\right\rangle \in R_l^{\mathrm{m}}$. This case is quite similar to the previous one in the particular instance of $n_i = 0$. Special care is needed in some parts in which we should consider that $v_i(x) = (v_i' - d)(x)$ for $x \in \kappa fv(p_i)$.

2. Two different cases arise according to the definition of R_l^{m}.

Case $\langle [p_1 \mid\mid_A q, v_1], [p_2 \mid\mid_A q, v_2]\rangle \in R_l^{\mathrm{m}}$.

$$
\begin{array}{ll}
[p_1 \mid\mid_A q, v_1] \xrightarrow{a(d)} (r, v_1 - d) & \\
\Longrightarrow & \text{(by Def. 9.7, rule \textbf{Open})} \\
\quad p_1 \mid\mid_A q \xrightarrow{a,C_1}{}' r \quad\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 & \\
\Longrightarrow & \text{(by rule \textbf{alpha})} \\
\quad p_1 \mid\mid_A q \xrightarrow{a,C_1} r' \quad\wedge\quad r \simeq_\alpha r' \quad\wedge\quad \neg\mathsf{cv}(r) & \\
\quad\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 & \\
\Longrightarrow & \text{(by Table 10.3)} \\
\quad (\ (p_1 \xrightarrow{a,C_1} p_1' \ \wedge\ r' \equiv p_1' \mid\mid_A \overline{\mathsf{ck}}(q)) & (*) \\
\quad\ \vee\ (q \xrightarrow{a,C_1} q' \ \wedge\ r' \equiv \overline{\mathsf{ck}}(p_1) \mid\mid_A q') & (**) \\
\quad\ \vee\ (p_1 \xrightarrow{a,C_1'} p_1' \ \wedge\ q \xrightarrow{a,C_1''} q' \ \wedge\ r' \equiv p_1' \mid\mid_A q' \ \wedge\ C_1 \equiv C_1' \cup C_1'')\ ) & (***) \\
\quad\wedge\quad r \simeq_\alpha r' \quad\wedge\quad \neg\mathsf{cv}(r) \quad\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 &
\end{array}
$$

We proceed with each subcase separately.

Sub-subcase (*)

$$
\begin{array}{ll}
p_1 \xrightarrow{a,C_1} p_1' \quad\wedge\quad r \simeq_\alpha p_1' \mid\mid_A \overline{\mathsf{ck}}(q) \quad\wedge\quad \neg\mathsf{cv}(r) & \\
\quad\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 & \\
\Longrightarrow & \text{(by Def. 10.11 } (\simeq_\alpha)\text{, with } r \equiv p_1'' \mid\mid_A q' \text{, and Def. 10.4 } (\mathsf{cv})) \\
\quad p_1 \xrightarrow{a,C_1} p_1' \quad\wedge\quad p_1'' \simeq_\alpha p_1' \quad\wedge\quad \neg\mathsf{cv}(p_1'') & \\
\quad\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 & \\
\Longrightarrow & \text{(by rule \textbf{alpha})} \\
\quad p_1 \xrightarrow{a,C_1}{}' p_1'' \quad\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 & \\
\Longrightarrow & \text{(by Def. 9.7, rule \textbf{Open})} \\
\quad [p_1, v_1] \xrightarrow{a(d)} (p_1'', v_1 - d) & \\
\Longrightarrow & \text{(by Cor. F.17.1)} \\
\quad [p_2, v_2] \xrightarrow{a(d)} (p_2'', v_2 - d) \quad\wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2'', (v_2 - d)) & \\
\Longrightarrow & \text{(by Def. 9.7, rule \textbf{Open})}
\end{array}
$$

$$
\begin{aligned}
& p_2 \xrightarrow{a,C_2}{}' p_2'' \quad\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2'', (v_2 - d)) \\
\Longrightarrow & \hfill \text{(by rule \textbf{alpha})} \\
& p_2 \xrightarrow{a,C_2} p_2' \quad\wedge\quad p_2'' \simeq_\alpha p_2' \quad\wedge\quad \neg\mathsf{cv}(p_2'') \\
& \wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2'', (v_2 - d)) \\
\Longrightarrow & \hfill \text{(by Table 10.3)} \\
& p_2 \,||_A\, q \xrightarrow{a,C_2} p_2' \,||_A\, \overline{\mathsf{ck}}(q) \quad\wedge\quad p_2'' \simeq_\alpha p_2' \quad\wedge\quad \neg\mathsf{cv}(p_2'') \\
& \wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2'', (v_2 - d)) \\
\Longrightarrow & \hfill \text{(by rule \textbf{alpha})} \\
& p_2 \,||_A\, q \xrightarrow{a,C_2}{}' p_2''' \,||_A\, q'' \quad\wedge\quad p_2''' \,||_A\, q'' \simeq_\alpha p_2' \,||_A\, \overline{\mathsf{ck}}(q) \quad\wedge\quad \neg\mathsf{cv}(p_2''' \,||_A\, q'') \\
& \wedge\quad p_2'' \simeq_\alpha p_2' \quad\wedge\quad \neg\mathsf{cv}(p_2'') \quad\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_2.\ (v_2 - d)(x) \leq 0 \\
& \wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2'', (v_2 - d)) \\
\Longrightarrow & \hfill \text{(by Def. 9.7, rule \textbf{Open}, and Def. 10.11 } (\simeq_\alpha)) \\
& [p_2 \,||_A\, q, v_2] \xrightarrow{a(d)} (p_2''' \,||_A\, q'', v_2 - d) \quad\wedge\quad p_2''' \simeq_\alpha p_2'' \\
& \wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2'', (v_2 - d)) \\
\Longrightarrow & \hfill (\simeq_\alpha \text{ implies } \sim_p) \\
& [p_2 \,||_A\, q, v_2] \xrightarrow{a(d)} (p_2''' \,||_A\, q', v_2 - d) \quad\wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2''', (v_2 - d)) \quad (\dagger)
\end{aligned}
$$

Since $p_1'' \,||_A\, q' \simeq_\alpha p_1' \,||_A\, \overline{\mathsf{ck}}(q)$ and $p_2''' \,||_A\, q'' \simeq_\alpha p_2' \,||_A\, \overline{\mathsf{ck}}(q)$, $q' \simeq_\alpha \overline{\mathsf{ck}}(q) \simeq_\alpha q''$. Thus, $fv(q') = fv(\overline{\mathsf{ck}}(q)) = fv(q'')$ and $\kappa(q') = \kappa(q'') = \kappa(\overline{\mathsf{ck}}(q)) = \varnothing$. As a consequence

$$
\begin{aligned}
& p_1'' \,||_A\, q' \simeq_\alpha p_1'' \,||_A\, \overline{\mathsf{ck}}(q), \quad p_2''' \,||_A\, q'' \simeq_\alpha p_2'' \,||_A\, \overline{\mathsf{ck}}(q), \\
& \neg\mathsf{cv}(p_1'' \,||_A\, \overline{\mathsf{ck}}(q)), \quad \text{and} \quad \neg\mathsf{cv}(p_2''' \,||_A\, \overline{\mathsf{ck}}(q)).
\end{aligned} \tag{F.20}
$$

In addition, notice that $(v_1 - d)(x) = (v_2 - d)(x)$ for all $x \in fv(\overline{\mathsf{ck}}(q)) \subseteq \kappa fv(q)$. Thus, we have,

$$
\begin{aligned}
& (\dagger) \implies \\
& \quad [p_2 \,||_A\, q, v_2] \xrightarrow{a(d)} (p_2''' \,||_A\, q', v_2 - d) \\
& \quad \wedge\quad \left\langle \left(p_1'' \,||_A\, \overline{\mathsf{ck}}(q), (v_1 - d)\right), \left(p_2''' \,||_A\, \overline{\mathsf{ck}}(q), (v_2 - d)\right) \right\rangle \in R_l^{\mathrm{m}} \\
& \Longrightarrow \hfill \text{(by (F.20), } \simeq_\alpha \text{ implies } \sim_p\text{, and transitivity of } \sim_p) \\
& [p_2 \,||_A\, q, v_2] \xrightarrow{a(d)} (p_2''' \,||_A\, q', v_2 - d) \\
& \quad \wedge\quad \langle (p_1'' \,||_A\, q', (v_1 - d)), (p_2''' \,||_A\, q'', (v_2 - d)) \rangle \in (\sim_p \cup R_l^{\mathrm{m}})^*
\end{aligned}
$$

Sub-subcase (**)

$$
\begin{array}{l}
q \xrightarrow{a,C_1} q' \quad \wedge \quad r \simeq_\alpha \overline{\mathsf{ck}}(p_1) \,||_A\, q' \quad \wedge \quad \neg \mathsf{cv}(r) \\
\quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 \\
\Longrightarrow \qquad\qquad \text{(by Def. 10.11 } (\simeq_\alpha)\text{, with } r \equiv p_1' \,||_A\, q_1' \text{, and Def. 10.4 } (\mathsf{cv})) \\
\quad q \xrightarrow{a,C_1} q' \quad \wedge \quad q' \simeq_\alpha q_1' \quad \wedge \quad \neg \mathsf{cv}(q_1') \\
\quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1.\ (v_1 - d)(x) \leq 0 \\
\Longrightarrow \qquad\qquad \text{(since } v_1(x) = v_2(x) \text{ for all } x \in C_1 \subseteq \kappa fv(q)) \\
\quad q \xrightarrow{a,C_1} q' \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1.\ (v_2 - d)(x) \leq 0 \\
\Longrightarrow \qquad\qquad \text{(by Table 10.3)} \\
\quad p_2 \,||_A\, q \xrightarrow{a,C_1} \overline{\mathsf{ck}}(p_2) \,||_A\, q' \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1.\ (v_2 - d)(x) \leq 0 \\
\Longrightarrow \qquad\qquad \text{(by rule } \textbf{alpha}) \\
\quad p_2 \,||_A\, q \xrightarrow{a,C_1}{}' p_2' \,||_A\, q_2' \quad \wedge \quad p_2' \,||_A\, q_2' \simeq_\alpha \overline{\mathsf{ck}}(p_2) \,||_A\, q' \quad \wedge \quad \neg \mathsf{cv}(p_2' \,||_A\, q_2') \\
\quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1.\ (v_2 - d)(x) \leq 0 \\
\Longrightarrow \qquad\qquad \text{(by Def. 9.7, rule } \textbf{Open}) \\
\quad [p_2 \,||_A\, q, v_2] \xrightarrow{a(d)} (p_2' \,||_A\, q_2', v_2 - d) \qquad\qquad \text{(F.21)}
\end{array}
$$

Choose $q^\star$ such that $q^\star \simeq_\alpha q'$ and $\kappa(q^\star)$ is a set of fresh variables not appearing in any of the processes above. Thus $\kappa(q^\star) \cap \kappa fv(\overline{\mathsf{ck}}(p_1)) = \kappa(q^\star) \cap \kappa fv(\overline{\mathsf{ck}}(p_2)) = \varnothing$. As a consequence, for $i = 1, 2$, $p_i' \,||_A\, q_i' \simeq_\alpha \overline{\mathsf{ck}}(p_i) \,||_A\, q' \simeq_\alpha \overline{\mathsf{ck}}(p_i) \,||_A\, q^\star$ and $\neg \mathsf{cv}(\overline{\mathsf{ck}}(p_i) \,||_A\, q^\star)$. (Recall that $r \equiv p_1' \,||_A\, q_1'$.) Thus,

$$(p_i' \,||_A\, q_i', v_i - d) \sim_p \left(\overline{\mathsf{ck}}(p_i) \,||_A\, q^\star, v_i - d\right) \qquad \text{(F.22)}$$

Moreover, $[p_1, v_1] \sim_p [p_2, v_2]$, by Corollary F.17.1, and so $\left(\overline{\mathsf{ck}}(p_1), v_1\right) \sim_p \left(\overline{\mathsf{ck}}(p_2), v_2\right)$ by Lemma F.24. Since in addition $(v_1 - d)(x) = (v_2 - d)(x)$ for all $x \in fv(q^\star) = fv(q') \subseteq \kappa fv(q)$, we have that

$$\langle \left(\overline{\mathsf{ck}}(p_1) \,||_A\, q^\star, v_1 - d\right), \left(\overline{\mathsf{ck}}(p_2) \,||_A\, q^\star, v_2 - d\right) \rangle \in R_l^{\mathrm{m}} \qquad \text{(F.23)}$$

All together, form (F.21), (F.22), and (F.23), we finally have

$$
\begin{array}{l}
[p_2 \,||_A\, q, v_2] \xrightarrow{a(d)} (p_2' \,||_A\, q_2', v_2 - d) \quad \text{and} \\
\qquad\qquad \langle (p_1' \,||_A\, q_1', v_1 - d), (p_2' \,||_A\, q_2', v_2 - d) \rangle \in (\sim_p \cup R_l^{\mathrm{m}})^*
\end{array}
$$

Sub-subcase (***)

$$
\begin{array}{l}
p_1 \xrightarrow{a,C_1'} p_1' \quad \wedge \quad q \xrightarrow{a,C_1''} q' \quad \wedge \quad r \simeq_\alpha p_1' \,||_A\, q' \quad \wedge \quad \neg \mathsf{cv}(r) \\
\quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall x \in C_1' \cup C_1''.\ (v_1 - d)(x) \leq 0 \\
\Longrightarrow \qquad\qquad \text{(by Def. 10.11 } (\simeq_\alpha)\text{, with } r \equiv p_1'' \,||_A\, q_1'\text{, and Def. 10.4 } (\mathsf{cv}))
\end{array}
$$

$$p_1 \xrightarrow{a,C_1} p_1' \quad\wedge\quad q \xrightarrow{a,C_1''} q' \quad\wedge\quad p_1'' \simeq_\alpha p_1' \quad\wedge\quad q_1' \simeq_\alpha q' \quad\wedge\quad \neg\mathsf{cv}(p_1'')$$
$$\wedge\quad \neg\mathsf{cv}(q_1') \quad\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_1' \cup C_1''.\ (v_1 - d)(x) \leq 0$$
$\Longrightarrow$ (by rule **alpha** and calculations)
$$p_1 \xrightarrow{a,C_1}{}' p_1'' \quad\wedge\quad q \xrightarrow{a,C_1''} q' \quad\wedge\quad d \in \mathbb{R}_{\geq 0}$$
$$\wedge\quad \forall x \in C_1'.\ (v_1 - d)(x) \leq 0 \quad\wedge\quad \forall x \in C_1''.\ (v_1 - d)(x) \leq 0$$
$\Longrightarrow$ (by Def. 9.7, rule **Open**)
$$[p_1, v_1] \xrightarrow{a(d)} (p_1'', v_1 - d) \quad\wedge\quad q \xrightarrow{a,C_1''} q'$$
$$\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_1''.\ (v_1 - d)(x) \leq 0$$
$\Longrightarrow$ (by Cor. F.17.1, and since $v_1(x) = v_2(x)$ for all $x \in C_1'' \subseteq \kappa fv(q)$)
$$[p_2, v_2] \xrightarrow{a(d)} (p_2''', v_2 - d) \quad\wedge\quad q \xrightarrow{a,C_1''} q'$$
$$\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_1''.\ (v_2 - d)(x) \leq 0$$
$$\wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2''', (v_2 - d))$$
$\Longrightarrow$ (by Def. 9.7, rule **Open**)
$$p_2 \xrightarrow{a,C_2}{}' p_2''' \quad\wedge\quad q \xrightarrow{a,C_1''} q'$$
$$\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_1''.\ (v_2 - d)(x) \leq 0$$
$$\wedge\quad \forall x \in C_2.\ (v_2 - d)(x) \leq 0 \quad\wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2''', (v_2 - d))$$
$\Longrightarrow$ (by rule **alpha** and calculations)
$$p_2 \xrightarrow{a,C_2} p_2' \quad\wedge\quad q \xrightarrow{a,C_1''} q' \quad\wedge\quad p_2''' \simeq_\alpha p_2' \quad\wedge\quad \neg\mathsf{cv}(p_2''')$$
$$\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_2 \cup C_1''.\ (v_2 - d)(x) \leq 0$$
$$\wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2''', (v_2 - d))$$
$\Longrightarrow$ (by Table 10.3)
$$p_2 \,||_A\, q \xrightarrow{a,C_2 \cup C_1''} p_2' \,||_A\, q' \quad\wedge\quad p_2''' \simeq_\alpha p_2'$$
$$\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_2 \cup C_1''.\ (v_2 - d)(x) \leq 0$$
$$\wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2''', (v_2 - d))$$
$\Longrightarrow$ (by rule **alpha**)
$$p_2 \,||_A\, q \xrightarrow{a,C_2 \cup C_1''}{}' p_2'' \,||_A\, q_2' \quad\wedge\quad p_2'' \,||_A\, q_2' \simeq_\alpha p_2' \,||_A\, q' \quad\wedge\quad \neg\mathsf{cv}(p_2'' \,||_A\, q_2')$$
$$\wedge\quad p_2''' \simeq_\alpha p_2' \quad\wedge\quad d \in \mathbb{R}_{\geq 0} \quad\wedge\quad \forall x \in C_2 \cup C_1''.\ (v_2 - d)(x) \leq 0$$
$$\wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2''', (v_2 - d))$$
$\Longrightarrow$ (by Def. 9.7, rule **Open**, and Def. 10.11 ($\simeq_\alpha$))
$$[p_2 \,||_A\, q, v_2] \xrightarrow{a(d)} (p_2'' \,||_A\, q_2', v_2 - d) \quad\wedge\quad p_2'' \simeq_\alpha p_2'''$$
$$\wedge\quad (p_1'', (v_1 - d)) \sim_p (p_2''', (v_2 - d))$$
$\Longrightarrow$ ($\simeq_\alpha$ implies $\sim_p$)

$$[p_2 \,||_A\, q, v_2] \xrightarrow{a(d)} (p_2'' \,||_A\, q_2', v_2 - d)$$
$$\wedge \quad (p_1'', (v_1 - d)) \sim_p (p_2'', (v_2 - d)) \tag{F.24}$$

As in the previous case, choose $q^\star$ such that $q^\star \simeq_\alpha q' \simeq_\alpha q_i'$ and $\kappa(q^\star)$ is a set of fresh variables not appearing in any of the processes above. Thus $\kappa(q^\star) \cap \kappa fv(p_i'') = \varnothing$, and moreover, $\kappa(p_i'') \cap fv(q^\star) = \kappa(p_i'') \cap fv(q_i') = \varnothing$, for $i = 1, 2$. As a consequence, $p_i'' \,||_A\, q_i' \simeq_\alpha p_i'' \,||_A\, q^\star$ and $\neg \mathsf{cv}(p_i'' \,||_A\, q^\star)$. (Recall that $r \equiv p_1'' \,||_A\, q_1'$.) Thus,

$$(p_i'' \,||_A\, q_i', v_i - d) \sim_p (p_i'' \,||_A\, q^\star, v_i - d) \tag{F.25}$$

Moreover, $(p_1'', v_1 - d) \sim_p (p_2'', v_2 - d)$ and $(v_1 - d)(x) = (v_2 - d)(x)$ for all $x \in fv(q^\star) = fv(q') \subseteq \kappa fv(q)$. Hence

$$\langle (p_1'' \,||_A\, q^\star, v_1 - d), (p_2'' \,||_A\, q^\star, v_2 - d) \rangle \in R_l^{\mathrm{m}} \tag{F.26}$$

All together, form (F.24), (F.25), and (F.26), we finally have

$$[p_2 \,||_A\, q, v_2] \xrightarrow{a(d)} (p_2'' \,||_A\, q_2', v_2 - d) \quad \text{and}$$
$$\langle (p_1'' \,||_A\, q_1', v_1 - d), (p_2'' \,||_A\, q_2', v_2 - d) \rangle \in (\sim_p \cup R_l^{\mathrm{m}})^*$$

Case $\left\langle \left[\overline{\mathsf{ck}}(p_1) \,||_A\, q, v_1\right], \left[\overline{\mathsf{ck}}(p_2) \,||_A\, q, v_2\right] \right\rangle \in R_l^{\mathrm{m}}$. This case proceeds quite similarly to the previous one. *Q.E.D.*

The proof of the following lemma is quite similar to the previous one.

Lemma F.26. *The following relations are probabilistic bisimulations up to* $\sim_p$.

$$\begin{aligned} R_l^{\mathrm{lm}} \stackrel{\text{def}}{=} \{ \langle (p_1 \between_A q, v_1), (p_2 \between_A q, v_2) \rangle \mid \exists v_1', v_2' \in \mathbf{V}.\ & (p_1, v_1') \sim_p (p_2, v_2') \\ & \wedge\ \forall x \in fv(q).\ v_1(x) = v_2(x) \\ & \wedge\ \forall i = 1, 2.\ \forall x \in fv(p_i).\ v_i(x) = v_i'(x) \ \} \\ \cup \{ \langle [p_1 \between_A q, v_1], [p_2 \between_A q, v_2] \rangle \mid \exists v_1', v_2' \in \mathbf{V}.\ & [p_1, v_1'] \sim_p [p_2, v_2'] \\ & \wedge\ \forall x \in \kappa fv(q).\ v_1(x) = v_2(x) \\ & \wedge\ \forall i = 1, 2.\ \forall x \in \kappa fv(p_i).\ v_i(x) = v_i'(x) \ \} \end{aligned}$$

$$\begin{aligned} R_r^{\mathrm{lm}} \stackrel{\text{def}}{=} \{ \langle (q \between_A p_1, v_1), (q \between_A p_2, v_2) \rangle \mid \exists v_1', v_2' \in \mathbf{V}.\ & (p_1, v_1') \sim_p (p_2, v_2') \\ & \wedge\ \forall x \in fv(q).\ v_1(x) = v_2(x) \\ & \wedge\ \forall i = 1, 2.\ \forall x \in fv(p_i).\ v_i(x) = v_i'(x) \ \} \\ \cup \{ \langle [q \between_A p_1, v_1], [q \between_A p_2, v_2] \rangle \mid \exists v_1', v_2' \in \mathbf{V}.\ & [p_1, v_1'] \sim_p [p_2, v_2'] \\ & \wedge\ \forall x \in \kappa fv(q).\ v_1(x) = v_2(x) \\ & \wedge\ \forall i = 1, 2.\ \forall x \in \kappa fv(p_i).\ v_i(x) = v_i'(x) \ \} \end{aligned}$$

$$
\begin{aligned}
R_l^{\mathrm{cm}} \stackrel{\mathrm{def}}{=} \{\langle (p_1 \mid_A q, v_1), (p_2 \mid_A q, v_2)\rangle \mid{}& \exists v_1', v_2' \in \mathbf{V}.\ \ (p_1, v_1') \sim_p (p_2, v_2') \\
&\wedge\ \forall x \in fv(q).\ \ v_1(x) = v_2(x) \\
&\wedge\ \forall i = 1, 2.\ \forall x \in fv(p_i).\ \ v_i(x) = v_i'(x)\ \} \\
\cup \{\langle [p_1 \mid_A q, v_1], [p_2 \mid_A q, v_2]\rangle \mid{}& \exists v_1', v_2' \in \mathbf{V}.\ \ [p_1, v_1'] \sim_p [p_2, v_2'] \\
&\wedge\ \forall x \in \kappa fv(q).\ \ v_1(x) = v_2(x) \\
&\wedge\ \forall i = 1, 2.\ \forall x \in \kappa fv(p_i).\ \ v_i(x) = v_i'(x)\ \}
\end{aligned}
$$

$$
\begin{aligned}
R_r^{\mathrm{cm}} \stackrel{\mathrm{def}}{=} \{\langle (q \mid_A p_1, v_1), (q \mid_A p_2, v_2)\rangle \mid{}& \exists v_1', v_2' \in \mathbf{V}.\ \ (p_1, v_1') \sim_p (p_2, v_2') \\
&\wedge\ \forall x \in fv(q).\ \ v_1(x) = v_2(x) \\
&\wedge\ \forall i = 1, 2.\ \forall x \in fv(p_i).\ \ v_i(x) = v_i'(x)\ \} \\
\cup \{\langle [q \mid_A p_1, v_1], [q \mid_A p_2, v_2]\rangle \mid{}& \exists v_1', v_2' \in \mathbf{V}.\ \ [p_1, v_1'] \sim_p [p_2, v_2'] \\
&\wedge\ \forall x \in \kappa fv(q).\ \ v_1(x) = v_2(x) \\
&\wedge\ \forall i = 1, 2.\ \forall x \in \kappa fv(p_i).\ \ v_i(x) = v_i'(x)\ \}
\end{aligned}
$$

Renaming

Lemma F.27. *The following relation is a probabilistic bisimulation.*

$$
\begin{aligned}
R^{\mathrm{r}} \stackrel{\mathrm{def}}{=} {}& \{\langle (p_1[f], v_1), (p_2[f], v_2)\rangle \mid (p_1, v_1) \sim_p (p_2, v_2)\} \\
&\cup \{\langle [p_1[f], v_1], [p_2[f], v_2]\rangle \mid [p_1, v_1] \sim_p [p_2, v_2]\}
\end{aligned}
$$

The proof of this lemma should be quite straightforward once the reader had understood the proofs for the other operators.

Appendix G

Proofs from Chapter 11

G.1 Proof of Proposition 11.4

Proposition 11.4. *Let $p = q$ be a particular instance of any axiom in Table 11.1. Then the behaviour of p is locally definable if and only if so is the behaviour of q.*

Proof. We only prove the case of axiom **CS3** which is the more involved one. The others follow the same technique.

Because of Theorem 11.3, $\{\!|C|\!\}\, p + \{\!|C'|\!\}\, q \sim_s \{\!|C \cup C'|\!\}\, (p + q)$. As a consequence $\kappa(\{\!|C|\!\}\, p + \{\!|C'|\!\}\, q) = \kappa(\{\!|C \cup C'|\!\}\, (p + q))$. From these, the requirement of finiteness on κ follows.

So, it remains to prove that $\neg\mathsf{cv}(\{\!|C|\!\}\, p + \{\!|C'|\!\}\, q)$ if and only if $\neg\mathsf{cv}(\{\!|C \cup C'|\!\}\, (p + q))$.

$$
\begin{array}{ll}
& \neg\mathsf{cv}(\{\!|C|\!\}\, p + \{\!|C'|\!\}\, q) \\
\iff & \text{(by Def. 10.4 (}\mathsf{cv}\text{))} \\
& \neg\mathsf{cv}(\{\!|C|\!\}\, p) \quad \wedge \quad \neg\mathsf{cv}(\{\!|C'|\!\}\, q) \\
& \wedge \quad \kappa(\{\!|C|\!\}\, p) \cap fv(\{\!|C'|\!\}\, q) = \kappa(\{\!|C'|\!\}\, q) \cap fv(\{\!|C|\!\}\, p) \\
& \qquad\qquad = \kappa(\{\!|C|\!\}\, p) \cap \kappa(\{\!|C'|\!\}\, q) = \varnothing \\
\iff & \text{(by Def. 10.4 (}\mathsf{cv}\text{), and def. of } \kappa \text{ and } fv\text{)} \\
& \neg\mathsf{cv}(p) \quad \wedge \quad \neg\mathsf{cv}(q) \\
& \wedge \quad (\kappa(p) \cup C) \cap (fv(q) - C') = (\kappa(q) \cup C') \cap (fv(p) - C) \\
& \qquad\qquad = (\kappa(p) \cup C) \cap (\kappa(q) \cup C') = \varnothing \\
\iff & \text{(by condition in } \mathbf{CS3} \text{ and calculations)} \\
& \neg\mathsf{cv}(p) \quad \wedge \quad \neg\mathsf{cv}(q) \\
& \wedge \quad \kappa(p) \cap fv(q) = \kappa(q) \cap fv(p) = \kappa(p) \cap \kappa(q) = \varnothing \\
\iff & \text{(by Def. 10.4 (}\mathsf{cv}\text{))}
\end{array}
$$

$$
\begin{aligned}
& \neg \mathsf{cv}(p+q) \\
\iff & \qquad\qquad\qquad\qquad\qquad \text{(by Def. 10.4 } (\mathsf{cv})) \\
& \neg \mathsf{cv}(\{\!|C \cup C'|\!\}\, p+q)
\end{aligned}
$$

Q.E.D.

G.2 Proof of Theorem 11.6

Theorem 11.6. *For every term $p \in \spadesuit^b$ whose behaviour is definable, there is a basic term q such that $p = q$ can be proven by means of the axiom system $ax(\spadesuit^b)$. Moreover, the behaviour of q is definable as well.*

Proof. We proceed by structural induction. In all the cases, since we start from a process whose behaviour is definable, thanks to Proposition 11.4, we always arrive to a basic term whose behaviour is definable.

Case $\mathbf{0}$. We can calculate

$$\mathbf{0} \stackrel{\text{CS1}}{=} \{\!|\varnothing|\!\}\, \mathbf{0}$$

Notice that the behaviours of both $\mathbf{0}$ and $\{\!|\varnothing|\!\}\, \mathbf{0}$ are definable.

Case $a;p$. By induction hypothesis, assume p is a basic term whose behaviour is definable. Then $a;p$ is also definable and

$$a;p \stackrel{\text{T1}}{=} \varnothing \longmapsto a;p \stackrel{\text{CS1}}{=} \{\!|\varnothing|\!\}\,(\varnothing \longmapsto a;p)$$

Case $C \longmapsto p$. Since the behaviour of $C \longmapsto p$ is definable, so is the behaviour of p. Therefore, by induction we can assume that

$$p = \{\!|C'|\!\} \left(\sum_{i \in I} C_i \longmapsto a_i; p_i \right)$$

where each p_i is already a basic term whose behaviour is definable. Notice that in addition $C \cap C' = \varnothing$. We proceed as follows.

$$
\begin{aligned}
C \longmapsto p \quad & \stackrel{\text{T4}}{=} \quad \{\!|C'|\!\} \left(C \longmapsto \sum_{i \in I} C_i \longmapsto a_i; p_i \right) \\
& \stackrel{\text{T5,T3}}{=} \quad \{\!|C'|\!\} \left(\sum_{i \in I} (C \cup C_i) \longmapsto a_i; p_i \right)
\end{aligned}
$$

Case $p + q$. Since the behaviour of $p + q$ is definable, so are the behaviours of p and q. Hence, by induction we can assume that

$$p = \{\!|C|\!\} \left(\sum_{i \in I} C_i \mapsto a_i; p_i \right) \quad \text{and} \quad q = \{\!|C'|\!\} \left(\sum_{j \in J} C'_j \mapsto b_j; q_j \right)$$

where each p_i and q_j are already basic terms whose behaviours are definable. Notice that in addition $C \cap C' = C \cap fv(q) = C' \cap fv(p) = \varnothing$. Hence, we calculate,

$$\begin{aligned} p + q \quad &\overset{\text{ind.}}{=} \quad \{\!|C|\!\} \left(\sum_{i \in I} C_i \mapsto a_i; p_i \right) + \{\!|C'|\!\} \left(\sum_{j \in J} C'_j \mapsto b_j; q_j \right) \\ &\overset{\text{CS3}}{=} \quad \{\!|C \cup C'|\!\} \left(\sum_{i \in I} (C_i \mapsto a_i; p_i) + \sum_{j \in J} (C'_j \mapsto b_j; q_j) \right) \end{aligned}$$

Case $\{\!|C|\!\}\, p$. Since the behaviour of $\{\!|C|\!\}\, p$ is definable, so is the behaviour of p. Therefore, by induction we can assume that

$$p = \{\!|C'|\!\} \left(\sum_{i \in I} C_i \mapsto a_i; p_i \right)$$

where each p_i is already a basic term whose behaviour is definable. Then,

$$\{\!|C|\!\}\, p \overset{\text{CS2}}{=} \{\!|C \cup C'|\!\} \left(\sum_{i \in I} C_i \mapsto a_i; p_i \right)$$

Q.E.D.

G.3 Proof of Theorem 11.12

Theorem 11.12. *Let p and q be two $\clubsuit$ processes such that their behaviour is α-definable. If $p = q$ is proved by means of equational reasoning using $ax(\clubsuit^b) + \mathbf{O}$ and α-conversion, then $p \sim_o q$.*

Proof. By α-conversion, we can assume that, for all axiom $p = q$ in $ax(\clubsuit^b) + \mathbf{O}$, p and q are locally definable. From this, Theorem 11.3, and Theorems 9.19 and 9.20, it follows that if $p = q$ is proved using $ax(\clubsuit^b)$ then $p \sim_o q$.

To prove that axioms **Sy1**, **Sy2**, and **Sy3** preserve open p-bisimulation, we show that

they actually preserve symbolic bisimulation. We respectively give the following relations.

$$R_1 \stackrel{\text{def}}{=} \{\langle \{\!|C|\!\}\, p, p, Id_{rel(p)}\rangle \mid p \in \hourglass^\alpha \ \wedge\ C \cap fv(p) = \varnothing\} \ \cup\ R_{Id}$$

$$\begin{aligned} R_2 \stackrel{\text{def}}{=} &\{\langle C \mapsto a; C \mapsto p, C \mapsto a; p, Id_{FV(p)\cup C}\rangle \mid p \in \hourglass^\alpha \ \wedge\ C \cap \kappa(p) = \varnothing\} \\ &\cup \{\langle C \mapsto p, p, SR\rangle \mid p \in \hourglass^\alpha \ \wedge\ C \cap \kappa(p) = \varnothing \\ &\qquad\qquad \wedge\ SR = Id_{rel(p)} \cup \{\langle \{x\}, \varnothing\rangle \mid x \in C - rel(p)\}\ \} \\ &\cup R_{Id} \end{aligned}$$

$$\begin{aligned} R_3 \stackrel{\text{def}}{=} &\{\langle \{\!|C|\!\}\, p, \{\!|z|\!\}\, \xi_{\{z/C\}}p, SR\rangle \mid p \in \hourglass^\alpha \ \wedge\ \mathsf{Lnk}(C,p) \ \wedge\ z \notin var(p) \\ &\qquad \wedge\ \forall t \in \mathbb{R}_{\geq 0}.\ F_z(t) = \textstyle\prod_{x\in C} F_x(t) \\ &\qquad \wedge\ SR = Id_{rel(p)-(C\cap FV(p))} \cup \{\langle C \cap FV(p), \{z\}\rangle \mid C \cap FV(p) \neq \varnothing\}\ \} \\ &\cup \{\langle p, \xi_{\{z/C\}}p, SR\rangle \mid p \in \hourglass^\alpha \ \wedge\ \mathsf{Lnk}(C,p) \ \wedge\ z \notin var(p) \\ &\qquad \wedge\ \forall t \in \mathbb{R}_{\geq 0}.\ F_z(t) = \textstyle\prod_{x\in C} F_x(t) \\ &\qquad \wedge\ SR = Id_{rel(p)-(C\cap FV(p))} \cup \{\langle C \cap FV(p), \{z\}\rangle \mid C \cap FV(p) \neq \varnothing\}\ \} \\ &\cup R_{Id} \end{aligned} \tag{G.1}$$

where $Id_C \stackrel{\text{def}}{=} \{\langle \{x\}, \{x\}\rangle \mid x \in C\}$, and $R_{Id} \stackrel{\text{def}}{=} \{\langle p, p, Id_{rel}(p)\rangle \mid p \in \hourglass^\alpha\}$. Clearly, R_{Id} is a symbolic bisimulation by itself. As an example, we only give the proof that R_3 is a symbolic bisimulation which can be found in Lemma G.3 below. Notice that in R_3, we have consider the stronger requirement that $z \notin var(p)$. This is not a problem since by α-conversion we can always rename z into any $z' \notin fv(p)$, which is the weaker requirement of axiom **Sy3**.

To prove that **A3′** and **Red** preserve open p-bisimulation, we give the following relations.

$$\begin{aligned} R_{\mathrm{A3'}} \stackrel{\text{def}}{=} \big(\ &\{\langle (\{\!|x,y|\!\}\, (\{x\} \mapsto a; p + \{y\} \mapsto a; p), v)\,, (\{\!|z|\!\}\, \{z\} \mapsto a; p, v)\rangle \mid \\ &\qquad p \in \hourglass^\alpha \ \wedge\ \{x,y,z\} \cap fv(p) = \varnothing \ \wedge\ \forall t \in \mathbb{R}_{\geq 0}.\ F_z(t) = F_{\min\{x,y\}}(t)\ \} \\ &\cup \{\langle [\{\!|x,y|\!\}\, (\{x\} \mapsto a; p + \{y\} \mapsto a; p), v_1]\,, [\{\!|z|\!\}\, \{z\} \mapsto a; p, v_2]\rangle \mid \\ &\qquad p \in \hourglass^\alpha \ \wedge\ \{x,y,z\} \cap fv(p) = \varnothing \ \wedge\ \forall t \in \mathbb{R}_{\geq 0}.\ F_z(t) = F_{\min\{x,y\}}(t) \\ &\qquad \wedge\ \min\{v_1(x), v_1(y)\} = v_2(z) \ \wedge\ \forall w \in \mathcal{C} - \{x,y,z\}.\ v_1(w) = v_2(w)\ \}\ \big)^{\mathsf{s}} \end{aligned} \tag{G.2}$$

$$\begin{aligned} R_{\mathrm{Red}} \stackrel{\text{def}}{=} \big(\ &\{\langle (\{\!|x|\!\}\, \{x\} \mapsto p, v)\,, (\{\!|x|\!\}\, p, v)\rangle \mid\ p \in \hourglass^\alpha\ \} \\ &\cup \{\langle [\{\!|x|\!\}\, \{x\} \mapsto p, v]\,, [\{\!|x|\!\}\, p, v]\rangle \mid\ p \in \hourglass^\alpha \ \wedge\ v(x) \leq 0\ \}\ \big)^{\mathsf{s}} \end{aligned}$$

It can be shown that both relations are probabilistic bisimulation up to $\sim_p$ in the open behaviour. In particular, we report the proof for $R_{\mathrm{A3'}}$ in Lemma G.4 below. *Q.E.D.*

In the proofs that relations R_3 and $R_{\mathrm{A}3'}$ are respectively a symbolic bisimulation and an open p-bisimulation we restrict to the subset of definable $\clubsuit$ terms (i.e. $\clubsuit^{\mathsf{cf}}$). We do so in order not to burden the proofs with α-conversion treatment which has already being quite exhaustive along proofs in Appendix F. The relevant reasoning and techniques are all included in the following proofs.

Lemma G.1. *Let $p \in \clubsuit^{\mathsf{cf}}$ and $C \in \mathcal{C}$. If $\mathsf{Lnk}(C, p)$ and $C \cap FV(p) \neq \varnothing$ then $C \subseteq FV(p)$.*

Proof. We proceed by induction on the depth proof of $\mathsf{Lnk}(C, p)$, by doing case analysis in each of the rules.

Case **Ln1**. Trivial. $C \cap fv(p) = \varnothing$ implies $C \cap FV(p) = \varnothing$.

Case **Ln2**.

$$
\begin{array}{ll}
\mathsf{Lnk}(C, a; p) \quad \wedge \quad C \cap FV(a; p) \neq \varnothing & \\
\Longrightarrow & (\text{by } \mathbf{Ln2} \text{ and } FV(a;p) = FV(p)) \\
\quad \mathsf{Lnk}(C, p) \quad \wedge \quad C \cap FV(p) \neq \varnothing & \\
\Longrightarrow & (\text{induction}) \\
\quad C \subseteq FV(p) = FV(a; p) &
\end{array}
$$

Case **Ln3**. Notice that if $C \cap FV(C' \mapsto p) \neq \varnothing$ then $C' \mapsto p \xrightarrow{a, C^\star}$, for some a and $C^\star$. Moreover, by rules in Table 10.3, $C' \subseteq C^\star$ and as a consequence $C' \subseteq FV(C' \mapsto p)$.

$$
\begin{array}{ll}
\mathsf{Lnk}(C, C' \mapsto p) \quad \wedge \quad C \cap FV(C' \mapsto p) \neq \varnothing & \\
\Longrightarrow & (\text{by } \mathbf{Ln3}) \\
\quad C \subseteq C' \quad \wedge \quad C \cap fv(p) = \varnothing & \\
\Longrightarrow & (\text{since } C' \subseteq FV(C' \mapsto p)) \\
\quad C \subseteq FV(C' \mapsto p) &
\end{array}
$$

Case **Ln4**.

$$
\begin{array}{ll}
\mathsf{Lnk}(C, C' \mapsto p) \quad \wedge \quad C \cap FV(C' \mapsto p) \neq \varnothing & \\
\Longrightarrow & (\text{by } \mathbf{Ln4}) \\
\quad \mathsf{Lnk}(C, p) \quad \wedge \quad C \cap C' = \varnothing \quad \wedge \quad C \cap FV(C' \mapsto p) \neq \varnothing & \\
\Longrightarrow & (\text{since } FV(p) \supseteq FV(C' \mapsto p) - C') \\
\quad \mathsf{Lnk}(C, p) \quad \wedge \quad C \cap FV(p) \neq \varnothing & \\
\Longrightarrow & (\text{by induction}) \\
\quad C \subseteq FV(p) \subseteq FV(C' \mapsto p) &
\end{array}
$$

Case **Ln5**.

$$
\begin{array}{ll}
& \mathsf{Lnk}(C, \{\!|C'|\!\}\, p) \quad \wedge \quad C \cap FV(\{\!|C'|\!\}\, p) \neq \varnothing \\
\Longrightarrow & \qquad \text{(by } \mathbf{Ln5} \text{ and } FV(p) \supseteq FV(\{\!|C'|\!\}\, p)) \\
& \mathsf{Lnk}(C, p) \quad \wedge \quad C \cap C' = \varnothing \quad \wedge \quad C \cap FV(p) \neq \varnothing \\
\Longrightarrow & \qquad \text{(by induction)} \\
& C \subseteq FV(p) \quad \wedge \quad C \cap C' = \varnothing \\
\Longrightarrow & \qquad \text{(since } FV(\{\!|C'|\!\}\, p) \supseteq FV(p) - C') \\
& C \subseteq FV(\{\!|C'|\!\}\, p)
\end{array}
$$

Case **Ln6**.

$$
\begin{array}{ll}
& \mathsf{Lnk}(C, p+q) \quad \wedge \quad C \cap FV(p+q) \neq \varnothing \\
\Longrightarrow & \qquad \text{(by } \mathbf{Ln5} \text{ and } FV(p+q) = FV(p) \cup FV(q)) \\
& \mathsf{Lnk}(C, p) \quad \wedge \quad \mathsf{Lnk}(C, q) \quad \wedge \quad (C \cap FV(p) \neq \varnothing \vee C \cap FV(q) \neq \varnothing) \\
\Longrightarrow & \qquad \text{(by calculations)} \\
& (\ \mathsf{Lnk}(C, p) \quad \wedge \quad C \cap FV(p) \neq \varnothing\ ) \quad \vee \quad (\ \mathsf{Lnk}(C, q) \quad \wedge \quad C \cap FV(q) \neq \varnothing\ ) \\
\Longrightarrow & \qquad \text{(by induction)} \\
& C \subseteq FV(p) \quad \vee \quad C \subseteq FV(q) \\
\Longrightarrow & \qquad \text{(since } FV(p+q) = FV(p) \cup FV(q)) \\
& C \subseteq FV(p+q)
\end{array}
$$

Q.E.D.

Lemma G.2. *Let $p \in \mathcal{P}^{\mathsf{cf}}$, $C \in \mathcal{C}$, and $z \notin var(p)$. If $\mathsf{Lnk}(C, p)$, then either $C \cap FV(p) = \varnothing$, or*

1. *$p \xrightarrow{a, C_1} p'$ implies one of the following statements:*
   - *(a) $\xi_{\{z/C\}}p \xrightarrow{a, C_1} \xi_{\{z/C\}}p'$, $C \cap C_1 = \varnothing$, and $\mathsf{Lnk}(C, p')$; or*
   - *(b) $\xi_{\{z/C\}}p \xrightarrow{a, C_2} p'$, $C \subseteq C_1$, and $C_2 = (C_1 - C) \cup \{z\}$.*
2. *$\xi_{\{z/C\}}p \xrightarrow{a, C_1} p'$ implies one of the following statements:*
   - *(a) $p \xrightarrow{a, C_1} q$ with $q \equiv \xi_{\{z/C\}}p'$, $C \cap C_1 = \varnothing$, and $\mathsf{Lnk}(C, p')$; or*
   - *(b) $p \xrightarrow{a, C_2} p'$, $C \subseteq C_2$, and $C_1 = (C_2 - C) \cup \{z\}$.*

Proof. We prove that, under the condition $C \cap FV(p) \neq \varnothing$, then items 1. and 2. hold.

1. We proceed by induction on the depth proof of $\mathsf{Lnk}(C, p)$, by doing case analysis in each of the rules.

Case **Ln1**. Trivial since $C \cap fv(p) = \varnothing$ and hence $C \cap FV(p) = \varnothing$.

Case **Ln2**.

$$
\begin{array}{l}
a; p \xrightarrow{a,\varnothing} p \quad \wedge \quad \mathsf{Lnk}(C, a; p) \\
\Longrightarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by def. of substitution, Table 10.3, and } \mathbf{Ln2}) \\
\quad \xi_{\{z/C\}}(a; p) \xrightarrow{a,\varnothing} \xi_{\{z/C\}} p \quad \wedge \quad C \cap \varnothing = \varnothing \quad \wedge \quad \mathsf{Lnk}(C, p)
\end{array}
$$

Therefore, item (a) always hold in this case.

Case **Ln3**.

$$
\begin{array}{l}
C' \mapsto p \xrightarrow{a,C' \cup C_1} p' \quad \wedge \quad \mathsf{Lnk}(C, C' \mapsto p) \\
\Longrightarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by Table 10.3, and } \mathbf{Ln3}) \\
\quad p \xrightarrow{a,C_1} p' \quad \wedge \quad C \subseteq C' \quad \wedge \quad C \cap fv(p) = \varnothing \\
\Longrightarrow \qquad\qquad\qquad\qquad \text{(since } C \cap fv(p) = \varnothing \text{ implies } \xi_{\{z/C\}} p \equiv p, \text{ and calculations)} \\
\quad \xi_{\{z/C\}} p \xrightarrow{a,C_1} p' \quad \wedge \quad C \subseteq C' \\
\quad \wedge \quad C \cap fv(p) = \varnothing \quad \wedge \quad \xi_{\{z/C\}} C' = (C' - C) \cup \{z\} \\
\Longrightarrow \qquad\qquad\qquad\qquad\qquad\qquad \left(\begin{array}{c} \text{since } (C \cap fv(p) = \varnothing \wedge \neg \mathsf{cv}((C' \cup C) \mapsto p)) \\ \text{implies } C_1 \cap C = \varnothing, \text{ and calculations} \end{array} \right) \\
\quad \xi_{\{z/C\}} p \xrightarrow{a,C_1} p' \quad \wedge \quad C \subseteq C' \cup C_1 \\
\quad \wedge \quad (\xi_{\{z/C\}} C') \cup C_1 = ((C' \cup C_1) - C) \cup \{z\} \\
\Longrightarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by Table 10.3 and def. of substitution)} \\
\quad \xi_{\{z/C\}}(C' \mapsto p) \xrightarrow{a,(\xi_{\{z/C\}} C') \cup C_1} p' \quad \wedge \quad C \subseteq C' \cup C_1 \\
\quad \wedge \quad (\xi_{\{z/C\}} C') \cup C_1 = ((C' \cup C_1) - C) \cup \{z\}
\end{array}
$$

In this case, item (b) always hold.

Case **Ln4**.

$$
\begin{array}{l}
C' \mapsto p \xrightarrow{a,C' \cup C_1} p' \quad \wedge \quad \mathsf{Lnk}(C, C' \mapsto p) \\
\Longrightarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by Table 10.3, and } \mathbf{Ln4}) \\
\quad p \xrightarrow{a,C_1} p' \quad \wedge \quad \mathsf{Lnk}(C, p) \quad \wedge \quad C \cap C' = \varnothing \\
\Longrightarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by induction)} \\
\quad \big(\; (\; \xi_{\{z/C\}} p \xrightarrow{a,C_1} \xi_{\{z/C\}} p' \quad \wedge \quad C \cap C_1 = \varnothing \quad \wedge \quad \mathsf{Lnk}(C, p') \;) \\
\qquad \vee \quad (\; \xi_{\{z/C\}} p \xrightarrow{a,C_2} p' \quad \wedge \quad C \subseteq C_1 \quad \wedge \quad C_2 = (C_1 - C) \cup \{z\} \;) \; \big) \\
\quad \wedge \quad C \cap C' = \varnothing
\end{array}
$$

$$
\begin{array}{ll}
\Longrightarrow & \text{(by Table 10.3, def. of substitution, and calculations)} \\
\quad (\ \xi_{\{z/C\}}(C' \mapsto p) \xrightarrow{a, C' \cup C_1} \xi_{\{z/C\}} p' & \\
\qquad\qquad \wedge \quad C \cap (C' \cup C_1) = \varnothing \quad \wedge \quad \mathsf{Lnk}(C, p') \ ) & \\
\quad \vee \quad (\ \xi_{\{z/C\}}(C' \mapsto p) \xrightarrow{a, C' \cup C_2} p' & \\
\qquad\qquad \wedge \quad C \subseteq C' \cup C_1 \quad \wedge \quad C' \cup C_2 = ((C' \cup C_1) - C) \cup \{z\} \ ) &
\end{array}
$$

Case **Ln5**.

$$
\begin{array}{ll}
\{\!|C'|\!\}\, p \xrightarrow{a, C_1} p' \quad \wedge \quad \mathsf{Lnk}(C, \{\!|C'|\!\}\, p) & \\
\Longrightarrow & \text{(by Table 10.3, and } \mathbf{Ln5}\text{)} \\
\quad p \xrightarrow{a, C_1} p' \quad \wedge \quad \mathsf{Lnk}(C, p) \quad \wedge \quad C \cap C' = \varnothing & \\
\Longrightarrow & \text{(by induction)} \\
\quad \big(\ (\ \xi_{\{z/C\}} p \xrightarrow{a, C_1} \xi_{\{z/C\}} p' \quad \wedge \quad C \cap C_1 = \varnothing \quad \wedge \quad \mathsf{Lnk}(C, p') \ ) & \\
\qquad \vee \quad (\ \xi_{\{z/C\}} p \xrightarrow{a, C_2} p' \quad \wedge \quad C \subseteq C_1 \quad \wedge \quad C_2 = (C_1 - C) \cup \{z\} \ ) \ \big) & \\
\quad \wedge \quad C \cap C' = \varnothing & \\
\Longrightarrow & \text{(by Table 10.3, def. of substitution)} \\
\quad (\ \xi_{\{z/C\}}(\{\!|C'|\!\}\, p) \xrightarrow{a, C_1} \xi_{\{z/C\}} p' \quad \wedge \quad C \cap C_1 = \varnothing \quad \wedge \quad \mathsf{Lnk}(C, p') \ ) & \\
\quad \vee \quad (\ \xi_{\{z/C\}}(\{\!|C'|\!\}\, p) \xrightarrow{a, C_2} p' \quad \wedge \quad C \subseteq C_1 \quad \wedge \quad C_2 = (C_1 - C) \cup \{z\} \ ) &
\end{array}
$$

Case **Ln6**. This case follows straightforwardly by induction in a similar manner to the previous case.

2. This item follows quite similarly to the previous one. We only report the most interesting cases.

Case **Ln3**.

$$
\begin{array}{ll}
\xi_{\{z/C\}}(C' \mapsto p) \xrightarrow{a, (\xi_{\{z/C\}} C') \cup C_1} p' \quad \wedge \quad \mathsf{Lnk}(C, C' \mapsto p) & \\
\Longrightarrow & \text{(by def. of substitution, Table 10.3, and } \mathbf{Ln3}\text{)} \\
\quad \xi_{\{z/C\}} p \xrightarrow{a, C_1} p' \quad \wedge \quad C \subseteq C' \quad \wedge \quad C \cap fv(p) = \varnothing & \\
\Longrightarrow & \text{(since } C \cap fv(p) = \varnothing \text{ implies } \xi_{\{z/C\}} p \equiv p\text{, and calculations)} \\
\quad p \xrightarrow{a, C_1} p' \quad \wedge \quad C \subseteq C' & \\
\quad \wedge \quad C \cap fv(p) = \varnothing \quad \wedge \quad \xi_{\{z/C\}} C' = (C' - C) \cup \{z\} & \\
\Longrightarrow & \left(\begin{array}{c} \text{since } (C \cap fv(p) = \varnothing \wedge \neg \mathsf{cv}((C' \cup C) \mapsto p)) \\ \text{implies } C_1 \cap C = \varnothing, \text{ and calculations} \end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
p \xrightarrow{a,C_1} p' \quad \wedge \quad C \subseteq C' \cup C_1 \\
\wedge \quad (\xi_{\{z/C\}} C') \cup C_1 = ((C' \cup C_1) - C) \cup \{z\} \\
\Longrightarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by Table 10.3)} \\
C' \mapsto p \xrightarrow{a,C' \cup C_1} p' \quad \wedge \quad C \subseteq C' \cup C_1 \\
\wedge \quad (\xi_{\{z/C\}} C') \cup C_1 = ((C' \cup C_1) - C) \cup \{z\}
\end{array}
$$

In this case, item (b) always holds.

Case **Ln4**.

$$
\begin{array}{l}
\xi_{\{z/C\}}(C' \mapsto p) \xrightarrow{a,(\xi_{\{z/C\}} C') \cup C_1} p' \quad \wedge \quad \mathsf{Lnk}(C, C' \mapsto p) \\
\Longrightarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by def. of substitution, Table 10.3, and \textbf{Ln4})} \\
\xi_{\{z/C\}} p \xrightarrow{a,C_1} p' \quad \wedge \quad \mathsf{Lnk}(C, p) \quad \wedge \quad C \cap C' = \varnothing \\
\Longrightarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by induction)} \\
\big(\, (\, p \xrightarrow{a,C_1} q \quad \wedge \quad q \equiv \xi_{\{z/C\}} p' \quad \wedge \quad C \cap C_1 = \varnothing \quad \wedge \quad \mathsf{Lnk}(C, p') \,) \\
\quad \vee \quad (\, p \xrightarrow{a,C_2} p' \quad \wedge \quad C \subseteq C_2 \quad \wedge \quad C_1 = (C_2 - C) \cup \{z\} \,) \, \big) \\
\wedge \quad C \cap C' = \varnothing \\
\Longrightarrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by Table 10.3 and calculations)} \\
(\, C' \mapsto p \xrightarrow{a,(\xi_{\{z/C\}} C') \cup C_1} q \quad \wedge \quad q \equiv \xi_{\{z/C\}} p' \\
\qquad\qquad \wedge \quad C \cap (C' \cup C_1) = \varnothing \quad \wedge \quad \mathsf{Lnk}(C, p') \,) \\
\vee \quad (\, C' \mapsto p \xrightarrow{a,C' \cup C_2} p' \\
\qquad\qquad \wedge \quad C \subseteq C' \cup C_2 \quad \wedge \quad (\xi_{\{z/C\}} C') \cup C_1 = ((C' \cup C_2) - C) \cup \{z\} \,)
\end{array}
$$

Q.E.D.

Lemma G.3. *The relation R_3, as specified in the proof of Theorem 11.12 (see (G.1), except that we restrict to terms in $\clubsuit^{\mathsf{cf}}$) is a symbolic bisimulation.*

Proof. We only consider the case $\langle \{\!|C|\!\}\, p, \{\!|z|\!\}\, \xi_{\{z/C\}} p, SR \rangle \in R_3$. The case in which $\langle p, \xi_{\{z/C\}} p, SR \rangle \in R_3$ follows in a similar manner, and case $\langle p, q, SR \rangle \in R_{Id} \subseteq R_3$ follows from the fact that R_{Id} is a symbolic bisimulation.

The case when $C \cap FV(p) = \varnothing$ is simple. Hence, in the following we assume $C \cap FV(p) \neq \varnothing$, and check the different transfer properties in Definition 9.16.

1. Immediate.
2. Notice that $rel(\{\!|C|\!\}\, p) = rel(p)$ and $rel(\{\!|z|\!\}\, \xi_{\{z/C\}} p) = rel(\xi_{\{z/C\}} p) = (rel(p) - C) \cup \{z\}$. From this observation this case follows easily.
3. Immediate.

4. Suppose $\{\!|C|\!\}\, p \xrightarrow{a,C_1} p'$. Then $p \xrightarrow{a,C_1} p'$. By Lemma G.2.1,

   (i) $\xi_{\{z/C\}}p \xrightarrow{a,C_1} \xi_{\{z/C\}}p'$, $C \cap C_1 = \varnothing$, and $\mathsf{Lnk}(C, p')$; or

   (ii) $\xi_{\{z/C\}}p \xrightarrow{a,C_2} p'$, $C \subseteq C_1$, and $C_2 = (C_1 - C) \cup \{z\}$.

   Assume (i) is the case. Then

   (a) $\xi_{\{z/C\}}p \xrightarrow{a,C_1} \xi_{\{z/C\}}p'$.

   (b) This subitem is straightforward since $C \cap C_1 = \varnothing$ and z is fresh.

   (c) Define $SR' \stackrel{\text{def}}{=} Id_{rel(p')-(C\cap FV(p'))} \cup \{\langle C \cap FV(p'), \{z\}\rangle \mid C \cap FV(p') \neq \varnothing\}$. Clearly $\langle p', \xi_{\{z/C\}}p', SR'\rangle \in R_3$. It remains to prove that

$$\begin{aligned}&\big\{\langle C', C''\rangle \in SR \mid (C' \cap FV(p')) \cup (C'' \cap FV(\xi_{\{z/C\}}p')) \neq \varnothing\big\}\\ &- \big(\wp(C_1) \times \wp(C_1)\big)\\ &\qquad = \big\{\langle C', C''\rangle \in SR' \mid (C' \cap FV(p')) \cup (C'' \cap FV(\xi_{\{z/C\}}p')) \neq \varnothing\big\}\\ &\qquad\qquad - \big(\wp(C_1) \times \wp(C_1)\big).\end{aligned}$$

   Since $FV(p') \subseteq rel(p)$, we only have to prove that, provided $C \cap FV(p') \neq \varnothing$ then $C \cap FV(p') = C \cap FV(p)$, which follows immediately since $C \subseteq FV(p')$ and $C \subseteq FV(p)$, by Lemma G.1.

   Suppose that (ii) is the case. Then

   (a) $\xi_{\{z/C\}}p \xrightarrow{a,C_2} p'$.

   (b) Immediate since $C \subseteq C_1$, and $C_2 = (C_1 - C) \cup \{z\}$.

   (c) Define $SR' \stackrel{\text{def}}{=} Id_{rel(p')}$. Then $\langle p', p', SR'\rangle \in R_{Id} \subseteq R_3$. The rest is straightforward.

5. This transfer property is quite similar to the previous one, but uses instead the second item of Lemma G.2. *Q.E.D.*

Lemma G.4. *The relation $R_{\mathrm{A3}'}$, as specified in the proof of Theorem 11.12 (see (G.2) except that we restrict to terms in $\mathcal{P}^{\mathsf{cf}}$) is a probabilistic bisimulation up to $\sim_p$.*

Proof. By definition, $R_{\mathrm{A3}'}$ is symmetric. Thus, we directly proceed to prove the transfer properties in Definition 8.10. We only prove the straight cases. The symmetric ones follow symmetric reasoning.

In the following we will abbreviate

$$p_1 \equiv \{\!|x, y|\!\}\, (\{x\} \mapsto a; p + \{y\} \mapsto a; p) \quad \text{and} \quad p_2 \equiv \{\!|z|\!\}\, \{z\} \mapsto a; p.$$

1. Let $\langle (p_1, v), (p_2, v)\rangle \in R_{\mathrm{A3}'}$. Then,

$$\begin{array}{rcl} T(p_1, v) & = & \mathcal{D}_v^{p_1}(\mathcal{R}(F_x, F_y)) \;=\; (\Omega_1, \mathcal{F}_1, P_1) \qquad \text{and} \\ T(p_2, v) & = & \mathcal{D}_v^{p_2}(\mathcal{R}(F_z)) \;=\; (\Omega_2, \mathcal{F}_2, P_2). \end{array}$$

Define the function $\zeta : \Omega_1 \to \mathbb{R}$ by

$$\zeta([p_1, v_1]) \stackrel{\text{def}}{=} \min\{v_1(x), v_1(y)\}$$

Notice that, $\zeta([p_1, v_1]) = (\mathcal{D}_v^{p_2})^{-1}([p_2, v_2])$ if and only if $\langle [p_1, v_1], [p_2, v_2]\rangle \in R_{\mathrm{A3}'}$. Moreover, notice that ζ is actualy the random variable $\min\{x, y\}$ which is the same as z. So, take $S \subseteq [\mathbb{Q}^{\mathsf{cf}} \times \mathbf{V}]/(\sim_p \cup R_{\mathrm{A3}'})^*$, and suppose $\mu((p_1, v_1), \bigcup S) > 0$, then

$$\begin{array}{rl} \mu((p_1, v_1), \bigcup S) & = P_1(\Omega_1 \cap \bigcup S) = P(\zeta(\Omega_1 \cap \bigcup S)) \\ & = P((\mathcal{D}_v^{p_2})^{-1}(\Omega_2 \cap \bigcup S)) = P_2(\Omega_2 \cap \bigcup S) = \mu((p_2, v_2), \bigcup S) \end{array}$$

where P is the probability function of the random variable $\min\{x, y\}$.

2. Let $\langle [p_1, v_1], [p_2, v_2]\rangle \in R_{\mathrm{A3}'}$.

$$\begin{array}{ll} [p_1, v_1] \xrightarrow{a(d)} (p, v_1 - d) & \\ \Longrightarrow & \text{(by Def. 9.7, rule } \mathbf{Open}\text{)} \\ \quad p_1 \xrightarrow{a,C} p \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad \forall w \in C.\ (v_1 - d)(w) \leq 0 & \\ \Longrightarrow & \text{(by def. of } \rightarrowtail\text{, applying rules in Table 10.3)} \\ \quad (\ p_1 \xrightarrow{a,\{x\}} p \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad (v_1 - d)(x) \leq 0\ ) & \\ \quad \vee \quad (\ p_1 \xrightarrow{a,\{y\}} p \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad (v_1 - d)(y) \leq 0\ ) & \\ \Longrightarrow & \text{(by calculations)} \\ \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad (\ (v_1 - d)(x) \leq 0 \ \vee\ (v_1 - d)(y) \leq 0\ ) & \\ \Longrightarrow & \text{(by def. of } R_{\mathrm{A3}'}\text{)} \\ \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad (v_2 - d)(z) \leq 0 & \\ \Longrightarrow & \text{(by def. of } \rightarrowtail\text{, applying rules in Table 10.3)} \\ \quad p_2 \xrightarrow{a,\{z\}} p \quad \wedge \quad d \in \mathbb{R}_{\geq 0} \quad \wedge \quad (v_2 - d)(z) \leq 0 & \\ \Longrightarrow & \text{(by Def. 9.7, rule } \mathbf{Open}\text{)} \\ \quad [p_2, v_2] \xrightarrow{a(d)} (p, v_2 - d) & \end{array}$$

Since $\{x, y, z\} \cap fv(p) = \varnothing$ and for all $w \in \mathcal{C} - \{x, y, z\}$, $(v_1 - d)(w) = (v_2 - d)(w)$, by Lemma F.17, $(p, v_1 - d) \sim_p (p, v_2 - d)$. *Q.E.D.*

Bibliography

L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. *Information and Computation*, 111(1):1–52, 1994.

M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Proceedings* 18^{th} *ICALP,* Madrid, volume 510 of *Lecture Notes in Computer Science*, pages 113–126. Springer-Verlag, 1991.

R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real time. *Information and Computation*, 104:2–34, 1993.

R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

R. Alur and D. Dill. Automata for modeling real-time systems. In Paterson (1990), pages 322–335.

R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126: 183–235, 1994.

R. Alur and T.A. Henzinger. Real-time system = discrete system + clock variables. In T. Rus and C. Rattray, editors, *Theories and Experiences for Real-Time System Development — Papers presented at First AMAST Workshop on Real-Time System Development,* Iowa City, Iowa, November 1993, pages 1–29. World Scientific, 1994.

R. Alur and T.A. Henzinger, editors. *Proceedings of the 8th International Conference on Computer Aided Verification,* New Brunswick, New Jersey, USA, volume 1102 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

R. Alur, T.A. Henzinger, and E.D. Sontag, editors. *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

S. Andova. Process algebra with probabilsitic choice. In Katoen (1999), pages 111–129.

J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

J.C.M. Baeten and J.A. Bergstra. Real time process algebra with infinitesimals. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes,* Utrecht, 1994, Workshops in Computing, pages 148–187. Springer-Verlag, 1995.

J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. *Formal Aspects of Computing*, 8(2):188–208, 1996.

J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 121:234–255, 1995.

J.C.M. Baeten and J.W. Klop, editors. *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In Best (1993), pages 477–492.

J.C.M. Baeten and C. Verhoef. Concrete process algebra. In S. Abramsky, D. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol 4*. Oxford University Press, 1995.

J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.

C. Baier. *On Algorithmics Verification Methods for Probabilistic Systems*. Habilitation thesis, University of Mannheim, 1999.

C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J.C.M. Baeten and S. Mauw, editors, *Proceedings CONCUR 99,* Eindhoven, The Netherlands, volume 1664 of *Lecture Notes in Computer Science*, pages 146–161. Springer-Verlag, 1999.

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*. Springer-Verlag, 1992.

H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984.

G. Barret. Formal methods applied to a floating-point number system. *IEEE Transactions on Software Engineering*, 15(5):611–621, 1989.

J. Bengtsson, D. Griffioen, K. Kristoffersen, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using UPPAAL. In Alur and Henzinger (1996), pages 244–256.

J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In Alur et al. (1996), pages 232–243.

J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

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.

M. Bernardo. *Theory and Application of Extended Markovian Process Algebras*. PhD thesis, Dottorato di Ricerca in Informatica. Università di Bologna, Padova, Venezia, 1999.

M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202 (1-2):1–54, 1998.

E. Best, editor. *Proceedings CONCUR 93,* Hildesheim, Germany, volume 715 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

G.M. Birtwistle and C. Tofts. Process semantics for simulation. Technical report, University of Swansea, 1996.

B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.

T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. In P.H.L. van Eijk, C.A. Vissers, and M. Diaz, editors, *The formal description technique LOTOS*, pages 23–73. Elsevier Science Publishers, 1989.

T. Bolognesi and F. Lucidi. Timed process algebras with urgent interactions and a unique powerful binary operator. In de Bakker et al. (1992), pages 124–148.

S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In W.-P. de Roever, H. Langmaack, and A. Pnueli, editors, *Compositionality: The Significant Difference*, volume 1536 of *Lecture Notes in Computer Science*, pages 103–129. Springer-Verlag, 1998.

R.H. Bourgonjon. Embedded systems in consumer products. In Rozenberg and Vaandrager (1998), pages 395–403.

J. Bowen, P. Breuer, and K. Lano. A compendium of formal techniques for software maintenance. *Software Engineering Journal*, 8(5):253–262, 1993.

R. Boyer and Y. Yu. Automated proofs of object code for a widely used microprocessor. *Journal of the ACM*, 43(1):166–192, 1996.

M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: A model-checking tool for real-time systems. In A.J. Hu and M. Vardi, editors, *Proceedings of the 10th International Conference on Computer Aided Verification,* Vancouver, Canada, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer-Verlag, 1998.

S. Bradley, W. Henderson, D. Kendall, and A. Robson. Application-oriented real-time algebra. *Software Engineering Journal*, 9(5):513–521, 1994.

S. Bradley, W. Henderson, D. Kendall, and A. Robson. Verification, validation and implementation of timed protocols using AORTA. In P. Dembiński and M. Średniawa, editors, *Protocol Specification, Testing, and Verification, XV,* Warsaw, Poland, pages 205–220. Chapman & Hall, 1995.

M. Bravetti, M. Bernardo, and R. Gorrieri. Towards performance evaluation with general distributions in process algebra. In D. Sangiorgi and R. de Simone, editors, *Proceedings CONCUR 98,* Nice, France, volume 1466 of *Lecture Notes in Computer Science*, pages 405–422. Springer-Verlag, 1998.

E. Brinksma. *On the design of Extended LOTOS — A specification language for open distributed systems.* PhD thesis, Department of Computer Science, University of Twente, 1988.

E. Brinksma, J.-P. Katoen, R. Langerak, and D. Latella. A stochastic causality-based process algebra. *The Computer Journal*, 38(6):552–565, 1995.

S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

P. Buchholz. Markovian process algebra: Composition and equivalence. In Herzog and Rettelbach (1994), pages 11–30.

C.G. Cassandras. *Discrete Event Systems. Modeling and Performance Analysis.* Aksen Associates – Irwin, 1993.

Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–276, 1991.

L. Chen. *Timed Processes: Models, Axioms and Decidability.* PhD thesis, Department of Computer Science, University of Edinburgh, 1993.

T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, 1978.

G. Clark. Stochastic process algebra structure for insensitivity. In J. Hillston, editor, *Proc. of 7th International Workshop on Process Algebras and Performance Modeling, PAPM'99,* Zaragoza, Spain, 1999. To appear.

E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logic of Programs,* Yorktown Heights, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

E.M. Clarke and J.M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27(8):725–747, 1990.

J. Couvillion, R. Freire, R. Johnson, W.D. Obal, M.A. Qureshi, M. Rai, W.H. Sanders, and J.E. Tvedt. Performability modeling with ultrasan. *IEEE Software*, 8(5):69–80, 1991.

J. Crow and B. Di Vito. Formalizing space shutle software requirements: Four case studies. *ACM Transactions on Software Engineering and Methodology*, 7(3):296–332, 1998.

P.R. D'Argenio. Regular processes and timed automata. In M. Bertran and T. Rus, editors, *Proceedings of the 4th AMAST Workshop on Real-Time System Development,* Palma, Mallorca, Spain, 1997, volume 1231 of *Lecture Notes in Computer Science*, pages 141–155. Springer-Verlag, 1997a.

P.R. D'Argenio. Regular processes and timed automata. Technical Report CTIT 97-02, Department of Computer Science, University of Twente, 1997b.

P.R. D'Argenio and E. Brinksma. A calculus for timed automata. Technical Report CTIT 96-13, Department of Computer Science, University of Twente, 1996a.

P.R. D'Argenio and E. Brinksma. A calculus for timed automata (Extended abstract). In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems,* Uppsala, Sweden, volume 1135 of *Lecture Notes in Computer Science*, pages 110–129. Springer-Verlag, 1996b.

P.R. D'Argenio, H. Hermanns, and J.-P. Katoen. On generative parallel composition. *Electronic Notes in Theoretical Computer Science*, volume 22, 25 pages, 1999a.

P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. A stochastic automata model and its algebraic approach. In E. Brinksma and A. Nymeyer, editors, *Proc. of 5th International Workshop on Process Algebras and Performance Modeling, PAPM'97,* Enschede, The Netherlands, number 97-14 in Technical Report CTIT, pages 1–16. University of Twente, 1997a.

P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. An algebraic approach to the specification of stochastic systems (extended abstract). In D. Gries and W.-P. de Roever, editors, *Proceedings of the IFIP Working conference on Programming concepts and Methods, PROCOMET'98,* Shelter Island, New York, USA, IFIP Series, pages 126–147. Chapman & Hall, 1998a.

P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. A compositional approach to generalised semi-Markov processes. In *Proceedings of the 4th International Workshop on Discrete Event Systems, WODES'98,* Caligari, Italy, pages 391–387. IEE, 1998b.

P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. General purpose discrete event simulation using  . In C. Priami, editor, *Proc. of 6th International Workshop on Process Algebras and Performance Modeling, PAPM'98,* Nice, France, pages 85–102, 1998c.

P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. Specification and analysis of soft real-time systems: quantity and quality. In *Proceedings of the 1999 IEEE Real-Time Systems Symposium, RTSS'99,* Phoenix, Arizona, USA. IEEE Computer Society Press, 1999b. To appear.

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, 1997b.

P.R. D'Argenio and C. Verhoef. A general conservative extension theorem in process algebras with inequalities. *Theoretical Computer Science*, 177(2):351–380, 1997.

J. Davies et al. Timed CSP: Theory and practice. In de Bakker et al. (1992), pages 640–675.

C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In Alur et al. (1996), pages 208–219.

C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In Hogrefe and Leue (1994), pages 207–222.

C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proceedings of the 1995 IEEE Real-Time Systems Symposium, RTSS'95,* Pisa, Italy. IEEE Computer Society Press, 1995.

L. de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, 1997.

L. de Alfaro. How to specify and verify the long-run average behaviour of probabilistic systems. In LICS (1998), pages 454–465.

J.L. Devore. *Probability and Statisitics for Engineering and the Sciences.* Brooks/Cole Publishing Company, Monterey, California, 1982.

S. Eastbrook, R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton. Experiences using lightweight formal methods for requirements modeling. *IEEE Transactions on Software Engineering*, 24(1):4–14, 1997.

H. Eertink. Executing LOTOS specifications: The SMILE tool. In T. Bolognesi, J. van de Lagemaat, and C. Vissers, editors, *LOTOSphere: Software Develoments with LOTOS*, pages 221–234. Kluwer, 1995.

J.-C. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13:219–236, 1989.

J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Monier, and M. Sighireanu. CADP (Cæsar / Aldébaran Development Package): A protocol validation and verification toolbox. In Alur and Henzinger (1996), pages 437–440.

A.J. Field, P.G. Harrison, and K. Kanani. Automatic generation of verifiable cache coherence simulation models from high-level specifications. In *4th Australian Theory Symposium, CATS'98. Australian Computer Science Communications*, 20(3):261–275, 1998.

W.J. Fokkink. An elimination theorem for regular behaviours with integration. In Best (1993), pages 432–446.

W.J. Fokkink. *Clocks, Trees and Stars in Process Theory.* PhD thesis, University of Amsterdam, 1994a.

W.J. Fokkink. The tyft/tyxt format reduces to tree rules. In M. Hagiya and J.C. Mitchell, editors, *Proceedings of the International Symposium on Theoretical Aspects of Computer Software (TACS'94),* Sendai, Japan, volume 789 of *Lecture Notes in Computer Science*, pages 440–453. Springer-Verlag, 1994b.

W.J. Fokkink and R.J. van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules. *Information and Computation*, 126(1):1–10, 1996.

W.J. Fokkink and A.S. Klusener. An effective axiomatization for real time ACP. *Information and Computation*, 122(2):286–299, 1995.

A. Giacalone, C.-C. Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In M. Broy and C.B. Jones, editors, *Proceedings IFIP TC2 Working Conference on Programming Concepts and Methods,* Sea of Galilee, Israel, pages 443–458. North-Holland, 1990.

R.J. van Glabbeek. The linear time – branching time spectrum. In Baeten and Klop (1990), pages 278–297.

R.J. van Glabbeek. The linear time – branching time spectrum II (the semantics of sequential systems with silent moves). In Best (1993), pages 66–81.

R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121:59–80, 1995.

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, 1990.

R.J. van Glabbeek and F.W. Vaandrager. Petri net models for algebraic theories of concurrency. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Proceedings PARLE conference,* Eindhoven*, Vol. II (Parallel Languages)*, volume 259 of *Lecture Notes in Computer Science*, pages 224–242. Springer-Verlag, 1987.

R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

P.W. Glynn. A GSMP formalism for discrete event simulation. *Proceedings of the IEEE*, 77(1):14–23, 1989.

P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems*, volume 1032 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

P. Godefroid and P. Wolper. A partial approach to model checking. In *Proceedings* 6^{th} *Annual Symposium on Logic in Computer Science,* Amsterdam, pages 406–416. IEEE Computer Society Press, 1991.

N. Götz, U. Herzog, and M. Rettelbach. TIPP - Introduction and application to protocol performance analysis. In H. König, editor, *Formale Beschreibungstechniken für verteilte Systeme*, FOKUS series. Saur Publishers, 1993.

J.F. Groote. Specification and verification of real time systems in ACP. In L. Logrippo, R.L. Probert, and H. Ural, editors, *Protocol Specification, Testing and Verification, X,* Ottawa, Canada, 1990, pages 261–274. North-Holland, 1991.

J.F. Groote. The syntax and semantics of timed μCRL. Report SEN-R9709, CWI, Amsterdam, 1997.

J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

H.A. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, Department of Computer Systems, Uppsala University, 1991. DoCS 91/27.

H.A. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real–Time Safety Critical Systems*. Elsevier Science Publishers, 1994.

H.A. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings 11th IEEE Real-Time Systems Symposium*, pages 278–287, Lake Buena Vista, Florida, 1990.

P.G. Harrison and N.M. Patel. *Performance Modelling of Communication Networks and Computer Architectures*. Addison-Wesley, 1992.

P.G. Harrison and B. Strulo. Stochastic process algebra for discrete event simulation. In F. Bacelli, A. Jean-Marie, and I. Mitrani, editors, *Quantitative Methods in Parallel Systems*, Esprit Basic Research Series, pages 18–37. Springer-Verlag, 1995.

B.R. Haverkort and I.G. Niemegeers. Performability modelling tools and techniques. *Performance Evaluation*, 25:17–40, 1996.

A.W. Heerink. *Ins and outs in refusal testing*. PhD thesis, Department of Computer Science, University of Twente, 1998.

L. Helmink, M.P.A. Sellink, and F.W. Vaandrager. Proof-checking a data link protocol. In H. Barendregt and T. Nipkow, editors, *Proceedings International Workshop TYPES'93,* Nijmegen, The Netherlands, May 1993, volume 806 of *Lecture Notes in Computer Science*, pages 127–165. Springer-Verlag, 1994. Full version available as Report CS-R9420, CWI, Amsterdam, March 1994.

M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2): 353–389, 1995.

T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The next generation. In *Proceedings of the 1995 IEEE Real-Time Systems Symposium, RTSS'95*, pages 55–65. IEEE Computer Society Press, 1995.

T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

H. Hermanns. *Interactive Markov Chains*. PhD thesis, University of Erlangen-Nürnberg, 1998.

H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the TIPPtool. In *10th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS 98)*, volume 1496 of *Lecture Notes in Computer Science*, pages 51–62. Springer-Verlag, 1998.

H. Hermanns and M. Rettelbach. Syntax, semantics, equivalences, and axioms for MTIPP. In Herzog and Rettelbach (1994), pages 71–87.

H. Hermanns and M. Rettelbach. Towards a superset of Basic Lotos for performance prediction. In Ribaudo (1996), pages 77–93.

H. Hermanns, M. Rettelbach, and T. Weiss. Formal characterisation of immediate actions in SPA with nondeterministic branching. *The Computer Journal*, 38(7):530–541, 1995.

U. Herzog. Formal description, time and performance analysis. A framework. In T. Härder, H. Wedekind, and G. Zimmermann, editors, *Entwurf und Betrieb verteilter Systeme*, pages 172–190. Springer-Verlag, 1990.

U. Herzog and V. Mertsiotakis. Stochastic process algebras applied to failure modelling. In Herzog and Rettelbach (1994), pages 107–126.

U. Herzog and M. Rettelbach, editors. *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling, PAPM'94*. University of Erlangen, 1994.

J. Hillston. The nature of syncronisation. In Herzog and Rettelbach (1994), pages 51–70.

J. Hillston. *A Compositional Approach to Performance Modelling*. Distinguished Dissertation in Computer Science. Cambridge University Press, 1996.

P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In P. Wolper, editor, *Proceedings of the 7th International Conference on Computer Aided Verification,* Liège, Belgium, volume 939 of *Lecture Notes in Computer Science*, pages 381–394. Springer-Verlag, 1995.

C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8): 666–677, 1978.

C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, Englewood Cliffs, 1985.

D. Hogrefe and S. Leue, editors. *Proceedings of the 7^{th} International Conference on Formal Description Techniques, FORTE'94*. North-Holland, 1994.

G.J. Holzmann. The model cheker Spin. *IEEE Transactions on Software Engineering*, 23 (5):279–295, 1997.

J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Std. 1394-1995, 1996.

IEEE Computer Society. Draft Standard for a High Performance Serial Bus (Suplement). P1394a, 1998. Draft 2.0.

R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

H.E. Jensen. Model checking probabilistic real time systems. In B. Bjerner, M. Larsson, and B. Nordström, editors, *Proceedings of the 7th Nordic Workshop on Programming Theory,* Göteborg Sweden, Report 86, pages 247–261. Chalmers University of Technolog, 1996.

P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

P. Kars. Formal methods in the design of a storm barrier control system. In Rozenberg and Vaandrager (1998), pages 353–367.

J.-P. Katoen. *Quantitative and Qualitative Extensions of Event Structures.* PhD thesis, Department of Computer Science, University of Twente, 1996.

J.-P. Katoen, editor. *Proceedings of the 5th AMAST Workshop on Real-Time and Probabilistic System,* Bamberg, Germany, volume 1601 of *Lecture Notes in Computer Science.* Springer-Verlag, 1999.

J.-P. Katoen, E. Brinksma, D. Latella, and R. Langerak. Stochastic simulation of event structures. In Ribaudo (1996), pages 21–40.

L. Kleinrock. *Queuing Systems.* John Wiley & Sons, 1975.

J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol 2*, pages 1–116. Oxford University Press, 1992.

A.S. Klusener. Completeness in real time process algebra. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings CONCUR 91,* Amsterdam, volume 527 of *Lecture Notes in Computer Science*, pages 376–392. Springer-Verlag, 1991.

A.S. Klusener. *Models and axioms for a fragment of real time process algebra.* PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1993.

D.E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading, Massachusetts, 1981.

M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with probability distributions. In Katoen (1999), pages 75–95.

S. Lang. *Real and Functional Analysis*, volume 142 of *Graduate Texts in Mathematics.* Springer-Verlag, third edition, 1993.

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.

K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.

G. Leduc and L. Léonard. A formal definition of time in LOTOS. In J. Quemada, editor, *Revised draft on enhancements to LOTOS*, 1994. Annex G of document ISO/IEC JTC1/SC21/WG1/Q48.6.

N.G. Leveson and J.L. Stolzy. Safety analysis using Petri nets. *IEEE Transactions on Software Engineering*, 13(3):384–397, 1987.

Proceedings 13^{th} *Annual Symposium on Logic in Computer Science,* Indianapolis, USA. IEEE Computer Society Press, 1998.

M. Lindahl, P. Pettersson, and W. Yi. Formal design and analysis of a gear controller. In B. Steffen, editor, *Proceedings of the Fourth Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Lisbon, Portugal, volume 1384 of *Lecture Notes in Computer Science*, pages 281–297. Springer-Verlag, 1998.

N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. Report, Computing Science Institute, University of Nijmegen, 1999.

N.A. Lynch and F.W. Vaandrager. Action transducers and timed automata. *Formal Aspects of Computing*, 8(5):499–538, 1996.

O. Maler and S. Yovine. Hardware timing verification using KRONOS. In *Proceedings of the IEEE 7th Israeli Conference on Computer Systems and Software Engineering, ICCBSSE'96*. IEEE Computer Society Press, 1996.

R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

R. Milner. *Communication and Concurrency*. Prentice-Hall International, 1989.

R. Milner. The polyadic π-calculus: A tutorial. In F.L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specifications*. Springer-Verlag, 1993.

R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.

F. Moller. The importance of the left merge operator in process algebras. In Paterson (1990), pages 752–764.

F. Moller and C. Tofts. A temporal calculus of communicating systems. In Baeten and Klop (1990), pages 401–415.

V.F. Nicola, M.V. Nakayama, P. Heidelberger, and A. Goyal. Fast simulation of dependability models with general failure and repair processes. *IEEE Transactions on Computers*, 42(12):1440–1452, 1993.

X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.

X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Transactions on Software Engineering*, 18(9):794–804, 1992.

X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30(2):181–202, 1993.

J. O'Leary, X. Zhao, R. Gerth, and C.-J.H. Seger. Formally verifying IEEE compliance of floating-point hardware. *Intel Technology Software*, Q1, 1999. URL: `http://developer.intel.com/technology/itj/`.

A. Olivero. *Modélisation et Analyse de Systèmes Temporisé et Hybrides*. PhD thesis, Institut National Politechnique de Grenoble, France, 1994.

W.T. Overmans. *Verification of Concurrent Systems: Functions and Timing*. PhD thesis, University of California Los Angeles, 1981.

R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

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.

J. Parrow and B. Victor. The fusion calculus: Expressiveness and syemetry in mobile processes (extended abstract). In LICS (1998), pages 176–185.

M. Paterson, editor. *Proceedings* 17^{th} *ICALP,* Warwick, volume 443 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

J. Peterson and K. Hammond, editors. Report on the Programming Language Haskell: A Non-strict, Purely Functional Language (Version 1.4), 1997. URL: `http://haskell.org/`.

P. Pettersson. *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*. PhD thesis, Department of Computer Systems, Uppsala University, 1999.

G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

A. Pnueli and L.D. Zuck. Probabilistic verification. *Information and Computation*, 103: 1–29, 1993.

R.J. Pooley. Integrating behavioural and simulation modelling. In *Quantitative Evaluation of Computing and Communication Systems*, volume 977 of *Lecture Notes in Computer Science*, pages 102–116. Springer-Verlag, 1995.

C. Priami. Stochastic π-calculus with general distributions. In Ribaudo (1996), pages 41–57.

J.P. Queille and J. Sifakis. Specification and verification of concurrent programs in CÆSAR. In *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 195–220. Springer-Verlag, 1982.

A. Rensink. Bisimilarity of open terms. Technical Report 5/97, Institut für Informatik, Universität Hildesheim, 1997. To appear in *Information and Computation*.

M. Ribaudo, editor. *Proc. of the 4th Workshop on Process Algebras and Performance Modelling, PAPM'96,* Torino, Italy. Università di Torino, 1996.

J.M.T. Romijn. A timed verification of the IEEE 1394 leader election protocol. In S. Gnesi and D. Latella, editors, *Proceedings of the 4th International Workshop on Formal Methods for Industrial Critical Systems, FMICS'99*, pages 3–29, 1999.

G. Rozenberg and F.W. Vaandrager, editors. *Lectures on Embedded Systems*, volume 1494 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

W. Rudin. *Real and Complex Analysis*. Series in Higher Mathematics. McGraw-Hill Inc., second edition edition, 1974.

T. Ruys and R. Langerak. Validation of the Bosch' mobile comunication network architecture with Spin. In *Participant Proceedings of the third* Spin*, workshop SPIN'97,* Enschede, The Netherlands, 1997.

D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996.

R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1995.

R. Segala, R. Gawlick, J.F. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. *Information and Computation*, 141:119–171, 1998.

R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

G.S. Shedler. *Regenerative Stochastic Simulation*. Academic Press, 1993.

A.N. Shiryaev. *Probability*, volume 95 of *Graduate Texts in Mathematics*. Springer-Verlag, second edition, 1996.

J. Sifakis. Use of Petri nets for performance evaluation. In H. Beilner and E. Gelgenbe, editors, *Proceedings of the 3rd. Int. Symposium on Measuring, Modelling and Evaluating Computer Systems*, pages 75–93. North-Holland, 1977.

J. Sifakis and S. Yovine. Compositional specification of timed systems. In *Proceedings of the 13th Annual Symp. on Theoretical Aspects of Computer Science, STACS'96*, volume 1046 of *Lecture Notes in Computer Science*, pages 347–359, Grenoble, France, 1996. Springer-Verlag.

R. de Simone. On Meije and SCCS: infinite sum operators vs. non-guarded definitions. *Theoretical Computer Science*, 30:133–138, 1984.

J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 1999. To appear.

W. Stewart. MARCA – MARkov Chain Analyzer. In *Introduction to the Numerical Solution of Markov Chains*, pages 500–504. Princeton University Press, 1994.

M.I.A. Stoelinga and F.W. Vaandrager. Root contention in IEEE 1394. In Katoen (1999), pages 53–74.

A. Stoughton. Substitution revisited. *Theoretical Computer Science*, 59:317–325, 1988.

B. Strulo. *Process Algebra for Discrete Event Simulation*. PhD thesis, Department of Computing, Imperial College, University of London, 1993.

J. Tretmans. *A Formal Approach to Conformance Testing*. PhD thesis, Department of Computer Science, University of Twente, 1992.

M.Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In 26^{th} *Annual Symposium on Foundations of Computer Science, Portland, Oregon*, pages 327–338. IEEE Computer Society Press, 1985.

J.J. Vereijken. *Discrete-Time Process Algebra*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1997.

C. Verhoef. A general conservative extension theorem in process algebra. In E.-R. Olderog, editor, *Proceedings IFIP Conference of Programming Concepts, Methods and Calculi,* San Miniato, Italy, volume A-56 of *IFIP Transactions*, pages 149–168. Elsevier Science Publishers, 1994.

B. Victor. *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes*. PhD thesis, Department of Computer Systems, Uppsala University, 1998.

E.P. de Vink and J.J.M.M. Rutten. Bisimulation for probabilistic transition systems: a coalgebraic approach. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proceedings* 24^{th} *ICALP,* Bologna, volume 1256 of *Lecture Notes in Computer Science*, pages 460–470. Springer-Verlag, 1997.

W. Whitt. Continuity of generalized semi-Markov processes. *Math. Oper. Res.*, 5:494–501, 1980.

G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.

P. Wolper, M.Y. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. In 24^{th} *Annual Symposium on Foundations of Computer Science, Tucson, Arizona*, pages 185–194. IEEE Computer Society Press, 1983.

W. Yi. Real-time behaviour of asynchronous agents. In Baeten and Klop (1990), pages 502–520.

W. Yi. *A Calculus of Real Time Systems*. PhD thesis, Department of Computer Sciences, Chalmers University of Technology, 1991.

W. Yi, P. Pettersson, and M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In Hogrefe and Leue (1994), pages 223–238.

S. Yovine. *Méthodes et outils pour la vérification symbolique de systèmes temporisés*. PhD thesis, Institut National Politechnique de Grenoble, France, 1993.

H. van Zuylen, editor. *The ReDo Compendium: Reverse Engineering for Software Maintenance*. John Wiley & Sons, 1993.

Nomenclature

Numbers

$\mathbb{N}$	set of non-negative integers
i, j, k, n, m	non-negative integers
I, J, K	index sets
$\mathbb{R}$	set of real numbers
$\mathbb{R}_{\geq 0}$	set of non-negative reals
$d, d', d_i, ..$	real numbers

Probabilities

$\Omega, \Omega', \Omega_i, ..$	sample spaces
$\mathcal{F}, \mathcal{F}', \mathcal{F}_i, ..$	σ algebras
$P, P', P_i, ..$	probability measures
$F, F', F_i, ..$	distribution functions
$\mathcal{P}$	probability space
$\mathcal{B}(\mathbb{R}^n)$	Borel algebra in the nth dimensional real space
$\mathcal{R}(\dots)$	probability space in a real hyperspace generated by ...
$Prob(\_)$	set of probability spaces with sample space in _
$\text{supp}(\_)$	support set of _
$\mathcal{D}, \mathcal{D}_v^s$	decorations
$\aleph, \aleph', \aleph_i, ..$	random variables

Clocks and Constraints

$\mathcal{C}$	set of clocks
$x, y, x'x_i, ..$	clocks
$C, C', C_i, ..$	subset of clocks
Φ	set of clocks constraints
$\overline{\Phi}$	set of past closed clocks constraints
$\phi, \phi', \phi_i, ..$	clocks constraints
$\psi, \psi', \psi_i, ..$	past closed clocks constraints
$var(\_)$	clock variables in _

Valuations and Substitutions

$\mathbf{V}$	set of valuations
$v, v', v_i, ..$	valuations
$\xi, \xi', \xi_i, ..$	substitutions
id	identity substitution
ξ^1_{SR}, ξ^2_{SR}	left and right substitutions up to SR
$\xi_{\{C'/C\}}$	surjective function in $C \to C'$
$\xi_{[C'/C]}$	bijective function in $C \to C'$

Automata

Σ, Σ'	sets of states
$\sigma, \sigma', \sigma_i, ..$	states
$\mathcal{L}$	set of labels
$\ell, \ell', \ell_i, ..$	labels
$\longrightarrow$	transition relation
$\mathcal{U}$	until predicate
T	probabilistic transition
$\mathcal{S}$	set of locations
$s, s', s_i, ..$	locations
$\longrightarrow\!\!\!\!\bullet$	edge
ι	invariant assignment
κ	clock resetting/setting function
TS	transition system
TTS	timed transition system
TA	timed automaton
SA	stochastic automaton
G	GSMS
$FV(\_)$	free variables at location _

$\mu(\sigma, S)$	measure of S in $T(\sigma)$
runs	runs of a *PTS*
last	last state of a run
$\mathcal{A}$	adversary or scheduler†
reach	reachable part

Relations and Functions

s	symmetry closure
*	reflexive-transitive closure
eq	equivalence closure
Id	identity relation
$R, R', R_i, ..$	bisimulation relations
$SR, SR', SR_i, ..$	synchronisation relations
$\mathfrak{I}$	isomorphism

Equivalences

$\cong$	isomorphism
$\simeq_\alpha$	α-congruence
$\sim$	bisimulation
$\sim_t$	timed bisimulation
$\sim_p$	probabilistic bisimulation
$\sim_c$	closed p-bisimulation
$\sim_o$	open p-bisimulation
$\sim_s$	structural bisimulation
$\sim_\&$	symbolic bisimulation

Actions and Processes

$\mathcal{A}$	set of action names†
$a, b, a', a_i, ..$	action names
A	synchronisation set
$\mathcal{V}$	set of process variables
$X, Y, X', X_i, ..$	process variables
$E, E', E_i, ..$	recursive specifications
$p, q, p', p_i, ..$	processes (or terms)
$\clubsuit$	set of $\clubsuit$ terms
$\clubsuit^b$	set of sequential and closed $\clubsuit$ terms
$\clubsuit^c$	set of closed $\clubsuit$ terms
$\heartsuit$	set of $\heartsuit$ terms
$\heartsuit^{\mathsf{cf}}$	set of conflict-free $\heartsuit$ terms
$\heartsuit^\alpha$	set of α-conflict-free $\heartsuit$ terms
$\heartsuit^{br}$	set of sequential and recursive $\heartsuit$ terms
$\heartsuit^b$	set of sequential and closed $\heartsuit$ terms
$\heartsuit^c$	set of closed $\heartsuit$ terms
$\spadesuit^{\mathsf{cf}}$	set of definable $\spadesuit$ terms
$\spadesuit^\alpha$	set of α-definable $\spadesuit$ terms
$\spadesuit^b$	set of sequential and closed $\spadesuit$ terms
$\spadesuit^c$	set of closed $\spadesuit$ terms

Operations and Axiomatisations

$sig(\_)$	signature of $\_$
$ax(\_)$	axioms of $\_$
$\mathbf{0}$	nil
$a;\_$	prefix
$+$	non-deterministic summation
$\|\|_A$	parallel composition
$\lfloor\!\lfloor_A$	left merge
$\|_A$	communication merge
$\_[f]$	renaming
$\{\!\|C|\!\}\,\_$	clock resetting/setting
$\psi \gg \_$	invariant operation
$\phi \mapsto \_$	guarding operation
$C \mapsto \_$	triggering operation
$\overline{\mathsf{ck}}$	clock reset/set removing
cv	local conflict of variables
def	local definability
$fv(\_)$	free variables of $\_$
TR	trivial root predicate

Semantics

$TS(\clubsuit)$	semantics of $\clubsuit$
$TTS(TA)$	semantics of timed automata
$PTS_c(SA)$	closed behaviour of stochastic automata
$PTS_c(SA)$	open behaviour of stochastic automata
$PTS(G)$	semantics of the GSMS G
$TTS(\heartsuit^{br})$	direct semantics of $\heartsuit$
$TA(\heartsuit^{\mathsf{cf}})$	semantics of $\heartsuit$

$TA(♡^{\alpha})$	semantics of ♡ up to $\simeq_{\alpha}$
$SA(♤^{\mathsf{cf}})$	semantics of ♤
$SA(♤^{\alpha})$	semantics of ♤ up to $\simeq_{\alpha}$

† The overoading of $\mathcal{A}$ in "adversary" and "set of actions" should be harmless.

Index

Abstract

Context and Aim

Our interaction with some kind of computational-based device is nowadays unavoidable. Daily, we encounter an average of 50 microprocessors hidden in a diversity of systems, such as wrist watches, CD players, telecommunication and medical equipment, cars, and air traffic control systems. The malfunction of any of these systems has a variety of consequences, ranging from simply annoying to life threatening ones. For many of such systems, it is crucial that they provide a correct and efficient service.

In order to gain confidence that such devices satisfy our standards of service, it has been recognised that *formal analysis* has to be carried out as part of their development. In this respect, we are particularly interested in the *design* stage.

Most of the "everyday" systems require *real-time* interaction. A real-time system is a system whose behaviour is constrained by requirements concerning the occurrence of events in (real) time. These timing conditions may speak about the acceptable performance of the system or about a deadline that should be met. We can differentiate between two classes of real-time constraints: those that require that a system *must* react in time, and those that it *should* react in time but occasionally may not. If a system belongs to the first category it is referred to as a *hard real-time* system, if it belongs to the second one, as a *soft real-time* system.

The analysis of hard real-time systems requires a full exploration of its behaviour searching for undesirable situations. The violation of a timing requirement is unacceptable. There are only two options: a system is correct or not. The act of formally analysing whether a system satisfies a property or not, is known as *verification*. In contrast, the analysis of soft real-time systems requires a parameter of adequacy. The soft real-time requirements are typically concerned with the *performance* characteristics of systems and are usually related to stochastic aspects of various forms of time delay, such as the mean and variance of a message transfer delay. In addition, a stochastic study of the system also allows for *reliability* analysis, such as, average fraction of time during which the system operates correctly.

Probably, the simplest way to represent the behaviour of systems is by means of *automata*. An automaton is a graph containing nodes and directed, labelled edges. Nodes represent the possible states of a system and edges (or transitions) represent activity by joining two nodes: one being the source state, that is, the state before "executing" the

transition, and the other, the target state, which is the state reached after executing the transition. Automata have been extended in many forms and used for many purposes, including verification of systems. A way to carry out verification is by means of semantic relations between automata. That is, both the requirements and the system are specified by automata, the first one being usually simpler, and they are checked to be related by an equivalence or a preorder relation. Such a relation represents a notion of "implementation" or "conformance". We are particularly interested in so-called *bisimulation*-like equivalence relations.

In order to specify complex systems we need a *structured* approach, a systematic methodology that allows us to build large systems from the composition of smaller ones. *Process algebras* were conceived for such a hierarchical specification of systems. Each element of a process algebra represents the behaviour of a system in the same way as an automaton does. In addition, a process algebra provides operations that allow to compose behaviours in order to obtain more complex systems. As any algebra, a process algebra satisfies axioms or laws. The interest of having an axiomatisation for a process algebra is two-fold. First, the concept of algebra is fundamental in mathematics. Therefore, the given axiom system will help for the understanding of the discussed process algebra and the concepts it involves. Second, the analysis and verification of systems described using the process algebra can be partially or completely carried out by mathematical proofs using the equational theory.

In this thesis we introduce and study *process algebras* and *automata* for the *design* and *analysis* of *hard* and *soft real-time* systems.

Since the characteristics of the information we want to collect from soft real-time system is stochastic, we also refer to these systems as *stochastic systems*. For simplicity, we refer to hard real-time systems as *timed systems*.

Algebras and Automata for Timed Systems

Timed automata are a well-established model for the analysis of hard real-time systems that has been used successfully in many case studies. Inherently, timed systems induce infinite behaviours due to the representation of dense time. Timed automata propose a symbolic way to describe this infinite behaviour. Therefore, a full state space exploration can be carried out in a "symbolically finite" state space. This is one of the main reasons for the popularity of this model. Despite this popularity, no algebraic theory of timed automata has been proposed so far.

In the first part of the dissertation we review the existing theory. We give a formal definition of *timed transition system* and adapt a *timed bisimulation* to this framework (Chapter 3). Chapter 4 discusses the *timed automata* model and give semantics in terms of timed transition systems. We also define several notions of bisimulation equivalence both at a concrete and at a symbolic level.

In the subsequent chapters, the main contribution of this first part is developed: a *process algebra for timed automata* which we call ♡ (read *hearts*). Its syntax contains the same "ingredients" as the timed automata model. Its semantics is given both in terms of timed automata and timed transition systems. We also provide an axiomatic theory that allows us to manipulate algebraically timed automata. Traditional results in process algebras such as *congruence*, *soundness*, *completeness*, *elimination*, and *expansion law* are studied. Finally, Chapter 7 discusses several applications of ♡ including the specification and analysis of timed systems, and algebraic reasoning of timed automata.

Algebras and Automata for Stochastic Systems

Although many models have been successfully used for the analysis of performance and reliability of soft real-time systems, none of them provide a general and suitable model to represent systems compositionally. In the second part of the thesis, this problem is addressed.

Chapter 8 introduces *probabilistic transition systems*. This model allows for arbitrary probability distributions and hence stochastically distributed dense time can be represented in a straightforward manner. A *probabilistic bisimulation* is defined on this model. The use of probabilistic transition systems as a specification model is not attractive however since they are highly infinite objects. In Chapter 9, we define a new model based on generalised semi-Markov processes and timed automata which we call *stochastic automata*. The stochastic automata model is a general symbolic stochastic model that provides an adequate framework for composition. Its semantics is given in terms of probabilistic transition systems. We also define equivalences at the symbolic level, i.e., on the level of stochastic automata.

Several stochastic process algebras have been introduced in the last decade. The most successful ones restrict to the so-called Markovian case. General approaches to stochastic process algebras followed a less successful path, mainly due to the lack of a suitable models to represent the general case in a compositional fashion. Using stochastic automata as the underlying semantic model, we define ♤ (read *spades*), a *process algebra for stochastic automata*. This stochastic process algebra considers arbitrary distributions and non-determinism. A particular characteristic is that parallel composition and synchronisation can be easily defined in the model, and it can be algebraically decomposed in terms of more primitive operations according to a so-called *expansion law*. We remark that this last property is not present in any of the other existing general stochastic process algebras. Like for ♡, we also study results such as *congruence*, *soundness*, *completeness*, and *elimination* in the context of ♤. This is done in Chapters 10 and 11.

In order to show the feasibility of the analysis of systems specified in ♤, we report in Chapter 12 on algorithms and prototypical tools we have developed to study correctness as well as performance and reliability. The tools were used to analyse several case studies that are reported as well.

Samenvatting

Achtergrond en doel

Tegenwoordig is interactie met een of ander informatieverwerkend apparaat onvermijdelijk. Gemiddeld komen we dagelijks met ongeveer 50 microprocessoren in aanraking die zijn ingebouwd in diverse systemen, zoals pols horloges, CD spelers, telecommunicatie en medische apparatuur, auto's, en verkeersgeleidingssystemen. Het incorrect functioneren van dergelijke systemen kan diverse consequentie hebben, variërend van eenvoudigweg vervelend tot levens bedreigende scenario's. Het correct en efficiënt uitvoeren van hun taak is voor veel van dergelijke systemen cruciaal.

Om een groter vertrouwen te krijgen in het correct functioneren van zulke systemen is erkend dat *formele analyse* moet worden uitgevoerd tijdens de ontwikkeling van het systeem. Dat wil zeggen, dat wij ons in het bijzonder op het *ontwerpproces* richten.

De meeste systemen waarmee we dagelijks in aanraking komen vereisen een tijds-kritische ("real-time") interactie. Een tijds-kritisch systeem is een systeem waarin bepaalde gebeurtenissen aan tijdsgebonden eisen moeten voldoen. Zulke eisen kunnen bijvoorbeeld de accepteerbare efficiëntie van het systeem vastleggen, of een bepaalde deadline die moet worden gehaald bepalen. Twee types tijds-kritische eisen kunnen worden onderscheiden: eisen die vastleggen dat een systeem altijd op tijd *moet* reageren, en eisen die vastleggen dat een systeem op tijd *zou moeten* reageren, maar waarbij het is toegestaan dat af en toe het systeem niet op tijd reageert. Systemen die behoren tot de eerste categorie worden met "hard real-time" systemen aangeduid, en systemen die tot de tweede categorie behoren worden met "soft real-time" aangeduid.

Om ongewenste gedragingen van hard real-time systemen vast te stellen, is een volledige analyse van het systeemgedrag noodzakelijk. Het niet nakomen van een tijds-kritische eis is voor zulke systemen onacceptabel. Er zijn slechts twee opties: het systeem is of correct, of niet. Het formeel analyseren van een systeem om vast te stellen of een bepaalde eigenschap door dat systeem wordt vervuld, staat bekend als *verificatie*. Daartegenover vereist de analyse van soft real-time systemen een bepaalde mate van geschiktheid. De tijds-kritische eisen voor zulke systemen hebben vaak betrekking op hun *prestatie*-kenmerken en zijn gerelateerd aan stochastische aspecten, zoals verschillende vormen van vertragingen (bijvoorbeeld de gemiddelde vertraging die een bericht ondergaat bij het versturen, of de spreiding daarvan). Bovendien maakt een bestudering van de stochastische systeem-karakteristieken het mogelijk om een betrouwbaarheidsanalyse uit te voeren, waarbij bijvoorbeeld de gemiddelde

tijdsfractie waarin het systeem correct functioneert wordt vastgesteld.

Waarschijnlijk zijn *automaten* het eenvoudigste hulpmiddel om systeemgedrag te representeren. Een automaat is een graaf die knopen en gerichte, gelabelde pijlen bevat. De knopen representeren de toestanden waarin het systeem zich kan bevinden, en de pijlen (ook wel transities genoemd) representeren de verandering van gedrag ten gevolge van het uitvoeren van een actie: de brontoestand geeft de toestand aan voor het uitvoeren van de transitie, en de resulterende toestand geeft de toestand na de uitvoering van de transitie aan. Er bestaan verschillende uitbreidingen van automaten, en automaten zijn voor verschillende doeleinden gebruikt, bijvoorbeeld voor verificatie. Verificatie kan plaatsvinden door de semantische relatie tussen automaten te bepalen. Hierbij worden zowel het gewenste gedrag (de requirements) als het systeemgedrag gespecificeerd door middel van automaten — waarbij de eerste automaat vaak eenvoudiger is — en wordt de relatie tussen deze automaten in de vorm van een equivalentie of een pre-order vastgesteld. Zo'n relatie representeert een notie van "is een implementatie van" of "conformeert aan". Wij zijn in het bijzonder in zgn. *bisimulatie*-achtige equivalenties geinteresseerd.

Om complexe systemen te kunnen specificeren volgen we een gestructureerde aanpak, waarbij grotere systemen systematisch door de samenstelling (ook wel compositie genoemd) van kleinere systemen worden gespecificeerd. *Procesalgebras* zijn ontwikkeld voor dergelijke hierarchische beschrijvingen. Ieder element van een procesalgebra representeert het systeemgedrag op dezelfde wijze als een automaat dat doet. Bovendien stelt een procesalgebra operatoren ter beschikking die het mogelijk maken om gedragingen samen te stellen om complexere systemen te kunnen beschrijven. Zoals iedere algebra bezit een procesalgebra een aantal axioma's en wetten. Het nut van een axiomatisering voor een procesalgebra is tweeledig. Ten eerste is een algebra een fundamenteel concept in de wiskunde. Axioma's en wetten dienen dus om het begrip van een procesalgebra en haar onderliggende principes te vergroten. Ten tweede kan de analyse en verificatie van systemen wier gedrag door procesalgebra is gespecificeerd geheel of gedeeltelijk worden uitgevoerd door middel van bewijzen met behulp van de axioma's en wetten.

> In deze dissertatie introduceren en bestuderen wij *procesalgebras* en *automaten* voor het *ontwerp* en de *analyse* van *hard* en *soft real-time* systemen.

Daar wij in het bijzonder geinteresseerd zijn in de stochastische karakteristieken van soft real-time systemen, noemen we dergelijke systemen ook wel *stochastische systemen*. Hard real-time systemen duiden we ook wel eenvoudigweg aan met *timed* systemen.

Algebras en automaten voor timed systemen

Timed automaten hebben zich gemanifesteerd als een belangrijk model voor de analyse van hard real-time systemen en zijn succesvol toegepast in diverse case studies. Als tijd wordt opgevat als een continu fenomeen, dan bezit een timed systeem oneindig gedrag. Timed automaten beschrijven zulke gedragingen op een symbolische wijze. Daardoor kan

een volledige toestandsruimte-exploratie worden gerealiseerd door deze op de symbolische, eindige toestandsruimte uit te voeren. Deze eigenschap is een van de belangrijkste redenen voor de populariteit van dit model. Ondanks deze populariteit bestaat er geen algebraische theorie van timed automaten.

In het eerste deel van de dissertatie bespreken we de bestaande theorie. We geven een formele definitie van *timed transitie-systemen* en passen het begrip *timed bisimulatie* aan voor dit model (Hoofdstuk 3). Hoofdstuk 4 presenteert het *timed automaten*-model en bespreekt haar semantiek in termen van timed transitie-systemen. Bovendien definiëren we diverse noties van bisimulatie, zowel op het concrete als op het symbolische nivo.

In de daaropvolgende hoofdstukken wordt de belangrijkste bijdrage van dit eerste deel ontwikkeld: een *procesalgebra voor timed automaten*, die we ♡ (lees: hearts) noemen. De syntax van deze procesalgebra bevat dezelfde ingrediënten als het timed automaten-model. De semantiek wordt gedefinieerd in termen van timed automaten en timed transitie-systemen. We geven ook een axiomatisering die de algebraische manipulatie van timed automaten mogelijk maakt. Traditionele resultaten uit de procesalgebra, zoals *congruentie*, *correctheid*, *volledigheid*, *eliminatie* en *expansiewet* worden bestudeerd. Uiteindelijk worden in hoofdstuk 7 diverse toepassingen van ♡ besproken, waaronder de specificatie en analyse van timed systemen, en de algebraische manipulatie van timed automaten.

Algebras en automaten voor stochastische systemen

Alhoewel vele verschillende modellen met succes zijn toegepast voor het bepalen van de prestatie en betrouwbaarheid van soft real-time systemen, is geen dezer modellen geschikt om systemen compositioneel te representeren. In het tweede deel van de dissertatie wordt dit onderwerp behandeld.

Hoofdstuk 8 introduceert *probabilistische transitie-systemen*. Dit model staat het gebruik van algemene waarschijnlijksverdelingen toe, waardoor stochastisch verdeelde continue tijd eenvoudig kan worden gemodelleerd. *Probabilistische bisimulatie* is op dit model gedefinieerd. Het gebruik van probabilistische transitie-systemen als specificatie model is echter niet attractief, daar zij in hoge mate oneindig zijn. In hoofdstuk 9 definiëren we een nieuw model, dat is gebaseerd op gegeneraliseerde semi-Markov processen en op timed automaten, die we *stochastische automaten* noemen. Dit model is een symbolisch stochastisch model dat een adequaat raamwerk vormt voor compositie. De semantiek is bepaald met behulp van probabilistische transitie-systemen. Bovendien presenteren wij een aantal bisimulaties op het symbolische nivo, dat wil zeggen voor stochastische automaten.

Diverse stochastische procesalgebras zijn geintroduceerd in de afgelopen 10 jaar. De meest succesvolle beperken zich tot het gebruik van negatief exponentiële verdelingen. Algemene benaderingen waren minder succesvol, wat in grote mate kan worden verklaard door de afwezigheid van een geschikt compositioneel model voor het algemene geval. Met behulp van stochastische automaten als onderliggend semantisch model, definiëren wij ♤ (lees: *spades*), een procesalgebra voor stochastische automaten. Deze stochastische procesalgebra laat het gebruik van algemene waarschijnlijkheidsverdelingen en niet-determinisme toe. Een bijzondere eigenschap is dat parallelle compositie en synchronisatie eenvoudig

kunnen worden gedefinieerd met behulp van stochastische automaten, en dat deze operator ontleed kan worden in termen van primitievere operatoren door middel van een zgn. *expansiewet*. Met name deze laatste eigenschap is uniek en is afwezig in alle andere algemene stochastische procesalgebras. Zoals we voor ♡ hebben gedaan, onderzoeken we noties zoals *congruentie*, *correctheid*, *volledigheid* en *eliminatie* in de context van ♤. Dit wordt gerapporteerd in hoofdstukken 10 en 11.

Om de analyse van systeemspecificaties in ♤ aan te tonen, behandelt hoofdstuk 12 een aantal algoritmen en prototypes van software-gereedschappen die wij hebben ontwikkeld om zowel de correctheid als de prestatie en de betrouwbaarheid te bestuderen. Bovendien worden verschillende case studies gepresenteerd die met behulp van de software-gereedschappen zijn geanalyseerd.

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